Logical Methods in Computer Science
Vol. 12(1:7)2016, pp. 1–32
www.lmcs-online.org

Submitted Sep. 10, 2015
Published Mar. 17, 2016

HISTORY-REGISTER AUTOMATA

RADU GRIGORE [a] AND NIKOS TZEVELEKOS [b]

[a] University of Kent
e-mail address: radugrigore@gmail.com

[b] Queen Mary University of London
e-mail address: nikos.tzevelekos@qmul.ac.uk

Abstract. Programs with dynamic allocation are able to create and use an unbounded number of fresh resources, such as references, objects, files, etc. We propose History-Register Automata (HRA), a new automata-theoretic formalism for modelling such programs. HRAs extend the expressiveness of previous approaches and bring us to the limits of decidability for reachability checks. The distinctive feature of our machines is their use of unbounded memory sets (histories) where input symbols can be selectively stored and compared with symbols to follow. In addition, stored symbols can be consumed or deleted by reset. We show that the combination of consumption and reset capabilities renders the automata powerful enough to imitate counter machines, and yields closure under all regular operations apart from complementation. We moreover examine weaker notions of HRAs which strike different balances between expressiveness and effectiveness.

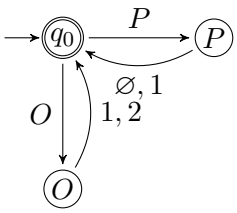
1. Introduction

Program analysis faces substantial challenges due to its aim to devise finitary methods and machines which are required to operate on potentially infinite program computations. A specific such challenge stems from dynamic generative behaviours such as, for example, object or thread creation in Java, or reference creation in ML. A program engaging in such behaviours is expected to generate a possibly unbounded amount of distinct resources, each of which is assigned a unique identifier, a *name*. Hence, any machine designed for analysing such programs is expected to operate on an infinite alphabet of names. The latter need has brought about the introduction of automata over infinite alphabets in program analysis, starting from prototypical machines for mobile calculi [27] and variable programs [23], and recently developing towards automata for verification tasks such as equivalence checks of ML programs [29, 30], context-bounded analysis of concurrent programs [10, 3] and runtime program monitoring [20].

The literature on automata over infinite alphabets is rich in formalisms each based on a different approach for tackling the infiniteness of the alphabet in a finitary manner (see e.g. [36] for an overview). A particularly intuitive such model is that of *Register*

2012 ACM CCS: [**Theory of computation**]: Formal languages and automata theory.
Key words and phrases: register automata, automata over infinite alphabets, infinite systems reachability, freshness, counter automata.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-12(1:7)2016



The automaton starts at state q_0 with empty history and non-deterministically makes a transition to state P or Q, accepting the respective symbol. From state P, it accepts any input name a which does not appear in any of its histories (this is what $\varnothing$ stands for), puts it in history number 1, and moves back to q_0. From state O, it accepts any input name a which appears in history number 1, puts it in history number 2, and moves back to q_0.

Figure 1: History-register automaton accepting $\mathcal{L}'$.

Automata (RA) [23, 31], which are machines built around the concept of an ordinary finite-state automaton attached with a fixed finite amount of registers. The automaton can store in its registers names coming from the input, and make control decisions by comparing new input names with those already stored. Thus, by talking about addresses of its memory registers rather than actual names, a so finitely-described automaton can tackle the infinite alphabet of names. Driven by program analysis considerations, register automata have been recently extended with the feature of name-freshness recognition [38], that is, the capability of the automaton to accept specific inputs just if they are *fresh* — they have not appeared before during computation. Those automata, called *Fresh-Register Automata (FRA)*, can account for languages like the following,

$$\mathcal{L}_0 = \{a_1 \dots a_n \in \mathcal{N}^* \mid \forall i \neq j.\ a_i \neq a_j\}$$

which captures the output of a fresh-name generator ($\mathcal{N}$ is a countably infinite set of names). FRAs are expressive enough to model, for example, finitary fragments of languages like the π-calculus [38] or ML [29].

The freshness oracle of FRAs administers the automata with perhaps too restricted an access to the full history of the computation: it allows them to detect name freshness, but not non-freshness. Consider, for instance, the following simple language,

$$\mathcal{L}' = \{w \in (\{O, P\} \times \mathcal{N})^* \mid \text{each letter of } w \text{ appears exactly once in it} \\ \wedge \text{ each } (O, a) \text{ in } w \text{ is preceded by some } (P, a)\}$$

where the alphabet is made of pairs containing an element from the set $\{O, P\}$ and a name (O and P can be seen as different processes or agents exchanging names). The language $\mathcal{L}'$ represents a paradigmatic scenario of a name generator P coupled with a name consumer O: each consumed name must have been created first, and no name can be consumed twice. It can capture e.g. the interaction of a process which creates new files with one that opens them, where no file can be opened twice. The inability of FRAs to detect non-freshness, as well as the fact that names in their history cannot be removed from it, do not allow them to express $\mathcal{L}'$. More generally, the notion of *re-usage* or *consumption* of names is beyond the reach of those machines. Another limitation of FRAs is the failure of closure under concatenation, interleaving and Kleene star.

Aiming at providing a stronger theoretical tool for analysing computation with names, in this work we further capitalise on the use of histories by effectively upgrading them to the status of registers. That is, in addition to registers, we equip our automata with a fixed number of unbounded sets of names (*histories*) where input names can be stored and compared with names to follow. As histories are internally unordered, the kind of

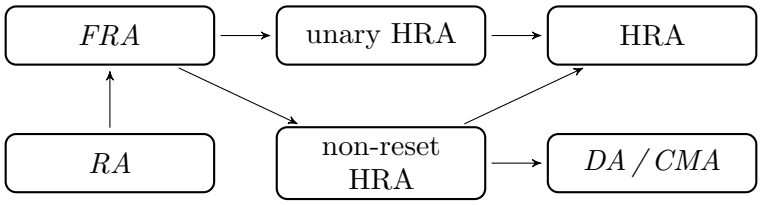


Figure 2: Expressiveness of history-register automata compared to previous models (in italics). The inclusion $\mathcal{M} \longrightarrow \mathcal{M}'$ means that for each $\mathcal{A} \in \mathcal{M}$ we can effectively construct an $\mathcal{A}' \in \mathcal{M}'$ accepting the same language as $\mathcal{A}$. All inclusions are strict.

name comparison we allow for is name belonging (*does the input name belong to the i-th history?*). Moreover, names can be selected and removed from histories, and individual histories can be emptied/reset. We call the resulting machines *History-Register Automata (HRA)*. For example, $\mathcal{L}'$ is accepted by the HRA with 2 histories depicted in Figure 1, where by convention we model pairs of symbols by sequences of two symbols.[1]

The strengthening of the role of histories substantially increases the expressive power of our machines. More specifically, we identify three distinctive features of HRAs:

(1) the capability to reset histories;
(2) the use of multiple histories;
(3) the capability to select and remove individual names from histories.

Each feature allows us to express one of the paradigmatic languages below, none of which are FRA-recognisable.

$$\mathcal{L}_1 = \{a_0 w_1 \dots a_0 w_n \in \mathcal{N}^* |\ \forall i.\ w_i \in \mathcal{N}^* \wedge a_0 w_i \in \mathcal{L}_0\} \text{ for given } a_0$$

$$\mathcal{L}_2 = \{a_1 a'_1 \dots a_n a'_n \in \mathcal{N}^* |\ a_1 \dots a_n, a'_1 \dots a'_n \in \mathcal{L}_0\}$$

$$\mathcal{L}_3 = \{a_1 \dots a_n a'_1 \dots a'_{n'} \in \mathcal{N}^* |\ a_1 \dots a_n, a'_1 \dots a'_{n'} \in \mathcal{L}_0 \wedge \forall i. \exists j.\, a'_i = a_j\}$$

Apart from the gains in expressive power, the passage to HRAs yields a more well-rounded automata-theoretic formalism for generative behaviours as these machines enjoy closure under all regular operations apart from complementation. On the other hand, the combination of features (1-3) above enables us to use histories as counters and simulate counter machines. We therefore obtain non-primitive recursive bounds for checking language emptiness. Given that language containment and universality are undecidable already for register automata [31], HRAs are fairly close to the decidability boundary for properties of languages over infinite alphabets. Nonetheless, starting from HRAs and weakening them in each of the first two factors (1,2) we obtain automata models which are still highly expressive but computationally more tractable. Overall, the expressiveness hierarchy of the machines we examine is depicted in Figure 2 (weakening in (1) and (2) respectively occurs in the second column of the figure).

Motivation and related work. The motivation for this work stems from semantics and verification. In semantics, the use of names to model resource generation originates in the work of Pitts and Stark on the ν-calculus [32] and Stark's PhD [37]. Names have subsequently been incorporated in the semantics literature (see e.g. [21, 4, 1, 24]), especially after the advent

[1] Although, technically speaking, the machines we define below do not handle constants (as e.g. O, P), the latter are encoded as names appearing in initial registers, in standard fashion.

of *Nominal Sets* [18], which provided formal foundations for doing mathematics with names. Moreover, recent work in game semantics has produced algorithmic representations of game models using extensions of fresh-register automata [29, 30, 28], thus achieving automated equivalence checks for fragments of ML and Java. In a parallel development, a research stream on automated analysis of dynamic concurrent programs has developed essentially the same formalisms, this time stemming from trace-based operational techniques [10, 3]. This confluence of different methodologies is exciting and encourages the development of stronger automata for a wider range of verification tasks, and just such an automaton we examine herein.

Although our work is driven by program analysis, the closest existing automata models to ours come from XML database theory and model checking. Research in the latter area has made great strides in the last years on automata over infinite alphabets and related logics (e.g. see [36] for an overview from 2006). As we show in this paper, history-register automata fit very well inside the big picture of automata over infinite alphabets (cf. Figure 2) and in fact can be seen as closely related to *Data Automata (DA)* [7] or, equivalently, *Class Memory Automata (CMA)* [5]. A crucial difference lies in the reset capabilities of our machines, which allow us to express languages like $\mathcal{L}_1$ that cannot be expressed by DA/CMAs. On the other hand, the local termination conditions of DA/CMAs allow them to express languages that HRAs cannot capture. We find the correspondence between HRAs and DAs particularly pleasing as it relates two seemingly very different kind of machines, with distant operational descriptions and intuitions.

A recent strand of research in foundations of atom-based computation [6, 9, 8] has examined nominal variants of classical machine models, ranging from finite-state automata to Turing machines. Finally, since the publication of the conference version of this paper [39], there has been work in *nested DA/CMAs* [14, 13], which can be seen as extensions of non-reset HRAs whereby the histories satisfy some nesting relations. The latter are a clean extension of our machines, leading to higher reachability complexities.

This article is the journal version of [39], with strengthened results and with full proofs. Section 4 is new: it collects all results that show how registers can be simulated by other means. Many upper bounds are tighter: Propositions 4.1, 4.4, 5.4, 6.2, 6.8. Some results are new: Propositions 4.5, 4.7, 5.6, 6.10. Example 4.3 is new. Most proofs have been revised. Section 7 is new: it collects in one place the main properties of HRAs. Also, we relate our work to what has been done after the conference version was published.

Overview. In Section 2 we introduce HRAs and their basic properties. In Section 3 we examine regular closure properties of HRAs. In Section 4 we explain how registers can be simulated by other means, such as histories. In Section 5 we prove that emptiness is Ackermann-complete. In Section 6 we introduce weaker models, and study their properties. In Section 7 we summarize the main properties of HRAs. In Section 8 we connect HRAs to existing automata formalisms. We conclude by discussing future directions which emanate from this work.

2. Definitions and first properties

We start by fixing some notation. Let $\mathcal{N}$ be a countably infinite alphabet of *names*, over which we range by a, b, c, etc. For any pair of natural numbers $i \leq j$, we write $[i, j]$ for the set $\{i, i{+}1, \ldots, j\}$, and for each i we let $[i]$ be the set $\{1, \ldots, i\}$. For any set S, we write $|S|$

for the cardinality of S, we write $\mathcal{P}(S)$ for the powerset of S, we write $\mathcal{P}_{\mathsf{fn}}(S)$ for the set of finite subsets of S, and we write $\mathcal{P}_{\neq\emptyset}(S)$ for the set of nonempty subsets of S. We write $\mathsf{id} : S \to S$ for the identity function on S, and $\mathsf{img}(f)$ for the image of $f : S \to T$.

We define automata which are equipped with a fixed number of ***registers*** and ***histories*** where they can store names. Each register is a memory cell where one name can be stored at a time; each history can hold an unbounded set of names. We use the term ***place*** to refer to both histories and registers. Transitions are of two kinds: name-accepting transitions and reset transitions. Those of the former kind have labels of the form (X, X'), for sets of places X and X'; and those of the latter carry labels with single sets of places X. A transition labelled (X, X') means:

- accept name a if it is contained precisely in places X, and
- update places in X and X' so that a be contained precisely in places X' after the transition (without touching other names).

By a being contained precisely in places X we mean that it appears in every place in X, and in no other place. In particular, the label $(\varnothing, X')$ signifies accepting a fresh name (one which does not appear in any place) and inserting it in places X'. On the other hand, a transition labelled by X resets all the places in X; that is, it updates each of them to the empty set (registers are modelled as sets with at most one element). Reset transitions do not accept names; they are ϵ-transitions from the outside. Note that the label $(X, \varnothing)$ has different semantics from the label X: the former stipulates that a name appearing precisely in X be accepted and then removed from X; whereas the latter clears all the contents of places in X, without accepting anything.

2.1. **Definitions.** Formally, let us fix positive integers m and n which will stand for the default number of histories and registers respectively in the machines we define below. The set Asn of ***assignments*** and the set Lab of ***labels*** are:

$$\mathsf{Asn} = \{\, H : [m+n] \to \mathcal{P}_{\mathsf{fn}}(\mathcal{N}) \mid \forall i > m.\, |H(i)| \leq 1 \,\}$$
$$\mathsf{Lab} = \mathcal{P}([m+n])^2 \cup \mathcal{P}([m+n])$$

For example, $\{(i, \varnothing) \mid i \in [m{+}n]\}$ is the empty assignment.[2] We range over elements of Asn by H and variants, and over elements of Lab by ℓ and variants.

Let $H \in \mathsf{Asn}$ be an assignment, let $a \in \mathcal{N}$ be a name, let $S \subseteq \mathcal{N}$ be a set of names, and let $X \subseteq [m+n]$ be a set of places. We introduce the following notation:

- We set $H@X$ to be the set of names which *appear precisely* in places X in H; that is, $H@X = \bigcap_{i\in X} H(i) \setminus \bigcup_{i\notin X} H(i)$. In particular, $H@\varnothing = \mathcal{N} \setminus \bigcup_i H(i)$ is the set of names which do not appear in H.
- $H[X \mapsto S]$ is the update H' of H so that all places in X are mapped to S; that is, $H' = \{(i, H(i)) \mid i \notin X\} \cup \{(i, S) \mid i \in X\}$. E.g., $H[X \mapsto \varnothing]$ resets all places in X.
- $H[a \,\mathsf{in}\, X]$ is the update of H which removes name a from all places and inserts it back in X; that is, $H[a \,\mathsf{in}\, X]$ is the assignment:

$$\{(i, H(i) \cup \{a\}) \mid i \in X \cap [m]\} \;\cup\; \{(i, \{a\}) \mid i \in X \setminus [m]\} \;\cup\; \{(i, H(i) \setminus \{a\}) \mid i \notin X\}$$

Note above that operation $H[a \,\mathsf{in}\, X]$ acts differently in the case of histories ($i \leq m$) and registers ($i > m$) in X: in the former case, the name a is added to the history $H(i)$, while in the latter the register $H(i)$ is set to $\{a\}$ and its previous content is cleared.

[2] We represent functions as sets of pairs.

We can now define our automata.

Definition 2.1. A ***history-register automaton (HRA)*** of type (m, n) is a tuple $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$ where:

- Q is a finite set of states, q_0 is the initial state, $F \subseteq Q$ are the final ones,
- $H_0 \in \mathsf{Asn}$ is the initial assignment, and
- $\delta \subseteq Q \times \mathsf{Lab} \times Q$ is the transition relation.

For brevity, we shall call $\mathcal{A}$ an (m, n)-*HRA*.

We write transitions in the forms $q \xrightarrow{X,X'} q'$ and $q \xrightarrow{X} q'$, for each kind of transition label. In diagrams, we may unify different transitions with common source and target, for example $q \xrightarrow{X,X'} q'$ and $q \xrightarrow{Y,Y'} q'$ may be written $q \xrightarrow{X,X' \,/\, Y,Y'} q'$; moreover, we shall lighten notation and write i for the singleton $\{i\}$, and ij for $\{i, j\}$.

We already gave an overview of the semantics of HRAs. This is formally defined by means of configurations representing the current computation state of the automaton. A ***configuration*** of $\mathcal{A}$ is a pair $(q, H) \in \hat{Q}$, where:

$$\hat{Q} = Q \times \mathsf{Asn}$$

From the transition relation δ we obtain the configuration graph of $\mathcal{A}$ as follows.

Definition 2.2. Let $\mathcal{A}$ be an (m, n)-HRA as above. Its ***configuration graph*** $(\hat{Q}, \longrightarrow)$, where $\longrightarrow \subseteq \hat{Q} \times \big(\mathcal{N} \cup \{\epsilon\}\big) \times \hat{Q}$, is constructed by setting $(q, H) \xrightarrow{x} (q', H')$ if and only if one of the following conditions is satisfied.

- $x = a \in \mathcal{N}$ and there is $q \xrightarrow{X,X'} q' \in \delta$ such that $a \in H@X$ and $H' = H[a \,\mathsf{in}\, X']$.
- $x = \epsilon$ and there is $q \xrightarrow{X} q' \in \delta$ such that $H' = H[X \mapsto \varnothing]$.

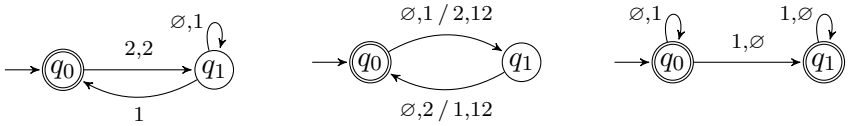
The language accepted by $\mathcal{A}$ is

$$\mathcal{L}(\mathcal{A}) = \{\, w \in \mathcal{N}^* \mid (q_0, H_0) \xrightarrow{w}\!\!\!\!\rightarrow (q, H) \text{ and } q \in F \,\}$$

where $\longrightarrow\!\!\!\!\rightarrow$ is the reflexive transitive closure of $\longrightarrow$ (i.e. $\hat{q} \xrightarrow{x_1 \ldots x_k}\!\!\!\!\rightarrow \hat{q}'$ if $\hat{q} \xrightarrow{x_1} \cdots \xrightarrow{x_k} \hat{q}'$).

Note that we use ϵ both for the empty sequence and the empty transition so, in particular, when writing sequences of the form $x_1 \ldots x_k$ we may implicitly consume ϵ's. It is worth noting here that our formulation follows *M-automata* [23] in that multiple places can be mentioned at each transition.

Example 2.3. The language $\mathcal{L}_1$ of the Introduction is recognised by the following $(1, 1)$-HRA (leftmost below), with initial assignment $\{(1, \varnothing), (2, a_0)\}$. The automaton starts by accepting a_0, leaving it in register 2, and moving to state q_1. There, it loops accepting fresh names (appearing in no place) which it stores in history 1. From q_1 it goes back to q_0 by resetting its history.



We can also see that the other two HRAs, of type $(2, 0)$ and $(1, 0)$, accept the languages $\mathcal{L}_2$ and $\mathcal{L}_3$ respectively. Both automata start with empty assignments.

Finally, the automaton we drew in Figure 1 is, in fact, a (2,2)-HRA where its two registers initially contain the names O and P respectively. The transition label O corresponds to $(3,3)$, and P to $(4,4)$.

As mentioned in the introductory section, HRAs build upon *(Fresh) Register Automata* [23, 31, 38]. The latter can be defined within the HRA framework as follows.[3]

Definition 2.4. A *Register Automaton (RA)* of n registers is a $(0,n)$-HRA with no reset transitions. A *Fresh-Register Automaton (FRA)* of n registers is a $(1,n)$-HRA $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$ such that $H_0(1) = \bigcup_i H_0(i)$ and:

- for all $(q, \ell, q') \in \delta$, there are X, X' such that $\ell = (X, X')$ and $1 \in X'$;
- for all $(q, \{1\}, X', q') \in \delta$, there is also $(q, \varnothing, X', q') \in \delta$.

Thus, in an FRA all the initial names must appear in its history, and the same holds for all the names the automaton accepts during computation ($1 \in X'$). As, in addition, no reset transitions are allowed, the history effectively contains all names of a run. On the other hand, the automaton cannot recognise *non-freshness*: if a name appearing only in the history is to be accepted at any point then a totally fresh name can be also be accepted in the same way. Now, from [38] we have the following.

Lemma 2.5. *The languages $\mathcal{L}_1, \mathcal{L}_2$ and $\mathcal{L}_3$ are not FRA-recognisable.*

Proof. $\mathcal{L}_1$ was explicitly examined in [38]. For $\mathcal{L}_2$ and $\mathcal{L}_3$ we use a similar argument as the one for showing that $\mathcal{L}_0 * \mathcal{L}_0$ is not FRA-recognisable [38] . □

2.2. **Bisimulation.** Bisimulation equivalence, also called *bisimilarity*, is a useful tool for relating automata, even from different paradigms. It implies language equivalence and is generally easier to reason about than the latter. We will be using it avidly in the sequel.

Definition 2.6. Let $\mathcal{A}_i = \langle Q_i, q_{0i}, H_{0i}, \delta_i, F_i\rangle$ be (m,n)-HRAs, for $i = 1, 2$. A relation $R \subseteq \hat{Q}_1 \times \hat{Q}_2$ is called a *simulation* on $\mathcal{A}_1$ and $\mathcal{A}_2$ if, for all $(\hat{q}_1, \hat{q}_2) \in R$,

- if $\hat{q}_1 \xrightarrow{\epsilon}\!\!\!\!\rightarrow \hat{q}_1'$ and $\pi_1(\hat{q}_1') \in F_1$ then $\hat{q}_2 \xrightarrow{\epsilon}\!\!\!\!\rightarrow \hat{q}_2'$ for some $\pi_1(\hat{q}_2') \in F_2$, where π_1 is the first projection function;
- if $\hat{q}_1 \xrightarrow{\epsilon}\!\!\!\!\rightarrow \cdot \xrightarrow{a} \hat{q}_1'$ then $\hat{q}_2 \xrightarrow{\epsilon}\!\!\!\!\rightarrow \cdot \xrightarrow{a} \hat{q}_2'$ for some $(\hat{q}_1', \hat{q}_2') \in R$.

R is called a ***bisimulation*** if both R and R^{-1} are simulations. We say that $\mathcal{A}_1$ and $\mathcal{A}_2$ are ***bisimilar***, written $\mathcal{A}_1 \sim \mathcal{A}_2$, if $((q_{01}, H_{01}), (q_{02}, H_{02})) \in R$ for some bisimulation R.

Lemma 2.7. *If $\mathcal{A}_1 \sim \mathcal{A}_2$ then $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$.*

2.3. **Determinism.** We close our presentation here by describing the deterministic class of HRAs. We defined HRAs in such a way that, at any given configuration (q, H) and for any input symbol a, there is at most one set of places X that can match a, i.e. such that $a \in H@X$. As a result, the notion of determinism in HRAs can be ensured by purely syntactic means. Below we write $q \xrightarrow{X}\!\!\!\!\rightarrow q' \in \delta$ if there is a sequence of transitions $q \xrightarrow{X_1} \cdots \xrightarrow{X_n} q'$ in δ such that $X = \bigcup_{i=1}^{n} X_i$. In particular, $q \xrightarrow{\varnothing}\!\!\!\!\rightarrow q \in \delta$.

[3] The definitions given in [23, 31, 38] are slightly different but can routinely be shown equivalent.

Definition 2.8. We say that an HRA $\mathcal{A}$ is ***deterministic*** when, for any reachable configuration $\hat{q}$ and any name a, if $\hat{q} \xrightarrow{\epsilon}\!\!\!\!\rightarrow \cdot \xrightarrow{a} \hat{q}_1$ and $\hat{q} \xrightarrow{\epsilon}\!\!\!\!\rightarrow \cdot \xrightarrow{a} \hat{q}_2$ then $\hat{q}_1 = \hat{q}_2$. We say that an HRA $\mathcal{A}$ is ***strongly deterministic*** when, for any state q and any sets X, X_1, X_2, Y_1, Y_2, if $q \xrightarrow{Y_1}\!\!\!\!\rightarrow \cdot \xrightarrow{X\setminus Y_1, X_1} q_1 \in \delta$ and $q \xrightarrow{Y_2}\!\!\!\!\rightarrow \cdot \xrightarrow{X\setminus Y_2, X_2} q_2 \in \delta$ then $q_1 = q_2$, $Y_1 = Y_2$ and $X_1 = X_2$.

Even if $\mathcal{A}$ is deterministic, it is still possible to have multiple paths in the configuration graph that are labeled by the same word. However, such paths may only differ in their ϵ-transitions. In the definition of 'strongly deterministic', the set X is guessing where the name a occurs.

Lemma 2.9. *If $\mathcal{A}$ is strongly deterministic then it is deterministic.*

3. Closure properties

History-register automata enjoy good closure properties with respect to regular language operations. In particular, they are closed under union, intersection, concatenation and Kleene star, but not closed under complementation.

In fact, the design of HRAs is such that the automata for union and intersection come almost for free through a straightforward product construction which is essentially an ordinary product for finite-state automata, modulo reindexing of places to account for duplicate labels (cf. [23]). The constructions for Kleene star and concatenation are slightly more involved as there is need for passing through intermediate automata which do not touch their initial names.

We shall need the following technical gadget. Given an (m,n)-HRA $\mathcal{A}$ and a sequence w of k distinct names, we construct a bisimilar $(m,n{+}k)$-HRA, denoted $\mathcal{A}\,\mathsf{fix}\,w$, in which the names of w appear exclusively in the additional k registers, which, moreover, remain unchanged during computation. The construction will allow us, for instance, to create feedback loops in automata ensuring that after each feedback transition the same initial configuration occurs.

Lemma 3.1. *Let $\mathcal{A}$ be an (m,n)-HRA with initial assignment H_0 and $w = a_1 \dots a_k$ a sequence of distinct names. We can effectively construct an $(m,n{+}k)$-HRA $\mathcal{A}\,\mathsf{fix}\,w$ with initial assignment H_0' such that $\mathcal{A}\,\mathsf{fix}\,w \sim \mathcal{A}$ and:*

- *$H_0'(m{+}n{+}i) = a_i$ for all $i \in [k]$, and $H_0'(i) = H_0(i) \setminus \{a_1, \dots, a_k\}$ for all $i \in [m{+}n]$;*
- *for all reachable configurations (q,H) of $\mathcal{A}\,\mathsf{fix}\,w$ and all $i > m{+}n$, $H(i) = H_0'(i)$.*

Proof. We construct $\mathcal{A}\,\mathsf{fix}\,w = \langle Q', q_0', H_0', \delta', F' \rangle$ as follows. First, we insert/move all names of w to the new registers (places $[m{+}n{+}1, m{+}n{+}k]$), i.e. we set $H_0'(i) = H_0(i) \setminus \{a_1 \dots a_k\}$ for all $i \in [m{+}n]$, and $H_0'(m{+}n{+}i) = \{a_i\}$ for each $i \in [k]$. The role of the new registers is to constantly store the names in w and act on the behalf of other places when the latter intend to use those names: during computation, whenever an a_i is captured by a transition of the initial automaton $\mathcal{A}$, in $\mathcal{A}\,\mathsf{fix}\,w$ it will be instead simulated by a transition involving the new registers. In order for the simulation to be accurate, we shall inject inside states information specifying the *intended* location of the a_is in the places of $\mathcal{A}$. Thus, the states of the new automaton are pairs (q,f), where $q \in Q$ and f is a function recording, for each of the new registers, where would the name of the register appear in the original automaton $\mathcal{A}$. That is,

$$Q' = Q \times \{f : [k] \to \mathcal{P}([m{+}n]) \mid \forall j \neq j'.\, f(j) \cap f(j') \subseteq [m]\}$$

while $q'_0 = (q_0, \{(i, \{j \mid a_i \in H_0(j)\}) \mid i \in [k]\})$ and $F' = \{(q, f) \in Q' \mid q \in F\}$. Finally, δ' operates just like δ albeit taking into account the f's of states to figure out the intended positions of the a_is and, at the same time, update the f's after each transition. We therefore include in δ' precisely the following transitions. Below we write i° for $m{+}n{+}i$. For each $(q, f) \in Q'$ and $q \xrightarrow{X,X'} q' \in \delta$,

- add a transition $(q, f) \xrightarrow{X,X'} (q', f)$;
- if $f(i) = X$ for some i then add $(q, f) \xrightarrow{\{i^\circ\},\{i^\circ\}} (q', f')$ where $f' = f[i \mapsto X']$;

Moreover, for each $q \xrightarrow{X} q' \in \delta$ include $(q, f) \xrightarrow{X} (q', f')$ where $f' = \{(j, f(j) \setminus X) \mid j \in [k]\}$. Following the above line of reasoning, we can show that the relation

$$\{((q, H), (q, f, H')) \mid \forall i \in [m{+}n].\, H(i) = H'(i) \cup \{a_j \mid i \in f(j)\}\}$$

with $(q, H), (q, f, H')$ reachable configurations, is a bisimulation. □

We write $\mathcal{L} \circ \mathcal{L}'$ for concatenation of languages, and $\mathcal{L}^*$ for Kleene closure of a language. We use the same definitions as the standard ones for languages over finite alphabets: $\mathcal{L} \circ \mathcal{L}'$ is $\{\, ww' \mid w \in \mathcal{L} \wedge w' \in \mathcal{L}' \,\}$, and $\mathcal{L}^*$ is the least fixed-point of the equations $\epsilon \in \mathcal{L}^*$ and $\mathcal{L}^* \circ \mathcal{L} \subseteq \mathcal{L}^*$, where ϵ is the empty word.

Proposition 3.2. *Languages recognised by HRAs are closed under union, intersection, concatenation and Kleene star.*

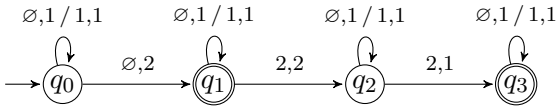
Proof. We show concatenation and Kleene star only. For the former, consider HRAs $\mathcal{A}_i = \langle Q_i, q_{0i}, H_{0i}, \delta_i, F_i \rangle$, $i = 1, 2$, and assume wlog that they have common type (m, n). Let w be an enlistment of all names in H_{02} and construct $\mathcal{A}'_i = \mathcal{A}_i \,\mathsf{fix}\, w$, for $i = 1, 2$. Then, the concatenation $\mathcal{L}(\mathcal{A}_1) \circ \mathcal{L}(\mathcal{A}_2)$ is the language recognised by connecting $\mathcal{A}'_1$ and $\mathcal{A}'_2$ serially, that is, the automaton obtained by connecting each final state of $\mathcal{A}'_1$ to the initial state of $\mathcal{A}'_2$ with a transition labelled $[m{+}n]$, and with initial/final states those of $\mathcal{A}'_1/\mathcal{A}'_2$ respectively.

Finally, given an (m, n)-HRA $\mathcal{A}$ and an enlistment w of its initial names, we construct an automaton $\mathcal{A}'$ by connecting the final states of $\mathcal{A} \,\mathsf{fix}\, w$ to its initial state with a transition labelled $[m{+}n]$. We can see that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})^*$. □

As we shall next see, while universality is undecidable for HRAs, their emptiness problem can be decided by reduction to coverability for transfer-reset vector addition systems. In combination, these results imply that HRAs cannot be effectively complemented. In fact, there are HRA-languages whose complements are not recognisable by HRAs. This can be shown via the following example, adapted from [26].

Lemma 3.3. *HRAs are not closed under complementation.*

Example 3.4. Consider $\mathcal{L}_4 = \{w \in \mathcal{N}^* \mid$ not all names of w occur exactly twice in it $\}$, which is accepted by the $(2, 0)$-HRA below.



The automaton non-deterministically selects an input name which either appears only once in the input or at least three times. We claim that $\overline{\mathcal{L}_4}$, the language of all words whose names occur exactly twice in them, is not HRA-recognisable.

Proof. Suppose it were recognisable (wlog, Proposition 4.1) by an $(m,0)$-HRA $\mathcal{A}$ with k states. Then, $\mathcal{A}$ would accept the word $w = a_1 \dots a_k\, a_1 \dots a_k$ where all a_i's are distinct and do not appear in the initial assignment of $\mathcal{A}$. Let $p = p_1 p_2$ be the path in $\mathcal{A}$ through which w is accepted, with each p_i corresponding to one of the two halves of w. Since all a_is are fresh for $\mathcal{A}$, the non-reset transitions of p_1 must carry labels of the form $(\varnothing, X)$, for some sets X. Let q be a state appearing twice in p_1, say $p_1 = p_{11}(q)p_{12}(q)p_{13}$. Consider now the path $p' = p'_1 p_2$ where p'_1 is the extension of p_1 which repeats p_{12}, that is, $p'_1 = p_{11}(q)p_{12}(q)p_{12}(q)p_{13}$. We claim that p' is an accepting path in $\mathcal{A}$. Indeed, by our previous observation on the labels of p_1, the path p'_1 does not block, i.e. it cannot reach a transition $q_1 \xrightarrow{X,Y} q_2$, with $X \neq \varnothing$, in some configuration (q_1, H_1) such that $H_1@X = \varnothing$. We need to show that p_2 does not block either (in p'). Let us denote (q, H_1) and (q, H_2) the configurations in each of the two visits of q in the run of p on w; and let us write (q, H_3) for the third visit in the run of p'_1, given that for the other two visits we assume the same configurations as in p. Now observe that, for each nonempty $X \subseteq [m]$, repeating p_{12} cannot reduce the number of names appearing precisely in X, therefore $|H_2@X| \leq |H_3@X|$. The latter implies that, since p does not block, p' does not block either. Now observe that any word accepted by w' is not in $\overline{\mathcal{L}_4}$, as p'_1 accepts more than k distinct names, a contradiction. □

4. Removing Registers

Although registers are convenient for expressing some languages (Example 4.2 and Example 4.3), when reasoning about HRAs it is more convenient to focus on histories only. In this section, we show that this approach is sound and in particular we present three ways of removing registers:

- we can construct a bisimilar automaton if we are allowed to use extra histories and resets;
- we can preserve language if we are allowed to use extra histories (but no resets);
- we can preserve emptiness if we are allowed to use extra states (but no histories nor resets).

Each of these constructions will be useful in the sequel either for devising emptiness checks and, more generally, they demonstrate that registers can be considered as a derivative notion.

4.1. Simulating Registers with Histories and Resets.

The semantics of registers is very similar to that of histories. The main difference is that registers are forced to contain at most one name. To simulate registers with histories, we reset histories before inserting names. Resetting histories might cause the automaton to forget names that are necessary for deciding how to proceed. The solution is to use two histories for each register: one holds the old name, the other holds the new name. Only the history where the new name will be written needs to be reset.

Proposition 4.1. *Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$ be an (m,n)-HRA. We can construct an $(m+2n, 0)$-HRA $\mathcal{A}' = \langle Q', q'_0, H'_0, \delta', F'\rangle$ that is bisimilar to $\mathcal{A}$. We have $|Q'| \in O(2^n|Q|)$ and $|\delta'| \in O(2^n|\delta|)$. The construction can be done in $O\big((m+n)(|Q'| + |\delta'|)\big)$ time.*

Proof. For each q in Q, we include 2^n states (q, f) in Q', where $f : [n] \to [2n]$ is such that $f(i) \in \{i, i+n\}$ for all i. The name that $\mathcal{A}$ holds in register i will be found in history $m + f(i)$ of $\mathcal{A}'$. We set $\bar{f}$ to be the complement of f; that is, $\bar{f}(i) \triangleq n + 2i - f(i)$. Let $f^\dagger(i)$ be i if $i \in [m]$ and $m + f(i)$ otherwise. Moreover, $q'_0 = (q_0, \mathsf{id})$, $\quad F' = \{(q, f) \mid q \in F\}$, and H'_0 is

H_0 extended so that $H'_0(i) = \varnothing$ for all $i > m + n$. Finally, we include in δ' precisely the following transitions.

- For each $q \xrightarrow{X,X'} q' \in \delta$, add $(q, f) \xrightarrow{Y} \cdot \xrightarrow{f^\dagger(X),\bar{f}^\dagger(X')} (q', f')$ where $Y = [m{+}1, m{+}2n] \setminus \mathsf{img}(f)$, and f' is given by: $f'(i) = \bar{f}(i)$ if $i \in X \cup X'$ and $f'(i) = f(i)$ otherwise. (Note that we need a few extra states in Q', which remain nameless in this proof.)
- For each $q \xrightarrow{X} q' \in \delta$, add $(q, f) \xrightarrow{f^\dagger(X)} (q', f)$.

The relation $\{\, ((q, H), ((q, f), H')) \mid H = H' \circ f^\dagger \,\}$ witnesses bisimilarity. □

4.2. **Replacing Registers with Histories using Colours.** The result of Proposition 4.1 comes at the cost of introducing reset transitions even if the original automaton does not have such transitions. Reset transitions are undesirable because they increase the complexity of the emptiness problem (Section 5). We can avoid introducing reset transitions by using the *colouring technique* of [5]. The construction is more involved and the resulting automaton is not bisimilar to the original, but it is language equivalent.

Before we proceed with the proof, let us illustrate the technique on two examples. Example 4.2 illustrates how to check that a name is not in the simulated register; Example 4.3 illustrates also how to check that a name is in the simulated register.

Example 4.2. The language

$$\mathcal{L}_5 \;=\; \{\, a_1 \dots a_n \mid a_i \neq a_{i+1} \text{ for all } i \,\}$$

is recognized by both of the automata below (with histories initially empty):

The one on the left is a $(0, 1)$-HRA, and we can straight away see its accepted language is $\mathcal{L}_5$. The one on the right is a $(2, 0)$-HRA for which it is less clear why it accepts $\mathcal{L}_5$. The reason is the following invariant:

- $H(1) \uplus H(2)$ is a partition of all names seen so far; and
- if the state is q_k then the last seen name is in $H(k + 1)$, for $k \in \{0, 1\}$; and
- all possible partitions of the names are realisable.

The first two points are easy to check. Because all transitions have labels of the form $(X, \{i\})$, all seen names are remembered in precisely one history. Because all transitions incoming into q_k have labels the form $(X, \{k + 1\})$, the last seen name is remembered in $H(k + 1)$. The consequence of these first two points is that the automaton on the right accepts a name only if it is different from the last seen name. Indeed, all transitions outgoing from q_k have labels of the form (X, X') with $k + 1 \notin X$. Thus, the first two points should be seen as lemmas which let us establish that all words accepted by the automaton on the right belong to $\mathcal{L}_5$.

Informally, the third point is the key lemma that lets us establish the converse, that all words in $\mathcal{L}_5$ are accepted. However, to see why this is so, we need to rephrase it in a more formal way. Given a word $a_1 \dots a_n \in \mathcal{L}_5$, we consider an arbitrary partition $H(1) \uplus H(2)$ of the set $\{a_1, \dots, a_n\}$. Without loss of generality, assume $a_n \in H(1)$. The claim is that

there exists a run of the automaton on the right that accepts the word $a_1 \dots a_n$ and ends in configuration (q_0, H). We can prove this by induction. If $a_{n-1} \in H(k+1)$ then the previous state in the run must have been q_k, for $k \in \{0, 1\}$. Suppose $a_{n-1} \in H(2)$; the other case is symmetric. Then, by the induction hypothesis, we know that there is a run that accepts the word $a_1 \dots a_{n-1}$ and ends in configuration (q_1, H'), where we choose

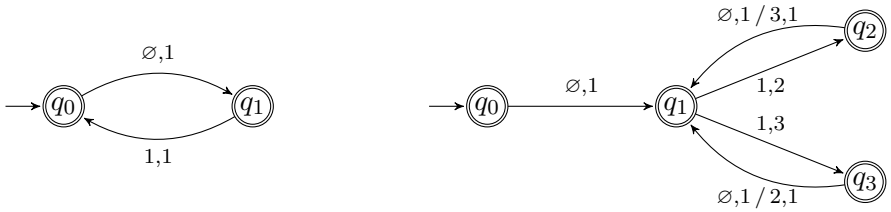
$$H'(1) \triangleq \begin{cases} H(1) \setminus \{a_n\} & \text{if } a_n \notin \{a_1, \dots, a_{n-1}\} \\ H(1) \cup \{a_n\} & \text{if } a_n \in \{a_1, \dots, a_{n-1}\} \end{cases} \qquad H'(2) \triangleq H(2) \setminus \{a_n\}$$

It is clear that $H'(1) \uplus H'(2)$ is a partition of $\{a_1, \dots, a_{n-1}\}$, as required. Finally, we need to show that $(q_1, H') \xrightarrow{a_n} (q_0, H)$ belongs to the configuration graph. If $a_n \notin \{a_1, \dots, a_{n-1}\}$, then this is true because of $q_1 \xrightarrow{\varnothing,1} q_0$; if $a_n \in \{a_1, \dots, a_{n-1}\}$, then this is true because of $q_1 \xrightarrow{1,1} q_0$.

Example 4.3. The language

$$\mathcal{L}_6 \;=\; \{\, a_1 \dots a_n \mid a_i = a_{i+1} \text{ iff } i \text{ is odd} \,\}$$

is recognized by both of the following automata:



The one on the left is a $(0, 1)$-HRA, while the one on the right is a $(3, 0)$-HRA. As in the previous example, the fact that the $(3, 0)$-HRA accepts $\mathcal{L}_6$ is not immediately clear. Informally, the reason is the following invariant:

- $H(1) \uplus H(2) \uplus H(3)$ is a partition of the names seen so far;
- if the state is q_k then the last seen name is in $H(k)$, for $k \in \{1, 2, 3\}$;
- $|H(1)| = 1$ in q_1, and $|H(1)| = 0$ otherwise; and
- for each partition $H(1) \uplus H(2) \uplus H(3)$ that is compatible with the previous constraints, there is a nondeterministic run that realises it.

The first two points hold for the same reasons as in Example 4.2. The third point holds because all incoming transitions of q_1 insert a name in $H(1)$, all outgoing transitions of q_1 remove a name from $H(1)$, and all runs alternate between state q_1 and some other state. After an odd number of names was processed, the automaton is in state q_1 and $H(1)$ contains only the last name seen. All outgoing transitions from q_1 accept a name only if it is in $H(1)$ — in other words, if it equals the last seen name. After an even and positive number of names was processed, the automaton is in state q_2 or q_3. All outgoing transitions from q_2 accept a name only if it is *not* in $H(2)$, where the last seen name is; q_3 acts symmetrically. Thus, the first three points let us establish that all words accepted by the automaton on the right belong to $\mathcal{L}_6$.

As in Example 4.2, the last point lets us establish that all word in $\mathcal{L}_6$ are accepted. Given that the proof of the last point is very similar to the one in Example 4.2, let us only sketch it. Given a word $a_1 \dots a_n \in \mathcal{L}_6$ with $n > 0$, we consider an arbitrary partition $H(1) \uplus H(2) \uplus H(3)$ of the set $\{a_1, \dots, a_n\}$ such that $H(1) \subseteq \{a_n\}$. Let k be such that $a_n \in H(k)$. Formally, the claim of the last point is that there exists a run of the automaton

on the right that is labelled by the word $a_1 \dots a_n$ and ends in the configuration (q_k, H). The case n odd and > 1 is similar to the previous example: We pick k' such that $a_{n-1} \in H(k')$ and

$$H'(1) \triangleq \emptyset \quad H'(k') \triangleq H(k') \setminus \{a_n\} \quad H'(5-k') \triangleq \begin{cases} H(5-k') \setminus \{a_n\} & \text{if } a_n \notin \{a_1, \dots, a_{n-1}\} \\ H(5-k') \cup \{a_n\} & \text{if } a_n \in \{a_1, \dots, a_{n-1}\} \end{cases}$$

Then we invoke the induction hypothesis to show there is a run that accepts $a_1 \dots a_{n-1}$ and ends in configuration $(q_{k'}, H')$. In the case n even, we pick

$$H'(1) \triangleq \{a_n\} \qquad H'(2) \triangleq H(2) \setminus \{a_n\} \qquad H'(3) \triangleq H(3) \setminus \{a_n\}$$

and then invoke the induction hypothesis to show that there is a run that accepts $a_1 \dots a_{n-1}$ and ends in configuration (q_1, H'). We skip the case $n = 1$ in this proof sketch.

We are now ready for the general result, which we prove in two steps. The main construction will be presented first and only applies to HRAs with initially empty registers. (The correctness of the main construction requires that certain graphs are 2-colourable. The arguments in the previous two examples can be seen as giving explicit colouring algorithms that work in special cases.) At a second stage, we show how to initially simulate nonempty registers at the expense of some more additional histories.

Proposition 4.4. *Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$ be an (m,n)-non-reset-HRA with registers initially empty. We can construct an $(m+3n, 0)$-non-reset-HRA $\mathcal{A}' = \langle Q', q_0', H_0', \delta', F'\rangle$ that accepts the same language as $\mathcal{A}$. We have $|Q'| \in O(2^{2n} \cdot |Q|)$ and $|\delta'| \in O(2^{3.3n} \cdot |\delta|)$.*

Proof. Each register i will be simulated by three histories, named i_{R}, i_{B} and i_{Y} respectively.[4] Each state $q \in Q$ will be simulated by several states $(q, f) \in Q'$, where $f : [m+1, m+n] \to \{\varnothing, \mathrm{R}, \mathrm{B}, \mathrm{Y}\}$. The construction will ensure the following invariant:

- $H(i_{\mathrm{R}}) \uplus H(i_{\mathrm{B}}) \uplus H(i_{\mathrm{Y}})$ is a partition of the names that have been written to register i and have subsequently been rewritten by other names or are still in register i;[5]
- $|H(i_{\mathrm{R}})| = 1$ if $f(i) = \mathrm{R}$, and $|H(i_{\mathrm{R}})| = 0$ otherwise; and
- if $f(i) \neq \varnothing$ then the current name of register i is in $H(i_{f(i)})$; register i is empty otherwise.

Thus, according to the last point above, f records in which of histories $i_{\mathrm{R}}, i_{\mathrm{B}}, i_{\mathrm{Y}}$ has the current name of register i been stored. We shall instrument $\mathcal{A}'$ in such a way that it will only store that name, say a, in i_{R} if the next time register i is being invoked by $\mathcal{A}$ is for reading a. This way we shall ensure that i_{R} never contains more than one name. Otherwise, i.e. if $\mathcal{A}$ next invokes i for overwriting its contents, f will be mapping i to one of $i_{\mathrm{B}}, i_{\mathrm{Y}}$. These can be seen as garbage collecting histories: they contain all names that have passed from register i and will not be immediately read from it. The reason why we need two of these, i_{B} and i_{Y}, is to be able to reuse old names of register i without running the risk of confusing them with its current name a.

To simulate one transition $q \xrightarrow{X,X'} q'$, accepting say a name a, we shall use several transitions of the form $(q, f) \xrightarrow{Z,Z'} (q', f')$ where f and f' agree outside $X \cup X'$. Let us consider an arbitrary such pair (f, f'), and see how to pick Z and Z'. On histories, X and Z coincide: $X \cap [m] = Z \cap [m]$. For each register $i \in X \setminus [m]$, we need that a be equal to the

[4] B and Y are the black and yellow colours of [5]; R stands for 'read'.

[5] note that a name can also be transferred out of register i (via transition with label X, Y where $i \in X \setminus Y$), instead of being directly rewritten, in which case we would not store it in $H(i_{\mathrm{R}}) \uplus H(i_{\mathrm{B}}) \uplus H(i_{\mathrm{Y}})$.

name currently written in register i. But, we know the current name in register i only if $f(i) = \mathrm{R}$. That is why we include a transition $(q, f) \xrightarrow{Z,Z'} (q', f')$ only if $X \setminus [m] \subseteq f^{-1}(\mathrm{R})$, and we include $\{\, i_{\mathrm{R}} \mid i \in X \setminus [m] \,\}$ in Z. For each register $i \in [m+1, m+n] \setminus X$, we must ensure that a is not equal to the current name in register i, which resides in $H(i_{f(i)})$. Hence, we pick all Z such that

$$Z = (X \cap [m]) \uplus Z_1 \uplus Z_0$$

where $Z_1 = \{\, i_{\mathrm{R}} \mid i \in X \setminus [m] \,\}$ and $Z_0 \subseteq \{\, i_x \mid i \in [m+1, m+n] \setminus X \wedge x \in \{\mathrm{B}, \mathrm{Y}\} \wedge x \neq f(i) \,\}$ is such that, for all i, $|Z_0 \cap \{i_{\mathrm{B}}, i_{\mathrm{Y}}\}| \leq 1$. Now we must write the current name to histories $X' \cap [m]$, and we must simulate writing the current name to registers $X' \setminus [m]$. For each $i \in X' \setminus [m]$ we shall *nondeterministically* write the current name to one of $i_{\mathrm{R}}, i_{\mathrm{B}}, i_{\mathrm{Y}}$ by *guessing* whether register i will be used next for reading or not. The place where we write is given by $f'(i)$, that is,

$$Z' = (X' \cap [m]) \uplus \{\, i_{f'(i)} \mid i \in X' \setminus [m] \,\}.$$

Finally, we make sure that all $i \in ([m+1, m+n] \cap X) \setminus X'$ in f' are mapped to $\varnothing$, as these registers are now empty, i.e. we impose $f'(([m+1, m+n] \cap X) \setminus X') \subseteq \{\varnothing\}$.

Thus, in summary, we take $Q' = Q \times ([m+1, m+n] \to \{\varnothing, \mathrm{R}, \mathrm{B}, \mathrm{Y}\})$ and:

- $q'_0 = (q_0, \{(i, \varnothing) \mid i \in [m+1, m+n]\})$ and $F' = F \times ([m+1, m+n] \to \{\varnothing, \mathrm{R}, \mathrm{B}, \mathrm{Y}\})$;
- $H'_0 = H_0 \cup \{(i, \varnothing) \mid i \in [m+n+1, m+3n]\}$;
- we include $(q, f) \xrightarrow{Z,Z'} (q', f')$ in δ' just if there is some $q \xrightarrow{X,X'} q'$ in δ such that:
  - $X \setminus [m] \subseteq f^{-1}(\mathrm{R})$,
  - for all $i \in [m+1, m+n] \setminus (X \cup X')$, $f'(i) = f(i)$,
  - for all $i \in ([m+1, m+n] \cap X) \setminus X'$, $f'(i) = \varnothing$,
  - $Z = (X \cap [m]) \uplus \{i_{\mathrm{R}} \mid i \in X \setminus [m]\} \uplus Z_0$
    with $Z_0 \subseteq \{i_x \mid i \in [m+1, m+n] \setminus X \wedge x \neq f(i)\}$,
  - $Z' = (X' \cap [m]) \uplus \{\, i_{f'(i)} \mid i \in X' \setminus [m] \,\}$.

Let us now see what is the size of the HRA $\mathcal{A}'$ so constructed. We have $|Q'| \in O(4^n \cdot |Q|)$. For each transition $q \xrightarrow{X,X'} q'$ in $\mathcal{A}$, we introduce several transitions $(q, f) \xrightarrow{Z,Z'} (q' f')$ in $\mathcal{A}'$. Let us count how many. There are $\leq 4^n$ choices for f; there are $\leq 3^n$ choices for Z_0 because for each i we pick i_{B}, or i_{Y}, or none of them; and there are $\leq 3^n$ choices for f' because $f'(X') \subseteq \{i_{\mathrm{R}}, i_{\mathrm{B}}, i_{\mathrm{Y}}\}$ and f' is uniquely determined outside X'. In summary, $|\delta'| \in O(2^{2(1+\log_2 3)n} \cdot |\delta|)$.

Finally, we show that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Let first $w \in \mathcal{L}(\mathcal{A}')$ have an accepting transition path p' in $\mathcal{A}'$ with edges $(q_k, f_k) \xrightarrow{Z_k,Z'_k} (q_{k+1}, f_{k+1})$, for $k = 1, \ldots, N$. Reading the definition of δ' backwards, this yields an accepting transition path p in $\mathcal{A}$ with edges $q_k \xrightarrow{X_k,X'_k} q_{k+1}$ where

- $X_k = (Z_k \cap [m]) \cup \{\, i \mid i_{\mathrm{R}} \in Z_k \,\}$,
- $X'_k = (Z'_k \cap [m]) \cup \{\, i \mid \{i_{\mathrm{R}}, i_{\mathrm{B}}, i_{\mathrm{Y}}\} \cap Z'_k \neq \emptyset \,\}$.

To see that p accepts w, suppose that p' yields a sequence of configurations $((q_k, f_k), H'_k)$. Then, by induction, we can show that p yields a sequence of configurations (q_k, H_k), where:

- for all $i \in [m]$, $H_k(i) = H'_k(i)$;
- for all $i \in [m+1, m+n]$, if $f_k(i) \neq \varnothing$ then $H_k(i) = \{a\}$ for some $a \in H'_k(i_{f_k(i)})$, otherwise $H_k(i) = \varnothing$;
- for all names a, if $a \in H'_k @ Z_k$ then $a \in H_k @ X_k$.

Hence, $w \in \mathcal{L}(\mathcal{A})$.

Conversely, let $w = a_1 \cdots a_N \in \mathcal{L}(\mathcal{A})$ have an accepting transition path p in $\mathcal{A}$ with edges $q_k \xrightarrow{X_k, X'_k} q_{k+1}$ for $k = 1, \ldots, N$. We construct a corresponding accepting path p' in $\mathcal{A}'$ with edges $(q_k, f_k) \xrightarrow{Z_k, Z'_k} (q_{k+1}, f_{k+1})$ as follows. We have that $f_0 = \{(i, \varnothing) \mid i \in [m]\}$. Moreover, $Z_k = (X_k \cap [m]) \cup W_k$ and $Z'_k = (X'_k \cap [m]) \cup W'_k$ where:

(a) For each position k such that the previous appearance of a_k in w is some $a_{k'} = a_k$ with $k' < k$, we set $W_k = W'_{k'}$. If there is no previous appearance, we set $W_k = \varnothing$.

(b) For each position k and $i \in X'_k \setminus [m]$ such that the next appearance of a_k in w is some $a_{k'}$ with $i \in X_{k'}$, we include i_{R} in W'_k.

(c) For each position k and $i \in X'_k \setminus [m]$ such that the next appearance of a_k in w is some $a_{k'}$ with $i \notin X_{k'}$, we include in W'_k one of $i_{\mathrm{B}}, i_{\mathrm{Y}}$. We do the same also if there is no next appearance of a_k in w.

The above specifications determine the values of all W_k, W'_k, modulo the choice between B and Y in case (c). Clearly, if the path p' can be so constructed then $\mathcal{A}'$ accepts w. It remains to show that p' can indeed be implemented in $\mathcal{A}'$. The form of the f_k's is derived from (a-c) according to the definition of δ'. But note that the definition of δ' imposes the following condition:

(d) For each position k and $i_{\mathrm{Y}} \in W'_k$ such that the next appearance of any i_x in p' is in some $W_{k'}$ (with $k < k'$), we must have $i_{\mathrm{B}} \in W_{k'}$. Dually if $i_{\mathrm{B}} \in W'_k$.

For example, if $i_{\mathrm{Y}} \in W'_2$ but none of $i_{\mathrm{B}}, i_{\mathrm{Y}}, i_{\mathrm{R}}$ occurs in any of $W_3, W'_3, W_4, W'_4, W_5, W'_5$, then $\{i_{\mathrm{B}}, i_{\mathrm{Y}}, i_{\mathrm{R}}\} \cap W_6 \subseteq \{i_{\mathrm{B}}\}$. This condition stems from the interdiction to include $i_{f(i)}$ in $W_{k'}$ when $f(i) \neq i_{\mathrm{R}}$. The consequence of condition (d) is that in case (c) above we cannot pick B and Y arbitrarily.

We need to show that a choice of "colours" (B and Y) satisfying both (c) and (d) can be made. We achieve this by applying a graph colouring argument. Let us define a labelled graph $\mathcal{G}$ with:

- Vertices (k, i) and $(k, i)'$ for each $k \in [0, N-1]$ and $i \in [m+1, m+n]$;
- For each i and $k < k'$ as in (c) above, an edge between $(k, i)'$ and (k', i) labelled with "=".
- For each i and $k < k'$ as in (d) above, an edge between $(k, i)'$ and (k', i) labelled with "$\neq$".

Then, a valid choice of colours can be made as long as $\mathcal{G}$ can be coloured with B and Y in such a way that =-connected vertices have matching colours, while $\neq$-connected vertices have different colours. For the latter, it suffices to show that the graph obtained by merging =-connected vertices can be 2-coloured, for which it is enough to show that $\mathcal{G}$ contains no cycles. Suppose $\mathcal{G}$ contained a cycle. Then, by definition of the edge relation of the graph, it must be the case that the leftmost vertex (i.e. the one with the least k index) in the cycle be some $(k, i)'$. The vertex $(k, i)'$ has two outgoing edges, one for each label. The $\neq$-edge in particular connects to some (k', i) such that $k < k'$, obtained from condition (d). Since (k', i) is part of the cycle, it must have an outgoing =-edge to some vertex $(k'', i)'$ with $k'' < k'$. But note that condition (d) stipulates that there is no mention of i between k and k' in p', and therefore $k'' \leq k$. Moreover, $k'' = k$ is not an option as it would imply that register i was not rewritten between steps k and k' in p, in which case k and k' would fall under case (b) above. Hence, $k'' < k$ which contradicts our assumption that $(k, i)'$ was the leftmost vertex in the cycle. □

Note that the above result can be extended to handle the case in which registers are not initially empty by simply making use of Lemma 3.1. However, the construction in that lemma leads to a doubly exponential blow-up in size, which we can be avoided by the alternative approach that follows.

Proposition 4.5. *Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$ be an (m,n)-non-reset-HRA. We can construct a bisimilar $(m+n,n)$-non-reset-HRA $\mathcal{A}' = \langle Q', q_0', H_0', \delta', F'\rangle$ such that, for all $i \in [m+n+1, m+2n]$, $H_0(i) = \varnothing$. Moreover, we have $|Q'| \in O(2^n \cdot |Q|)$ and $|\delta'| \in O(2^{2n} \cdot |\delta|)$.*

Proof. The main idea behind the construction of $\mathcal{A}'$ is to use the additional n histories to store just the initial names of the registers in $\mathcal{A}$. Once these names have been used in the computation, they are transferred to their actual registers (if any). We will also need to track which of the registers in $\mathcal{A}$ are still simulated by histories in $\mathcal{A}'$. Thus, we set

$$Q' = Q \times ([m+1, m+n] \to \{0,1\})$$

and $q_0' = (q_0, \{(i,1) \mid i \in [m+1, m+n]\})$, $H_0' = H_0 \cup \{(m+n+i, \varnothing) \mid i \in [n]\}$ and $F' = F \times ([m+1, m+n] \to \{0,1\})$. Moreover, for each $q \xrightarrow{X,X'} q'$ in δ and map f, we include in δ' a transition $(q,f) \xrightarrow{Y,Y'} (q', f')$ where:

- $Y = (X \cap [m]) \uplus Y_1 \uplus Y_0$, where
$$Y_1 = \{\, i \in X \mid f(i) = 1 \,\} \cup \{\, n+i \mid i \in X \wedge f(i) = 0 \,\}$$
and $Y_0 \subseteq \{\, i \in [m+1, m+n] \mid i \notin X \wedge f(i) = 0 \,\}$;
- $Y' = (X' \cap [m]) \cup \{\, n+i \mid i \in X' \setminus [m] \,\}$;
- $f' = f[i \mapsto 0 \mid i \in X \cup X']$.

Then, taking R to be the relation:

$$\begin{aligned} R = \{\, ((q,H),((q,f),H') \mid\ & H \restriction [m] = H' \restriction [m] \wedge \forall i \in [m+1, m+n]. \\ & \wedge f(i) = 1 \implies H'(n+i) = \varnothing \wedge H'(i) = H(i) \\ & \wedge f(i) = 0 \implies H'(n+i) = H(i) \\ & \wedge \forall j \in [m+1, m+n].\, H'(i) \cap H'(n+j) = \varnothing \,\} \end{aligned}$$

we can show that R is a bisimulation.

Let us now see what is the size of the HRA $\mathcal{A}'$ we constructed. We have $|Q'| \in O(2^n \cdot |Q|)$. For each transition $q \xrightarrow{X,X'} q'$ in $\mathcal{A}$, we introduce several transitions $(q,f) \xrightarrow{Y,Y'} (q'f')$ in $\mathcal{A}'$: there are $\leq 2^n$ choices for f; and there are $\leq 2^n$ choices for Y_0. In summary, $|\delta'| \in O(2^{2n} \cdot |\delta|)$. □

Hence, the general case follows.

Corollary 4.6. *Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$ be an (m,n)-non-reset-HRA. We can construct an $(m+4n, 0)$-non-reset-HRA $\mathcal{A}' = \langle Q', q_0', H_0', \delta', F'\rangle$ that accepts the same language as $\mathcal{A}$. We have $|Q'| \in O(2^{3n} \cdot |Q|)$ and $|\delta'| \in O(2^{5.3n} \cdot |\delta|)$.*

4.3. **Simulating Registers Symbolically.** So far we saw how to simulate registers using histories. If we are interested only in emptiness/reachability rather than language equivalence, we can actually simulate the behaviour of registers without the inclusion of additional histories. This alternative is going to be crucial in Section 6.2, where the number of histories will be fixed to just one.

We next describe how this simulation can be done. Given an assignment H with m histories and n registers, we can represent H *symbolically* as follows:

- we map each name stored in the registers of H to a number from the set $[n]$;
- we subsequently replace in H all these names by their number.

For example, consider the assignment

$$\big\{1 \mapsto \{a, b, c\},\ 2 \mapsto \{d\},\ 3 \mapsto \varnothing,\ 4 \mapsto \{a\},\ 5 \mapsto \{d\}\big\}$$

of a $(1, 4)$-HRA. We can simulate it symbolically by mapping d to 1, and a to 2. This results to a symbolic representation:

$$\big\{1 \mapsto \{2, b, c\},\ 2 \mapsto 1,\ 3 \mapsto \varnothing,\ 4 \mapsto 2,\ 5 \mapsto 1\big\}$$

where the nominal part has been curtailed to the fact that $H(1)$ contains the names b and c.

We can now employ this representation technique to represent configurations of (m, n)-HRAs by corresponding ones belonging to $(m, 0)$-HRAs. In particular, given the configuration

$$\Big(q,\ \big\{1 \mapsto \{a, b, c\},\ 2 \mapsto \{d\},\ 3 \mapsto \varnothing,\ 4 \mapsto \{a\},\ 5 \mapsto \{d\}\big\}\Big)$$

of a $(1, 4)$-HRA, we map it to the configuration

$$\Big(\big(q, \big\{1 \mapsto \{2\},\ 2 \mapsto 1,\ 3 \mapsto \varnothing,\ 4 \mapsto 2,\ 5 \mapsto 1\big\}\big),\ \big\{1 \mapsto \{b, c\}\big\}\Big)$$

of a $(1, 0)$-HRA which incorporates the non-nominal part of our representation scheme in its state. Clearly, the state space of the new automaton in this simulation will experience an exponential blowup, as the next result shows. However, no additional histories will be needed, which is the main target here.

Proposition 4.7. *Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$ be an (m, n)-HRA. We can construct an $(m, 0)$-HRA $\mathcal{A}' = \langle Q', q_0', H_0', \delta', F'\rangle$ that is empty if and only if $\mathcal{A}$ is empty. We have $|Q'| \in O(2^{mn} n B_n |Q|)$ and $|\delta'| \in O(2^{mn} n B_n |\delta|)$, where B_n is the nth Bell number. Moreover, $\mathcal{A}'$ contains reset transitions if and only if $\mathcal{A}$ contains reset transitions.*

Proof. Each state $q \in Q$ will be simulated by several states $(q, f) \in Q'$, where $f : [m + n] \to \mathcal{P}([n])$ will be called an assignment *skeleton*. Such a skeleton f is *valid* when:

- $|f(i)| \leq 1$ for registers $i \in [m + 1, m + n]$;
- $f(i) \subseteq \bigcup_{j=1}^{n} f(m + j)$ for histories $i \in [m]$;
- for all $k \in [n]$ there is a $k' \in [k]$ such that $\bigcup_{i=1}^{k} f(m + i) = [k']$.

The latter condition essentially stipulates that f is a partition function on the set $[m+1, m+n]$: the elements of the set are uniquely assigned numbers which can be seen as class indices — two elements are assigned the same number iff they belong to the same class. There is also a special class in this partition, namely of all elements of $[m + 1, m + n]$ to which f assigns $\varnothing$.

We can now define the rest of $\mathcal{A}'$. First, we let $q_0' = (q_0, f_0)$, where (f_0, H_0') is the symbolic representation of H_0. In order to construct δ' we define a transition relation on skeletons, which is very similar to the configuration graph of HRAs (Definition 2.2) except that it allows symbols to be permuted after the transition is taken. We write $f \xrightarrow{X,X'} f'$ when

there exists a permutation π on $[n]$ and a $k \in [n]$ such that $k \in f@X$ and $f' = \pi \circ (f[k \,\mathsf{in}\, X'])$. We write $f \xrightarrow{X} f'$ when there exists a permutation π such that $f' = \pi \circ (f[X \mapsto \varnothing])$.

To simulate one transition of the form $q \xrightarrow{X,X'} q'$ from δ, we use several transitions of the form $(q, f) \xrightarrow{\ell} (q', f')$ in δ'. Let us consider an arbitrary pair (f, f') of valid skeletons, and see how to pick ℓ. There are four cases, depending on whether X and X' mention or not registers.

- Case $X \subseteq [m]$ and $X' \subseteq [m]$. It must be that $f \xrightarrow{\varnothing,\varnothing} f'$, and we pick $\ell = (X, X')$.
- Case $X \subseteq [m]$ and $X' \not\subseteq [m]$. It must be that $f \xrightarrow{\varnothing,X'} f'$, and we pick $\ell = (X, \varnothing)$.
- Case $X \not\subseteq [m]$ and $X' \subseteq [m]$. It must be that $f \xrightarrow{X,\varnothing} f'$, and we pick $\ell = (\varnothing, X')$.
- Case $X \not\subseteq [m]$ and $X' \not\subseteq [m]$. It must be that $f \xrightarrow{X,X'} f'$, and we pick $\ell = (\varnothing, \varnothing)$.

Similarly, each reset transition $q \xrightarrow{X} q'$ from δ yields several transitions of the form $(q, f) \xrightarrow{Z} (q', f')$ in δ'. Given an arbitrary pair (f, f') of valid skeletons, we pick Z as follows. If $X \subseteq [m]$ then it must be that $f' = f$ and we pick $Z = X$. Otherwise, $f \xrightarrow{X} f'$ and we pick $Z = \varnothing$.

To estimate $|Q'|$ it suffices to count how many valid skeletons there are. The values $f(m+1), \ldots, f(m+n)$ of a valid skeleton correspond to a partition of the registers and a selection of a class (if any) whose registers are empty. There are B_n possible partitions and $\leq (n+1)$ possible selections, which gives $\leq (n+1)B_n$ cases. For the values $f(1), \ldots, f(m)$ of a valid skeleton there are $\leq 2^{mn}$ possibilities. In total, $|Q'| \leq 2^{mn}(n+1)B_n|Q|$.

To estimate $|\delta'|$, note that once f is fixed in the construction above, the constraints on f' determine it uniquely. So, the number of transitions increases by the same factor as the number of states. □

Since $\log B_n \in \Theta(n \log n)$, we have that $\log\big(2^{mn}(n+1)B_n\big) \in \Theta(mn + n \log n)$.

5. Emptiness and Universality

5.1. **Emptiness.** Here we show that deciding emptiness is Ackermann-complete. We work by reducing from and to state reachability problems in counter systems (similarly e.g. to [7, 5]). For the upper bound, we reduce nonemptiness of HRAs to control-state reachability of T-VASSs. For the lower bound, we reduce control-state reachability of R-VASSs to nonemptiness of HRAs. Recall that the ***nonemptiness problem*** for HRAs asks, given an HRA $\mathcal{A}$ with initial state q_0 and initial assignment H_0, whether $(q_0, H_0) \xrightarrow{w} (q_F, H_F)$ for some word w, final state q_F and assignment H_F.

The configurations of a k-dimensional TR-VASS (Transfer–Reset Vector Addition System with States) have the form $(q, \vec{v})$, where q is a state from a finite set, and $\vec{v}$ is a k-dimensional vector of nonnegative counters. A VASS has moves that shift the counter vector, changing $\vec{v}$ into $\vec{v} + \vec{v}\,'$, where $\vec{v}\,'$ comes from some finite and fixed subset of $\mathbb{Z}^k$. An R-VASS also has moves that reset a counter, changing $\vec{v}$ into $\vec{v}[i \mapsto 0]$ for some counter i. A T-VASS also has moves that transfer the content of one counter into another counter, changing $\vec{v}$ into $\vec{v}[j \mapsto \vec{v}(i) + \vec{v}(j)][i \mapsto 0]$ for some $i \neq j$. A TR-VASS is the obvious combination of the above, and is formally defined as follows.

Definition 5.1 (TR-VASS)**.** A k-dimensional ***Transfer-Reset Vector Addition System with States*** $\mathcal{A}$ is a pair $\langle Q, \delta\rangle$, where Q is a finite set of states, and $\delta \subseteq Q \times (\mathbb{Z}^k \uplus [k]^2 \uplus [k]) \times Q$ is a transition relation. A ***configuration*** of $\mathcal{A}$ is a pair $(q, \vec{v})$ of a state q and a vector $\vec{v} \in \mathbb{N}^k$ of counter values. The ***configuration graph*** of $\mathcal{A}$ is constructed by including an arc $(q, \vec{v}) \to (q', \vec{v}\,')$ when one of the following holds:

- there is some $(q, \vec{v}\,'', q') \in \delta$ such that $\vec{v}\,' = \vec{v} + \vec{v}\,''$
- there is some $(q, (i, j), q') \in \delta$ such that $\vec{v}\,' = \vec{v}[i \mapsto 0][j \mapsto \vec{v}(i) + \vec{v}(j)]$ and $i \neq j$
- there is some $(q, (i, i), q') \in \delta$ and $\vec{v}\,' = \vec{v}$
- there is some $(q, i, q') \in \delta$ such that $\vec{v}\,' = \vec{v}[i \mapsto 0]$.

The ***control-state reachability problem*** for $\mathcal{A}$ asks whether, given states q_0, q_F and initial vector $\vec{v}_0$, is there some $\vec{v}_F$ such that $(q_0, \vec{v}_0) \longrightarrow\!\!\!\!\!\rightarrow (q_F, \vec{v}_F)$.

The reduction from a $(m, 0)$-HRA to a T-VASS of dimension $2^m - 1$ is done by mapping each nonempty set of histories X into a counter $\widetilde{X}$ of the corresponding T-VASS. Then, name-accepting transitions are mapped into counter decreases and increases, while resets result in transfers between the counters. Let $\widetilde{\cdot} : \mathcal{P}\big([m]\big) \to [0, 2^m - 1]$ be a bijection such that $\widetilde{\emptyset} = 0$; for instance, one could take $\widetilde{X} \triangleq \sum_{i \in X} 2^{i-1}$. Further, given an assignment H, let $\widetilde{H}$ denote the vector $(h_1, \ldots, h_{2^m-1}) \in \mathbb{N}^{2^m-1}$ such that $h_{\widetilde{X}} = |H@X|$, for all nonempty $X \subseteq [m]$; that is, $h_{\widetilde{X}}$ counts how many names occur in exactly the histories indexed by X.

Lemma 5.2. *Given an $(m, 0)$-HRA $\mathcal{A}$ it is possible to construct a T-VASS $\mathcal{A}'$ of dimension $2^m - 1$ such that, for all q, q', H, H',*

$$\exists w,\ (q, H) \xrightarrow{w}\!\!\!\!\!\rightarrow_{\mathcal{A}} (q', H') \qquad \text{if and only if} \qquad (q, \widetilde{H}) \twoheadrightarrow_{\mathcal{A}'} (q', \widetilde{H'}).$$

Let Q and δ be the states and the transitions of $\mathcal{A}$, and let Q' and δ' be the transitions of $\mathcal{A}'$. We have that $|Q'| \in O(2^m|Q|)$ and $|\delta'| \in O(2^m|\delta|)$. Moreover, the construction takes $O(|Q'| + m|\delta'|)$ time. If there are no reset transitions in $\mathcal{A}$, then $\mathcal{A}'$ is a $|\delta|$-dimensional VASS with $Q' = Q$ and $|\delta'| = |\delta|$ that uses only increments and decrements.

Let $\vec{0}$ be the all-zero vector $(0, \ldots, 0)$. Let $\vec{\delta}_i$ be $\vec{0}[i \mapsto 1]$ for $i \in [m]$, and $\vec{\delta}_0$ be $\vec{0}$.

Proof. For each transition $q \xrightarrow{X, X'} q'$ of the HRA, we construct a transition $q \xrightarrow{\vec{\delta}_{\widetilde{X}'} - \vec{\delta}_{\widetilde{X}}} q'$ in the T-VASS. For each transition $q \xrightarrow{X} q'$ of the HRA, we construct a path

$$q \xrightarrow{1, j_1} \cdot \xrightarrow{2, j_2} \cdot \xrightarrow{3, j_3} \cdots \xrightarrow{2^m - 2, j_{2^m-2}} \cdot \xrightarrow{2^m - 1, j_{2^m-1}} q'$$

in the T-VASS such that $j_{\widetilde{Y}} = \widetilde{Y \setminus X}$. To construct such a path we iterate through $2^m - 1$ nonempty sets Y, and for each we compute $Y \setminus X$ in $O(m)$ time. □

Lemma 5.2 implies that nonemptiness of a HRA reduces to control-state reachability of a T-VASS. We shall describe an algorithm that solves control-state reachability for the T-VASS constructed in Lemma 5.2. The analysis of this algorithm depends on the so-called Length Function Theorem, which is phrased in terms of the Fast Growing Hierarchy and bad sequences. We define these next.

The Fast Growing Hierarchy consists of classes $\mathcal{F}_0$, $\mathcal{F}_1$, $\mathcal{F}_2$, ... of functions, where $\mathcal{F}_0 = \mathcal{F}_1$ contain the linear functions, $\mathcal{F}_2$ contains the elementary functions, primitive

recursive functions are in $\mathcal{F}_k$ for some finite k, and $\mathcal{F}_\omega$ is the ACKERMANN complexity class. The classes $\mathcal{F}_k$ are defined in terms of the following functions:

$$F_0(x) \triangleq x+1 \qquad F_{n+1}(x) \triangleq (\underbrace{F_n \circ \cdots \circ F_n}_{x+1 \text{ times}})(x) = F_n^{x+1}(x) \qquad F_\omega(x) \triangleq F_x(x)$$

For $k \geq 2$, (a) $f \in \mathcal{F}_k$ if and only if $f \in O(F_k^n)$ for some n; and (b) a nondeterministic algorithm using space bounded by some function in $\mathcal{F}_k$ can be transformed into a deterministic algorithm using time bounded by some (other) function in $\mathcal{F}_k$.

Let X be a partially ordered set with some size function $|\cdot| : X \to \mathbb{N}$. We say that a sequence $x_0, x_1, x_2, \ldots$ of elements of X is a ***bad sequence*** when $x_i \not\leq x_j$ for all $i < j$. Given a strictly increasing function $g : \mathbb{N} \to \mathbb{N}$, we say that the sequence is ***controlled*** by g when $|x_{i+1}| \leq g(|x_i|)$ for all i. We will consider such sequences of VASS configurations, where the order is given by

$$(q, \vec{v}) \leq (q', \vec{v}') \qquad \text{iff} \qquad q = q' \ \wedge\ \vec{v}(1) \leq \vec{v}'(1) \ \wedge\ \vec{v}(2) \leq \vec{v}'(2) \ \wedge\ \vec{v}(3) \leq \vec{v}'(3) \ \wedge\ \ldots$$

Lemma 5.3 (Length Function Theorem [34])**.** *Let $\hat{q}_0, \hat{q}_1, \hat{q}_2, \ldots$ be a bad sequence of k-dimensional VASS configurations. If the sequence is controlled by some function $g \in \mathcal{F}_\gamma$ with $\gamma \geq 1$, then its length is bounded by $f(|\hat{q}_0|)$ for some function $f \in \mathcal{F}_{\gamma+k}$.*

We can now describe and analyze an algorithm for deciding emptiness of a $(m, 0)$-HRA.

Proposition 5.4. *The emptiness problem for $(m, 0)$-HRAs is in $\mathcal{F}_{2^m}$ when $m > 0$. Thus, the emptiness problem is in $\mathcal{F}_\omega$ when m is part of the input.*

Proof. Let $\mathcal{A}$ be the given HRA, and let $\mathcal{A}'$ be the T-VASS constructed as in Lemma 5.2. We use the backward coverability algorithm [34, Sections 1.2.2 and 2.2.2], which explores all bad sequences $(q_0, \vec{v}_0),\ (q_1, \vec{v}_1),\ \ldots,\ (q_L, \vec{v}_L)$ such that

- $(q_0, \vec{v}_0)$ is a minimal final configuration, and
- $(q_k, \vec{v}_k)$ is a minimal configuration out of those that can reach a configuration $\geq (q_{k-1}, \vec{v}_{k-1})$.

The constraint that $(q_0, \vec{v}_0)$ is a minimal final configuration simply means that q_0 is final and $\vec{v}_0 = \vec{0}$. To construct such sequences, we need an effective way of generating all possible $(q_k, \vec{v}_k)$, given a fixed $(q_{k-1}, \vec{v}_{k-1})$. For this, we enumerate all transitions of $\mathcal{A}'$ that go to q_{k-1}. There are two types of such transitions: $q_k \xrightarrow{\vec{\delta}_i - \vec{\delta}_j} q_{k-1}$ and $q_k \xrightarrow{i,j} q_{k-1}$. For $q_k \xrightarrow{\vec{\delta}_i - \vec{\delta}_j} q_{k-1}$, we let $\vec{v}_k$ be $\max(\vec{v}_{k-1} - \vec{\delta}_i + \vec{\delta}_j, \vec{0})$, where max is taken pointwise. For $q_k \xrightarrow{i,j} q_{k-1}$ with $i = j$, we take $\vec{v}_k$ to equal $\vec{v}_{k-1}$. For $q_k \xrightarrow{i,j} q_{k-1}$ with $i \neq j$, we may have multiple choices for $\vec{v}_k$. Assuming $\vec{v}_{k-1}(i) = 0$, it could be that $\vec{v}_k(i)$ is any of $0, 1, \ldots, \vec{v}_{k-1}(j)$; otherwise, if $\vec{v}_{k-1}(i) \neq 0$, the transition could not have been taken. In all the cases from above, we keep only those choices of $\vec{v}_k$ that ensure the sequence is bad.

To show that the sequences so constructed are finite, we use Lemma 5.3. Let $|(q, \vec{v})|$ be the number bits in a concrete representation of $(q, \vec{v})$: we encode q with $\sim \log_2 |Q'| = m \log_2 |Q|$ bits, then we write each $\vec{v}(i)$ in binary, precede each of its bits by 1 and mark the end with 0. (For example, we represent 5 by 1110110.) For the sequences constructed as in the previous paragraph, we have $|(q_k, \vec{v}_k)| \leq 2 \cdot |(q_{k-1}, \vec{v}_{k-1})|$. Thus, the sequences are controlled by $g(x) = 2x$, which is a function in $\mathcal{F}_1$. As $\mathcal{A}'$ has dimension $2^m - 1$, Lemma 5.3 gives us that the length L of the sequence is bounded by some function in $\mathcal{F}_{2^m}$.

A nondeterministic algorithm can repeatedly guess the correct successor in the sequence, using $2^L \cdot |(q_0, \vec{0})|$ space. If $m \geq 1$, then this is bounded by $f(m \log |Q|)$ for some function

$f \in \mathcal{F}_{2^m}$, and we are in a situation where the distinctions time/space and deterministic/nondeterministic are irrelevant. □

It is possible to modify the algorithm described in the previous proof so that it works directly on the HRA representation, without appealing to Lemma 5.2. Similarly, it is possible to extend the algorithm described in the previous proof to handle registers directly, without appealing to Proposition 4.1. Such improvements may be worthwhile in an implementation, but the complexity upper bound remains Ackermannian.

Doing the opposite reduction we show that deciding emptiness is Ackermann-hard even for strongly deterministic HRAs. In this direction, each R-VASS of dimension m can be simulated by an $(m, 0)$-HRA so that the value of each counter i of the former is the same as the number of names appearing precisely in history i of the latter. In order to extend the bound to strongly deterministic HRAs one can choose to reduce from a restricted class of R-VASSs, so that the image of the reduction can be made strongly deterministic, or resolve nondeterminacy at the level of HRAs by appropriate obfuscation. We follow the latter, simpler solution.

Proposition 5.5. *The emptiness problem for strongly deterministic HRAs is* Ackermann*-hard.*

Proof. Let $\mathcal{A}$ be an m-dimensional R-VASS whose additive transitions only increment or decrement single counters: for each transition $q \xrightarrow{\vec{v}} q'$, we have $\vec{v} = \pm\vec{\delta}_i$ for some i. By [35], control-state reachability for such R-VASSs is Ackermann-hard. We construct an $(m, 0)$-HRA $\mathcal{A}'$ with the same states as $\mathcal{A}$, and we map: each $q \xrightarrow{\delta_i} q'$ to $q \xrightarrow{\varnothing,\{i\}} q'$, each $q \xrightarrow{-\delta_i} q'$ to $q \xrightarrow{\{i\},\varnothing} q'$, and each $q \xrightarrow{i} q'$ to $q \xrightarrow{\{i\}} q'$. We can see that $\mathcal{A}'$ simulates the behaviour of $\mathcal{A}$ by storing the value of each counter i as $|H@\{i\}|$. Hence, $\mathcal{L}(\mathcal{A}')$ is nonempty if and only if q_F is reachable, from $(q_0, \vec{v}_0)$, in the R-VASS $\mathcal{A}$.

We observe that $\mathcal{A}'$ may not be strongly deterministic. Suppose that the size of the transition function of $\mathcal{A}$ is n. We can then impose strong determinacy on $\mathcal{A}'$ by enriching it with n registers and preluding each transition of the above translation with a transition reading from one of the additional registers. We thus obtain an (m, n)-HRA that is strongly deterministic and simulates $\mathcal{A}$ as above. □

Proposition 5.6. *The emptiness problem of HRAs is* Ackermann*-complete.*

Proof. Combine Proposition 4.1 with Proposition 5.4 and Proposition 5.5. □

5.2. **Universality.** We finally consider universality and language containment. Note first that our machines inherit undecidability of these properties from register automata [31]. However, these properties are decidable in the deterministic case.

In order to simplify our analysis, we shall be reducing HRAs to the following compact form where ϵ-transitions are incorporated inside name-accepting ones. As we show below, no expressiveness is lost by this packed form.

A *packed* $(m, 0)$*-HRA* is a tuple $\mathcal{A} = \langle Q, q_0, \delta, H_0, F\rangle$ defined exactly as an $(m, 0)$-HRA, with the exception that now:

$$\delta \subseteq Q \times \mathcal{P}([m]) \times \mathcal{P}([m]) \times \mathcal{P}([m]) \times Q$$

We shall write $q \xrightarrow{Y;X,X'} q'$ for $(q, Y, X, X', q') \in \delta$. The semantics of such a transition is the same as that of a pair of transitions $q \xrightarrow{Y} \cdot \xrightarrow{X,X'} q'$ of an ordinary HRA. Formally, configurations of packed HRAs are pairs (q, H), like in HRAs, and the configuration graph of a packed HRA $\mathcal{A}$ like the above is constructed as follows. We set $(q, H) \xrightarrow{a} (q, H')$ if there is some $q \xrightarrow{Y;X,X'} q'$ in δ such that, setting $H_Y = H[Y \mapsto \varnothing]$, we have $a \in H_Y @ X$ and $H' = H_Y[a \,\mathsf{in}\, X']$.

Lemma 5.7. *Let $\mathcal{A}$ be an $(m, 0)$-HRA. There is a packed $(m, 0)$-HRA $\mathcal{A}'$ such that $\mathcal{A} \sim \mathcal{A}'$.*

Proof. Let $\mathcal{A} = \langle Q, q_0, \delta, H_0, F\rangle$. We set $\mathcal{A}' = \langle Q, q_0, \delta', H_0, F'\rangle$ where:

$$F' = \{q' \in Q \mid \exists q \in F, Y.\ q' \xrightarrow{Y} q \in \delta\}$$

$$\delta' = \{(q, Y, X, X', q') \mid q \xrightarrow{Y} \cdot \xrightarrow{X,X'} q' \in \delta\}$$

Bisimilarity of $\mathcal{A}$ and $\mathcal{A}'$ is witnessed by the identity on configurations, which means that $R = \{\, ((q, H), (q, H)) \mid q \in Q \wedge H \in \mathsf{Asn} \,\}$ is a bisimulation. □

We shall decide language containment via complementation. In particular, given a deterministic packed HRA $\mathcal{A}$, the automaton $\mathcal{A}'$ accepting the language $\mathcal{N}^* \setminus \mathcal{L}(\mathcal{A})$ can be constructed in the analogous way as for deterministic finite-state automata, namely by obfuscating the automaton with all missing transitions and swapping final with non-final states.

Lemma 5.8. *Deterministic packed HRAs are closed under complementation.*

Proof. Let $\mathcal{A} = \langle Q, q_0, \delta, H_0, F\rangle$ be a packed $(m, 0)$-HRA. Following the above rationale, we construct a packed $(m, 0)$-HRA $\mathcal{A}' = \langle Q \uplus \{q_F\}, q_0, \delta \cup \delta', H_0, F'\rangle$, where $F' = \{q_F\} \cup (Q \setminus F)$ and δ' is given as follows. For each $q \in Q$ and all X such that there is no $q \xrightarrow{Y;X\setminus Y,X'} q'$ add a transition $q \xrightarrow{\varnothing;X,\varnothing} q_F$ in δ'. In addition, δ' contains a transition $q_F \xrightarrow{[m];\varnothing,\varnothing} q_F$.
We claim that $\mathcal{L}(\mathcal{A}') = \mathcal{N}^* \setminus \mathcal{L}(\mathcal{A})$. Indeed, if $s \in \mathcal{L}(\mathcal{A}')$ and s is accepted at a state in $Q \setminus F$ then, since $\mathcal{A}$ is deterministic, we have $s \notin \mathcal{L}(\mathcal{A})$. Otherwise, if $s = s'as''$ with a the point where a transition to the sink state is taken then, upon acceptance of s' by $\mathcal{A}$, a appears precisely in some histories X such that $\mathcal{A}$ has no transition to accept a at that point. Thus, $s \notin \mathcal{L}(\mathcal{A})$.
Conversely, if $s \in \mathcal{N}^* \setminus \mathcal{L}(\mathcal{A})$ then either s induces a configuration in $\mathcal{A}$ which does not end in a final state, or $s = s'as''$ where s' is accepted by $\mathcal{A}$ but at that point a is not a possible transition. We can see that, in each case, $s \in \mathcal{L}(\mathcal{A}')$. □

Proposition 5.9. *Language containment and universality are undecidable for (general) HRAs and* Ackermann*-complete for strongly deterministic HRAs.*

Proof. Undecidability in the general case is inherited from RAs.

Now consider two HRAs $\mathcal{A}$ and $\mathcal{A}'$ such that we can compute the complement of $\mathcal{A}'$. Then, we can decide the language containment $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$ by checking whether the product of $\mathcal{A}$ with the complement of $\mathcal{A}'$ is empty. The product construction is polynomial, and the emptiness check is in Ackermann (Proposition 5.4). Thus, language containment is in Ackermann if computing the complement of $\mathcal{A}'$ is in Ackermann. This is the case because (a) removing registers can be done while preserving determinism with only an exponential increase in size (Proposition 4.1), and (b) complementing deterministic HRAs

without registers takes polynomial time (Lemma 5.8). For hardness, note that emptiness and universality are equally hard in the deterministic case (Lemma 5.8), and emptiness is Ackermann-hard (Proposition 5.5).

We showed that language containment is in Ackermann and universality is Ackermann-hard. Finally, note that there is a trivial reduction from universality to language containment. □

6. Weakening HRAs

Since the complexity of HRAs is substantially high, e.g. for deciding emptiness, it is useful to seek for restrictions thereof which allow us to trade expressiveness for efficiency. As the encountered complexity stems from the fact that HRAs can simulate computations of R-VASSs, our strategy for producing weakenings is to restrict the functionalities of the corresponding R-VASSs. We follow two directions:

(a) We remove reset transitions. This corresponds to removing counter transfers and resets and drops the complexity of control-state reachability to exponential space.
(b) We restrict the number of histories to just one. We thus obtain polynomial space complexity as the corresponding counter machines are simply one-counter automata. This kind of restriction is also a natural extension of FRAs with history resets.

Observe that each of the aspects of HRAs targeted above corresponds to features (1,2) we identified in the Introduction, witnessed by the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. We shall see that each restriction leads to losing the corresponding language.

6.1. Non-reset HRAs.

We first weaken our automata by disallowing resets. We show that the new machines retain all their closure properties apart from Kleene-star closure. The latter is concretely manifested in the fact that language $\mathcal{L}_1$ of the Introduction is lost. On the other hand, the emptiness problem reduces in complexity to exponential space.

Definition 6.1. A ***non-reset HRA*** of type (m, n) is an (m, n)-HRA $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$ such that there is no $q \xrightarrow{X} q' \in \delta$.

Closure properties. Of the closure constructions of Section 3 we can see that union and intersection readily apply to non-reset HRAs, while the construction for concatenation needs some amendments.

More specifically, of the two constructions presented in the proof of Proposition 3.2, the one for concatenation can be adapted to non-reset HRAs as follows. We add empty transitions from the final states of $\mathcal{A}'_1$ to the initial state of a version of $\mathcal{A}'_2$ which keeps the places used by $\mathcal{A}'_1$ untouched and uses its own separate copy of places, obfuscating its own transitions so as to capture accidental matchings of the legacy names of $\mathcal{A}'_1$. This solution cannot be used for Kleene closure as in each loop the automaton needs to find a fresh copy of its initial configuration, and be able to use it (in the previous construction, the final assignment of $\mathcal{A}'_1$ is lost).

On the other hand, using an argument similar to that of [5, Proposition 7.2], we can show that the language $\mathcal{L}_1$ is not recognised by non-reset HRAs and, hence, the latter are not closed under Kleene star. Finally, note that the HRA constructed for the language $\mathcal{L}_4$

in Example 3.4 is a non-reset HRA, which implies that non-reset HRAs are not closed under complementation.

Emptiness. In the general case we saw an upper bound of $\mathcal{F}_{2^m}$ (Proposition 5.4), by a reduction to T-VASS followed by the backward coverability algorithm. For a non-reset HRA, the same reduction yields a VASS, without transfers. In the absence of transfers, better bounds are known for the backward coverability algorithm [11]. More generally, it has been known for some time that coverability for VASS is ExpSpace-complete [12, 25].

The following result refers to the number N of bits used to represent an HRA. Of course, N depends on the exact representation being used. Still, we do not make this representation explicit because the result holds for a wide variety of possible representations. We only require that the representation obeys $m, n, |\delta|, \log|Q| \in O(N)$.

Proposition 6.2. *The emptiness problem for a non-reset HRA is in* ExpSpace. *More precisely, it is in* $\textsc{NSpace}\big(2^{O(N \log N)}\big)$, *where N is the number of bits used to represent the HRA.*

Proof. We start with an (m,n)-non-reset-HRA $\mathcal{A} = \langle Q, q_0, \delta, H_0, F\rangle$. We use Proposition 4.7 to construct an $(m,0)$-non-reset-HRA $\mathcal{A}' = \langle Q', q_0', \delta', H_0', F'\rangle$ that preserves emptiness. Moreover, $\log|Q'| \in O(mn + n\log n + \log|Q|)$. Using Lemma 5.2, we reduce $\mathcal{A}'$ to a $(|\delta|+1)$-dimensional VASS with $|Q'|$ states that uses only increments/decrements. (Lemma 5.2 creates an m-dimensional VASS where m is the number of sets labelling transitions. Proposition 4.7 puts in $\mathcal{A}'$ only sets that already occurred in $\mathcal{A}$, with the possible exception of $\varnothing$.) Now we apply the backward coverability algorithm, as described in the proof of Proposition 5.4. By [11, Theorem 2], the algorithm will only consider counter values less than some $V = (3|Q'|)^{2^{O(|\delta|\log|\delta|)}}$. In the nondeterministic version of the algorithm, we guess the next configuration, which means we only need space $O((|\delta|+1)\log V)$ to store a couple of configurations. We have

$$\log\log V = O(|\delta|\log|\delta| + \log\log|Q'|) = O(|\delta|\log|\delta| + \log(mn + n\log n + \log|Q|))$$

Since $m, n, |\delta|, \log|Q| \in O(N)$, we conclude $\log|\delta| + \log\log V \in O(N\log N)$. This implies that a nondeterministic version of the backward coverability algorithm works in $\textsc{NSpace}\big(2^{O(N\log N)}\big)$. □

The previous proposition has a couple of obvious consequences. First, emptiness is also in $\textsc{DSpace}\big(2^{O(N\log N)}\big)$, by Savitch's theorem. Second, emptiness is also in $\textsc{DTime}\big(2^{2^{O(N\log N)}}\big)$, by a standard easy argument [19, Theorem 5.3]. In fact, one can show that the time bound applies to the backward coverability algorithm, without invoking generic constructions from complexity theory: By [11, Theorem 2], the runtime of the backward coverability algorithm — like the counter values — is also upper bounded by some $T = O\big((3|Q'|)^{2^{O(|\delta|\log|\delta|)}}\big)$. The rest of the argument is as in the proof of Proposition 6.2.

Proposition 6.3. *The emptiness problem for non-reset HRAs is* ExpSpace*-hard.*

Proof. By [25], the control-state reachability problem for VASS is ExpSpace-hard even if all the transitions are restricted to have labels of the form $\pm\vec{\delta}_i$. (More precisely, Lipton proves that certain parallel programs of size *poly*(k) can simulate any Turing machine that uses $< 2^k$ space. Then, [25, Lemma 2] asserts that reachability in these programs reduces to reachability in VAS, the full proof being: 'We omit a detailed proof of this lemma. It

should, however, be clear that parallel programs can be encoded as vector addition systems.' Similarly, we claim without proof, that it should be clear how Lipton's programs reduce to the control-state reachability problem for VASSs whose transitions only increment/decrement single counters.) We shall reduce the control-state reachability problem for such VASSs to the emptiness problem for non-reset HRAs.

Let m be the dimension of the VASS. We construct a HRA with m' histories, where m' is the smallest integer such that $m \leq 2^{m'} - 1$. As a result, there exists an injection $\phi : [m] \to \mathcal{P}_{\neq\emptyset}([m'])$; we fix arbitrarily one such injection.

- For each transition $q \xrightarrow{+\vec{\delta}_i} q'$ in the VASS, we include a transition $q \xrightarrow{\varnothing,\phi(i)} q'$ in the HRA.
- For each transition $q \xrightarrow{-\vec{\delta}_i} q'$ in the VASS, we include a transition $q \xrightarrow{\phi(i),\varnothing} q'$ in the HRA.

This construction maintains the invariant $|H@\phi(i)| = \vec{v}(i)$. To establish the invariant, we pick the initial history assignment H_0 accordingly. Finally, we set as final the state in whose reachability we are interested.

The reduction described above is clearly polynomial, from which it follows that emptiness of non-reset HRAs (even without registers) is ExpSpace-hard. □

Proposition 6.4. *The emptiness problem for non-reset HRAs is* ExpSpace*-complete.*

Proof. Immediate from Proposition 6.2 and Proposition 6.3. □

6.2. **Unary HRAs.** Our second restriction concerns allowing resets but bounding the number of histories to just one. Thus, these automata are closer to the spirit of FRAs and, in fact, extend them by rounding up their history capabilities. We show that these automata require polynomial space complexity for emptiness and retain all their closure properties apart from intersection. The latter is witnessed by failing to recognise $\mathcal{L}_2$ from the Introduction. We can see that extending this example to multiple interleavings we can show that intersection is in general incompatible with bounding the number of histories.

Definition 6.5. A $(1, n)$-HRA is called ***unary HRA*** of n registers.

In other words, unary HRAs are extensions of FRAs where names can be selectively inserted or removed from the history and, additionally, the history can be reset. These capabilities give us in fact a strict extension.

Example 6.6. The automata used in Example 2.3 for $\mathcal{L}_1$ and $\mathcal{L}_3$ were unary HRAs. Note that neither of those languages is FRA-recognisable. On the other hand, in order to recognise $\mathcal{L}_2$, an HRA would need to use at least two histories: one history for the odd positions of the input and another for the even ones. We can formalise an argument to show that $\mathcal{L}_2$ is not recognisable by unary HRAs as follows.

Proof. Suppose $\mathcal{L}_2 = \mathcal{L}(\mathcal{A})$ for some unary HRA $\mathcal{A}$ of n registers and let

$$w = a_1 b_1 \dots a_k b_k b_1 a_1 \cdots b_k a_k$$

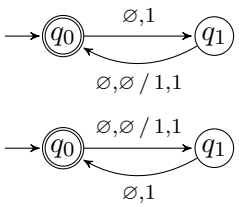
for $k = n + 1$ and some pairwise distinct names $a_1, b_1, \dots, a_k, b_k$. As $w \in \mathcal{L}_2$, there is a path, say p, in $\mathcal{A}$ which accepts w. We divide p as $p_1 p_2$ with p_2 accepting the second half of w. Let $\hat{p} = \hat{p}_1 \hat{p}_2$ be the corresponding configuration path and let (q', H') be the first configuration in $\hat{p}_2$. We set $S = \{a_1, b_1, \cdots, a_k, b_k\} \setminus \{a \mid a \in H'(i) \wedge i > 1\}$ and do a case analysis on the labels of the form (X, X') which appear in p_2 and accept names from S. Since names in S

do not appear in any $H'(i)$, for $i > 0$, it must be that each such X is either $\{1\}$ or $\varnothing$. We have the following cases.

- There are two such labels, say $(\{1\}, X_i)$ and $(\{1\}, X_j)$, accepting names a_i and b_j respectively. But this would imply that $\mathcal{A}$ also accepts w', where w' is w with these occurrences of a_i and b_j swapped, contradicting $\mathcal{L}(\mathcal{A}) = \mathcal{L}_2$ (as $w' \notin \mathcal{L}_2$).
- There are two such labels, say $(\varnothing, X_i)$ and $(\varnothing, X_j)$, accepting names a_i and b_j respectively. In order for $\mathcal{A}$ not to accept w' (w' as above), it is necessary that a reset transition with label $Y \ni 1$ occurs between the two transitions. Suppose $i < j$. Then, since $k > n$, there is a name $a_{i'}$ which does not appear in any place after clearing Y. Thus, $(\varnothing, X_j)$ can accept $a_{i'}$ and complete the path p by accepting a word $w' \notin \mathcal{L}_2$. Dually if $j \leq i$.
- Each $a_i \in S$ is accepted by a label $(\{1\}, X')$, and each $b_j \in S$ by a label $(\varnothing, X')$. Let $a_i \in S$ be the last such accepted in p_2. This means that the rest of the path has length at most $2n$. Therefore, since $k > n$, there is a $b_j \in S$ accepted in p_2 before a_i. Let (q, H) be the configuration just before accepting b_j. In order for $\mathcal{A}$ not to accept any $a_{i'}$ at that point, it must be that all $a_{i'} \in S$ appear in H. Since $|S| > n + 1$, there exists $a_{i'} \in H(1) \cap S$ such that $a_{i'} \neq a_i$. But then, the transition accepting a_i can accept $a_{i'}$ instead and lead to acceptance of a word $w' \notin \mathcal{L}_2$.

We therefore reach a contradiction in every case. □

Closure properties. The closure constructions of Section 3 readily apply to unary HRAs, with one exception: intersection. For the latter, we can observe that $\mathcal{L}_2 = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$, where $\mathcal{L}(\mathcal{A}_1) = \{a_1 a'_1 \dots a_n a'_n \in \mathcal{N}^* \mid a_1 \dots a_n \in \mathcal{L}_0\}$ and $\mathcal{L}(\mathcal{A}_2) = \{a_1 a'_1 \dots a_n a'_n \in \mathcal{N}^* \mid a'_1 \dots a'_n \in \mathcal{L}_0\}$, and $\mathcal{A}_1$ and $\mathcal{A}_2$ are the unary $(1, 0)$-HRAs on the side, with empty initial assignments. On the other hand, unary HRAs are not closed under complementation as well, as one can construct unary HRAs accepting $\overline{\mathcal{L}(\mathcal{A}_1)}$ and $\overline{\mathcal{L}(\mathcal{A}_2)}$, and then take their union to obtain a unary HRA for $\overline{\mathcal{L}_2}$.



Emptiness. In the case of just one history, the results on TR-VASS reachability [35, 16] from Section 5 provide rather rough bounds. It is therefore useful to do a direct analysis. We reduce nonemptiness for unary HRAs to control-state reachability for one dimensional R-VASSs. Our analysis below shows that the minimal path has length at most quadratic, from which it follows that nonemptiness has polynomial complexity.

The following result applies to any R-VASS representation for which $|Q| \log |Q| \in O(N)$, where $|Q|$ is the number of states of the R-VASS, and N is the number of bits used to represent the R-VASS. Note that the condition is true if all the states are listed in the R-VASS representation, something all reasonable representations would do.

Lemma 6.7. *Control-state reachability for one dimensional R-VASSs is in* NL*, provided that non-reset transitions increase and decrease the counter by at most* 1*.*

Proof. Let $\mathcal{A} = \langle Q, \delta \rangle$ be an R-VASS of dimension 1. The proof relies on two observations:

Fact 1: If $(q, i) \longrightarrow\!\!\!\!\!\rightarrow (q', i')$ is a configuration path of $\mathcal{A}$ then, for each $k > 0$, there is a path $(q, i{+}k) \longrightarrow\!\!\!\!\!\rightarrow (q', i'')$ of the same length.

Fact 2: If $(q, i) \longrightarrow\!\!\!\!\!\rightarrow (q', i')$ is a configuration path of $\mathcal{A}$ in which there are no reset transitions and the counter never becomes less than some $k > 0$, then there is a path $(q, i{-}k) \longrightarrow\!\!\!\!\!\rightarrow (q', i'')$ of the same length.

Consider an instance $(\mathcal{A}, q_0, i_0, q_F)$ of the control-state reachability problem: Is the state q_F reachable in $\mathcal{A}$ starting from configuration (q_0, i_0)? Let p be a configuration path of minimal length from (q_0, i_0) to some configuration whose state is q_F. Let us see if a state can appear repeatedly in p. By Fact 1, p is non-decreasing: any path segment $(q, i) \longrightarrow\!\!\!\!\!\rightarrow (q, i')$ can be circumvented if $i \geq i'$. Now suppose that p contains a segment $(q, i) \longrightarrow\!\!\!\!\!\rightarrow (q, i + k)$ for some $k > 0$. Consider the segment $(q, i + k) \longrightarrow\!\!\!\!\!\rightarrow (q'', i'')$ that follows, uses only non-reset transitions, and is maximal. By Fact 2, if the counter never becomes $< k$ in the latter segment, then there exists a path $(q, i) \longrightarrow\!\!\!\!\!\rightarrow (q'', i'')$ of the same length. Since p is minimal, this is a contradiction, and therefore the counter must become $< k$, somewhere after $(q, i+k)$. Let p' be the segment $(q, i + k) \longrightarrow\!\!\!\!\!\rightarrow (q', k - 1)$. Since non-reset transitions decrease the counter by ≤ 1, it must be that all the values $i + k, i + k - 1, \ldots, k - 1$ occur in p'. When one of these values is reached for the first time, it must be paired with a state that was not used for the bigger values. It follows that $(i + k) - (k - 1) + 1 \leq |Q|$, and so $i \leq |Q| - 2$. This gives us a bound on the counter value of any state that can be repeated in p. Thus, each state can appear in p at most $|Q|$ times. This implies that the length of p is at most $|Q|^2$ and that in p the counter does not exceed the value $i_0 + |Q|^2$.

We can therefore answer the instance $(\mathcal{A}, q_0, i_0, q_F)$ of the control-state reachability problem as follows. Note first that, by Facts 1 and 2 and because the length of minimal reaching path is $\leq |Q|^2$, we can replace i_0 by $\min(i_0, |Q|^2)$. Because we only consider initial counter values $\leq |Q|^2$ and because the minimal path has length $\leq |Q^2|$, we can store one configuration on the minimal path using $O(\log |Q|)$ bits. Since $|Q| \log |Q| \in O(N)$, we have $\log |Q| \in O(\log N)$, and therefore $O(\log N)$ bits suffice to represent a configuration of the minimal path. Finally, we note that a nondeterministic algorithm can guess the next configuration on the minimal path. □

We remark that an NL upper bound follows from an analysis of the backward coverability algorithm as well. However, the proof from above has the advantage that it is self-contained.

We now give an upper bound for the emptiness problem of unary HRAs. The result holds for all representations that obey several weak requirements. Let $\langle Q, q_0, \delta, H_0, F\rangle$ be a unary HRA with n registers, represented with N bits. We require that

- $n \in O(N)$, which is justified because there are $> 2^n$ possible labels on transitions;
- $|\delta| \in O(N)$, which is justified because we expect each transition to require at least a bit;
- $|Q| \log |Q| \in O(N)$, which is justified because we expect each state to be mentioned at least once in the representation. (This last point implies that $\log |Q| \in O(\log N)$.)

Proposition 6.8. *The emptiness problem for unary HRAs is in* PSPACE. *More precisely, it is in* NSPACE$(N \log N)$, *where N is the number of bits used to represent the HRA.*

Proof. Let $\mathcal{A} = \langle Q, q_0, \delta, H_0, F\rangle$ be the given unary HRA. Using Proposition 4.7, we build a $(1, 0)$-HRA $\mathcal{A}'$ that preserves emptiness, has $O(B_n 2^n n \log |\delta|)$ transitions, and has $O(B_n 2^n n \log |Q|)$ states. Using the construction from Lemma 5.2, we reduce the emptiness of $\mathcal{A}'$ to control-state reachability in an R-VASS $\mathcal{A}''$. Specialized to our case, the construction says that

- for each transition $q \xrightarrow{\varnothing,\{1\}} q'$ in $\mathcal{A}'$, we include a transition $q \xrightarrow{+1} q'$ in $\mathcal{A}''$;
- for each transition $q \xrightarrow{\{1\},\varnothing} q'$ in $\mathcal{A}'$, we include a transition $q \xrightarrow{-1} q'$ in $\mathcal{A}''$; and
- for each transition $q \xrightarrow{\{1\}} q'$ in $\mathcal{A}'$, we include a transition $q \xrightarrow{\text{reset}} q'$ in $\mathcal{A}''$.

According to Lemma 6.7, the control-state reachability problem for $\mathcal{A}''$ is in NSPACE($\log N''$), where N'' is the number of bits used to represent $\mathcal{A}''$. Thus, it remains to compute N'' as a function of N. For this, we pick one particular representation of $\mathcal{A}''$, namely a list of transitions. For such a representation we have $N'' = O\big(B_n 2^n n|\delta| \cdot \log(B_n 2^n n|Q|)\big)$. Thus,

$$\log N'' = O(n \log n + \log|\delta| + \log(n \log n + \log|Q|)) = O(N \log N)$$

The last step assumes that $n,\ |Q| \log|Q|,\ |\delta| \in O(N)$. We require the representation of $\mathcal{A}$ to satisfy these assumptions. □

Proposition 6.9. *The emptiness problem for unary HRAs is* PSPACE*-hard.*

Proof. By [15, Theorem 5.1a], the nonemptiness problem of register automata is PSPACE-hard. Register automata are a special case of unary HRAs. □

Proposition 6.10. *The emptiness problem for unary HRAs is* PSPACE*-complete.*

Proof. Immediate from Proposition 6.8 and Proposition 6.9. □

7. Summary of Main Results

The theorems in this section summarize the main results proved in the previous sections.

Theorem 7.1. Languages recognised by HRAs are closed under union, intersection, concatenation, and Kleene star, but not under complementation. Also,

- if resets are banned, then closure under Kleene star is lost;
- if the number of histories is bounded, then closure under intersection is lost.

Proof. Immediate from Proposition 3.2, Lemma 3.3, and the closure results of Section 6. □

Theorem 7.2. Deciding emptiness of an (m, n)-HRA has the following complexity:

(a) NL-complete if $m = n = 0$;
(b) NP-complete if $m = 0$ and all sets labelling transitions are singletons;
(c) PSPACE-complete if $m \leq 1$;
(d) EXPSPACE-complete if there are no reset transitions; and
(e) ACKERMANN-complete in the general case.

Proof. (a) When $m = n = 0$, nonemptiness is equivalent to reachability in a directed graph, which is a standard NL-complete problem. (b) In this case, HRAs are equivalent to RAs that disallow repetitions of values in registers, as they were originally defined [23]. For such RAs, nonemptiness is known to be NP-complete [33, Theorem 4]. (c) Proposition 6.10. (d) Proposition 6.4. (e) Proposition 5.6. □

For universality and language inclusion, see Proposition 5.9.

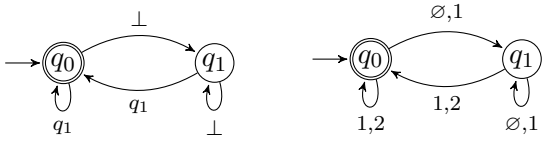
8. Connections with existing formalisms

We have already seen that HRAs strictly extend FRAs. In this section, we compare HRAs with CMAs (class memory automata). Like HRAs and FRAs, CMAs work on infinite alphabets, and have a decidable nonemptiness problem. Implicitly, we also compare with other formalisms: CMAs have been shown to express the same languages as data automata [5, Proposition 3.7]; and data automata have been shown to express the same languages as the two-variable fragment of existential monadic second order logic with data equality, position successor, and class successor [7, Proposition 14].

Definition 8.1. A ***Class Memory Automaton (CMA)*** is a tuple $\mathcal{A} = \langle Q, q_0, \phi_0, \delta, F_1, F_2 \rangle$ where Q is a finite set of states, $q_0 \in Q$ is initial, $F_1 \subseteq F_2 \subseteq Q$ are sets of final states and the transition relation is of type $\delta \subseteq Q \times (Q \cup \{\bot\}) \times Q$. Moreover, ϕ_0 is an initial *class memory function*, that is, a function $\phi : \mathcal{N} \to Q \cup \{\bot\}$ with finite domain ($\{\, a \mid \phi(a) \neq \bot \,\}$ is finite).

The semantics of a CMA $\mathcal{A}$ is given as follows. Configurations of $\mathcal{A}$ are pairs of the form (q, ϕ), where $q \in Q$ and ϕ a class memory function. The configuration graph of $\mathcal{A}$ is constructed by setting $(q, \phi) \xrightarrow{a} (q', \phi')$ just if there is $(q, \phi(a), q') \in \delta$ and $\phi' = \phi[a \mapsto q']$. The initial configuration is (q_0, ϕ_0), while a configuration (q, ϕ) is accepting just if $q \in F_1$ and, for all $a \in \mathcal{N}$, $\phi(a) \in F_2 \cup \{\bot\}$.

Thus, CMAs resemble HRAs in that they store input names in "histories", only that histories are identified with states: for each state q there is a corresponding history q (note notation overloading), and a transition which accepts a name a and leads to a state q must store a in the history q. Moreover, each name appears in at most one history (hence the type of ϕ), while the finality conditions for configurations allow us to impose that, at the end, all names must appear in specific histories, if they appear in any. For instance, the language $\overline{\mathcal{L}_4}$ of Example 3.4, which we know cannot be recognized by HRAs (Lemma 3.3), can be recognized by the following CMA on the left (with $F_1 = F_2 = \{q_0\}$).



Each name is put in history q_1 when seen for the first time, and in history q_0 when seen for the second time. The automaton accepts if all its names are in q_0. This latter condition is what makes the essential difference to HRAs, namely the capability to check where the names reside for acceptance. For example, the HRA on the right above would accept the same language it we were able to impose the condition that accepting configurations (q, H) satisfy $a \in H@\{2\}$ for all names $a \in \bigcup_i H(i)$. Note though that, extending HRAs with such finality conditions would render their nonemptiness problem reducible from reachability of R-VASS (i.e. the question whether a specific state *and* counter content can be reached), a problem known to be undecidable [2].

The above example proves that HRAs cannot express the same languages as CMAs. Conversely, as shown in [5, Proposition 7.2], the fact that CMAs lack resets does not allow them to express languages like, for example, $\mathcal{L}_1$. As a result, the languages expressed by CMAs are closed under intersection, union and concatenation, but not under Kleene star. In the latter sections of [5] several extensions of CMAs are considered, one of which does

involve resets. However, the resets considered there do not seem directly comparable to the reset capability of HRAs.

On the other hand, a direct comparison can be made with non-reset HRAs. We already saw in Proposition 4.4 that, in the latter idiom, histories can be used for simulating register behaviour. In the absence of registers, CMAs differ from non-reset HRAs solely in their constraint of relating histories to states (and their termination behaviour, which is more expressive). As the latter can be easily counterbalanced by obfuscating the set of states, we obtain the following.

Proposition 8.2. *For each non-reset HRA $\mathcal{A}$ there is a CMA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

9. Further directions

Our goal is to apply automata with histories in static and runtime verification. For static verification, the complexity results derived in this paper may seem discouraging at first. However, they are based on very specific representations of hard problems; in practice, we expect programs to yield automata of simpler complexities. Experience with tools based on coverability of TR-VASSs, like e.g. BFC [22], positively testify in that respect. Another solution, already pursued herein, is to explore constrained versions of our machines. A specific such variant we envisage to consider is one with restricted resets, in analogy to e.g. [17]. In a related direction, we aim to look at abstractions that would allow us to attack the model-checking problem for these automata, and also look at temporal logics that capture part or all of the expressivity of HRAs.

In this work we examined nondeterministic automata but did not look at alternating variants. This is justified by the undecidability of universality already at the level of register automata. However, if one is willing to restrict the number of registers and histories, there may still be room for decidability. In the case of register automata, it has been shown [15] that alternating register automata with one register are decidable for emptiness, and become undecidable at two registers. While these automata cannot capture languages that inherently require more than one register, they can use alternation to express name freshness and e.g. capture the languages $\mathcal{L}_0, \mathcal{L}_2$ of the Introduction, and also a variant of $\mathcal{L}_1$ which uses constants for tokenizing the input (instead of a_0). It would be useful to examine whether a similar restriction can yield decidable alternating HRAs, and what would their expressivity be. Finally, a problem left open here is decidability and complexity of bisimilarity. (In a private communication, Piotrek Hofman sketched a proof that bisimilarity is decidable.)

References

[1] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *Logic in Computer Science (LICS)*, 2004.

[2] T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science (TCS)*, 1976.

[3] M. Faouzi Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Logical Methods in Computer Science (LMCS)*, 2011.

[4] N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Typed Lambda Calculi and Applications (TLCA)*, 2005.

[5] H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science (TCS)*, 2010.

[6] M. Bojanczyk, L. Braud, B. Klin, and S. Lasota. Towards nominal computation. In *Principles of Programming Languages (POPL)*, 2012.
[7] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *Transactions on Computational Logic (TOCL)*, 2011.
[8] M. Bojanczyk, B. Klin, and S. Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science (LMCS)*, 2014.
[9] M. Bojanczyk, B. Klin, S. Lasota, and S. Torunczyk. Turing machines with atoms. In *Logic in Computer Science (LICS)*, 2013.
[10] A. Bouajjani, S. Fratani, and S. Qadeer. Context-bounded analysis of multithreaded programs with dynamic linked structures. In *Computer Aided Verification (CAV)*, 2007.
[11] L. Bozzelli and P. Ganty. Complexity analysis of the backward coverability algorithm for VASS. In *Reachability Problems (RP)*, 2011.
[12] Rackoff C. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science (TCS)*, 1978.
[13] C. Cotton-Barratt, A. S. Murawski, and C.-H. Luke Ong. Weak and nested class memory automata. In *Language and Automata Theory and Applications (LATA)*, 2015.
[14] N. Decker, P. Habermehl, M. Leucker, and D. Thoma. Ordered navigation on multi-attributed data words. In *Concurrency Theory (CONCUR)*, 2014.
[15] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *Transactions on Computational Logic (TOCL)*, 2009.
[16] D. Figueira, S. Figueira, S. Schmitz, and P. Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's lemma. In *Logic in Computer Science (LICS)*, 2011.
[17] A. Finkel and A. Sangnier. Mixing coverability and reachability to analyze VASS with one zero-test. In *Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 2010.
[18] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 2002.
[19] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.
[20] R. Grigore, D. Distefano, R. L. Petersen, and N. Tzevelekos. Runtime verification based on register automata. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2013.
[21] A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *Logic in Computer Science (LICS)*, 1999.
[22] A. Kaiser, D. Kroening, and T. Wahl. Efficient coverability analysis by proof minimization. In *Concurrency Theory (CONCUR)*, 2012.
[23] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science (TCS)*, 1994.
[24] J. Laird. A fully abstract trace semantics for general references. In *Automata, Languages, and Programming (ICALP)*, 2007.
[25] R. J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, 1976.
[26] A. Manuel and R. Ramanujam. Class counting automata on datawords. *Foundations of Computer Science (IJFCS)*, 2011.
[27] U. Montanari and M. Pistore. An introduction to history dependent automata. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 1997.
[28] A. S. Murawski, S. J. Ramsay, and N. Tzevelekos. Game semantic analysis of equivalence in IMJ. In *Automated Technology for Verification and Analysis (ATVA)*, 2015.
[29] A. S. Murawski and N. Tzevelekos. Algorithmic nominal game semantics. In *European Symposium on Programming (ESOP)*, 2011.
[30] A. S. Murawski and N. Tzevelekos. Algorithmic games for full ground references. In *Automata, Languages, and Programming (ICALP)*, 2012.
[31] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *Transactions on Computational Logic (TOCL)*, 2004.
[32] A. M. Pitts and I. Stark. On the observable properties of higher order functions that dynamically create local names, or: What's new? In *Mathematical Foundations of Computer Science (MFCS)*, 1993.
[33] H. Sakamoto and D. Ikeda. Intractability of decision problems for finite-memory automata. *Theoretical Computer Science (TCS)*, 2000.

[34] S. Schmitz and P. Schnoebelen. Algorithmic aspects of WQO theory. Lecture Notes ⟨cel-00727025v2⟩, 2012.
[35] P. Schnoebelen. Revisiting Ackermann-hardness for Lossy Counter Machines and Reset Petri Nets. In *Mathematical Foundations of Computer Science (MFCS)*, 2010.
[36] L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Computer Science Logic (CSL)*, 2006.
[37] I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge Computing Laboratory, 1995.
[38] N. Tzevelekos. Fresh-register automata. In *Principles of Programming Languages (POPL)*, 2011.
[39] N. Tzevelekos and R. Grigore. History-register automata. In *Foundations of Software Science and Computation Structures (FoSSaCS)*, 2013.