History-Register Automata

Nikos Tzevelekos and Radu Grigore

Queen Mary, University of London

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 and analysing 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 (Petri nets with reset arcs), 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 [23] and variable programs [18], and recently developing towards automata for verification tasks such as equivalence checks of ML programs [24,25], context-bounded analysis of concurrent programs [7,3] and runtime program monitoring [14].

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. [31] for an overview). A particularly intuitive such model is that of *Register Automata (RA)* [18,26], 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 [33], 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,

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

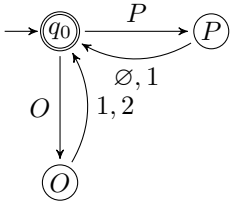
which captures the output of a fresh-name generator ($\mathcal{N}$ is an infinite set of names). FRAs are expressive enough to model, for example, finitary fragments of languages like the π-calculus [33] or ML [24].

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,

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

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 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*



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.

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

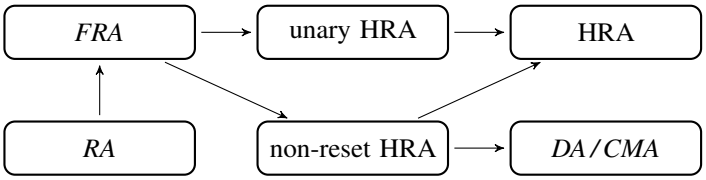


Fig. 2. Expressiveness of history-register automata compared to previous models (in italics). The inclusion $\mathcal{M} \longrightarrow \mathcal{M}'$ means that for each $\mathcal{A} \in \mathcal{M}$ we can effectively construct an $\mathcal{A}' \in \mathcal{M}'$ accepting the same language as $\mathcal{A}$.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

(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.

$$\mathcal{L}_1 = \{a_0 w_1 \cdots a_0 w_n \in \mathcal{N}^* |\ \forall i.\ w_i \in \mathcal{N}^* \wedge a_0 w_i \in \mathcal{L}_0\} \text{ for given } a_0$$
$$\mathcal{L}_2 = \{a_1 a'_1 \cdots a_n a'_n \in \mathcal{N}^* |\ a_1 \cdots a_n, a'_1 \cdots a'_n \in \mathcal{L}_0\}$$
$$\mathcal{L}_3 = \{a_1 \cdots a_n a'_1 \cdots a'_{n'} \in \mathcal{N}^* |\ a_1 \cdots a_n, a'_1 \cdots a'_{n'} \in \mathcal{L}_0 \wedge \forall i. \exists j.\ a'_i = a_j\}$$

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 enable us to use histories as counters and simulate counter machines, and in particular Petri nets with reset arcs [2]. We therefore obtain non-primitive recursive bounds for checking language emptiness. Given that language containment and universality are undecidable already for register automata [26], 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 1 (weakening in (2) and (1) 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 [27] and Stark's PhD [32]. Names have subsequently been incorporated in the semantics literature (see e.g. [16,4,1,19]), especially after the advent of *Nominal Sets* [13], 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

[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.

automata [24,25], thus achieving automated equivalence checks for fragments of ML. In a parallel development, a research stream on automated analysis of dynamic concurrent programs has developed essentially the same formalisms, this time stemming from basic operational semantics [7,3]. 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 propose 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 [31] 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 1) and in fact can be seen as a variant of *Data Automata (DA)* [6] or, equivalently, *Class Memory Automata (CMA)* [5]. This fit leaves space for transfer of technologies and, more specifically, of the associated logics of data automata.

2 Definitions and first properties

We start by fixing some notation. Let $\mathcal{N}$ be an infinite alphabet of ***names*** (or *data values*, in terminology of [31]), which we range over by a, b, c, etc. For any pair of natural numbers $i \leq j$, we write $[i, j]$ for the set $\{i, i{+}1, \cdots, j\}$, and for each i we let $[i]$ be the set $\{1, \cdots, i\}$. For any set S, we write $|S|$ for the cardinality of S, $\mathcal{P}(S)$ for the powerset of S, $\mathcal{P}_{\mathsf{fin}}(S)$ for the set of finite subsets of S, and $\mathcal{P}_{\neq\emptyset}(S)$ for the set of non-empty 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 $(\emptyset, X')$ 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. Reset transitions do not accept names; they are ϵ-transitions from the outside. Note then that the label $(X, \emptyset)$ 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.

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, $\{(i, \emptyset) \mid i \in [m{+}n]\}$ is the empty assignment. We range over elements of Asn by H and variants, and over elements of Lab by ℓ and variants. Moreover, it will be handy to introduce the following notation for assignments. For any assignment H and any $a \in \mathcal{N}$, $S \subseteq \mathcal{N}$ and $X \subseteq [m{+}n]$:

- 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, $H@\emptyset = \mathcal{N} \setminus \bigcup_i H(i)$ 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. $H[X \mapsto \emptyset]$ 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.

We can now define our automata.

Definition 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. *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 $(q, H) \xrightarrow{x} (q', H')$ just if one of the following conditions is satisfied.*

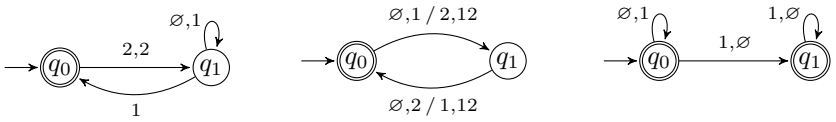
- *$x = a \in \mathcal{N}$ and there is $q \xrightarrow{X,X'} q' \in \delta$ such that $a \in H@X$ and $H' = H[a \,\mathsf{in}\, X']$.*

– $x = \epsilon$ *and there is* $q \xrightarrow{X} q' \in \delta$ *such that* $H' = H[X \mapsto \emptyset]$.

The language accepted by $\mathcal{A}$ *is* $\mathcal{L}(\mathcal{A}) = \{ w \in \mathcal{N}^* \mid (q_0, H_0) \xrightarrow{w}\!\!\!\!\rightarrow (q, H)$ *and* $q \in F \}$ *where* $\longrightarrow\!\!\!\!\rightarrow$ *is the reflexive transitive closure of* $\longrightarrow$ *(i.e.* $\hat{q} \xrightarrow{x_1 \cdots x_n}\!\!\!\!\rightarrow \hat{q}'$ *if* $\hat{q} \xrightarrow{x_1} \cdots \xrightarrow{x_n} \hat{q}'$*).*

Note that we use ϵ both for the empty sequence and the empty transition so, in particular, when writing sequences of the form $x_1 \cdots x_n$ we may implicitly consume ϵ's.

Example 3. The language $\mathcal{L}_1$ of the Introduction is recognised by the following $(1,1)$-HRA (leftmost below), with initial assignment $\{(1, \emptyset), (2, a_0)\}$. 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_1 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* [18,26,33]. The latter can be defined within the HRA framework as follows.[2]

Definition 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* $(q, \emptyset, X', q') \in \delta$*.*

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 [33] we have the following.[3]

Lemma 5. *The languages* $\mathcal{L}_1, \mathcal{L}_2$ *and* $\mathcal{L}_3$ *are not FRA-recognisable.*

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.

[2] The definitions given in [18,26,33] are slightly different but can routinely be shown equivalent.
[3] $\mathcal{L}_1$ was explicitly examined in [33]. 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 [33].

Definition 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.*

The following is a standard result.

Lemma 7. *If $\mathcal{A}_1 \sim \mathcal{A}_2$ then $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$.*

As a first taste of HRA reasoning, we sketch a technique for simulating registers by histories in HRAs. The idea is to represent a register by a history whose size is always kept at most 1. To ensure that histories are effectively kept in size ≤ 1 they must be cleared before inserting names, which in turn complicates deciding when a transition can be taken as it may depend on the deleted names. To resolve this, we keep two copies of each register so that, for each transition with label (X, X'), we use one set of copies for the name comparisons needed for the X part of the label, and the other set for the assignments dictated by X'. Resets are used so that one set of copies is always empty.

Proposition 8. *Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$ be an (m,n)-HRA. There is an $(m{+}2n, 0)$-HRA $\mathcal{A}'$ such that $\mathcal{A} \sim \mathcal{A}'$.*

In Proposition 20 we show that registers can be simulated also without using resets. Both that and the above reductions, though, come at the cost of an increased number of histories and, more importantly, the simulation technique obscures the intuition of registers and produces automata which need close examination even for simple languages like the one which contains all words $a_1 \cdots a_n$ such that $a_i \neq a_{i+1}$ for all i (see Example 21). As, in addition, it is not applicable to the weaker unary HRAs we examine in Section 4, we preferred to explicitly include registers in HRAs. Another design choice regards the use of sets of places in transitions instead e.g. of single places. Although the latter description would lead to an equivalent and probably conciser formalism, it would be inconvenient for combining HRAs e.g. in order to produce the intersection of their accepted languages. In fact, our formulation follows *M-automata* [18], an equivalent presentation of RAs susceptible to closure constructions.

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, $q \xrightarrow{\emptyset}\!\!\!\!\rightarrow q \in \delta$.

Definition 9. *We say that an HRA $\mathcal{A}$ is* **deterministic** *if, 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 $\hat{q}_1 = \hat{q}_2$.*
$\mathcal{A}$ is **strongly deterministic** *if $q \xrightarrow{Y_1}\!\!\!\!\rightarrow \cdot \xrightarrow{X\setminus Y_1, X_1} q_1 \in \delta$ and $q \xrightarrow{Y_2}\!\!\!\!\rightarrow \cdot \xrightarrow{X\setminus Y_2, X_2} q_2 \in \delta$ imply $q_1 = q_2$, $Y_1 = Y_2$ and $X_1 = X_2$.*

Lemma 10. *If $\mathcal{A}$ is strongly deterministic then it is deterministic.*

3 Closure properties, emptiness and universality

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.

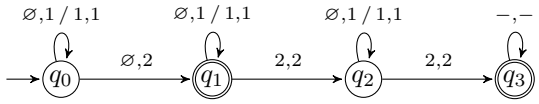
In fact, the design of HRAs is such that the automata for union and intersection come almost for free through a straightforward product construction which is essentially an ordinary product for finite-state automata, modulo reindexing of places to account for duplicate labels (cf. [18]). The constructions for Kleene star and concatenation are slightly more involved as we make use of 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 allows us, for instance, to create loops such that after each loop transition the same initial configuration occurs (in this case, w would enlist all initial names).

Proposition 11. *Languages recognised by HRAs are closed under union, intersection, concatenation and Kleene star.*

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 with states. 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 [22].

Lemma 12. *HRAs are not closed under complementation.*

Example 13. Consider $\mathcal{L}_4 = \{w \in \mathcal{N}^* \mid \text{ not all names of } w \text{ occur exactly twice in it }\}$, which is accepted by the $(2, 0)$-HRA below, where "$-$" can be any of $\varnothing, 1, 2$.



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. For suppose it were recognisable (wlog) by an $(m, 0)$-HRA $\mathcal{A}$ with k states. Then, $\mathcal{A}$ would accept the word $w = a_1 \cdots a_k\, a_1 \cdots a_k$ 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 $(\emptyset, X)$, 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 $X \neq \emptyset$, in some configuration (q_1, H_1) such that $H_1@X = \emptyset$. 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 non-empty $X \subseteq [m]$, 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.

We now turn to the question of checking emptiness. The use of unbounded histories effectively renders our machines into counter automata: where a counter automaton would increase (or decrease) a counter, an HRA would add (remove) a name from one of its histories, or set of histories. Nonetheless, HRAs cannot decide their histories for emptiness, which leaves space for decidability.[4] The capability for resetting histories, on the other hand, leads us to consider Transfer-Reset Vector Addition Systems with States [8,2] (i.e. Petri nets with reset and transfer arcs) as appropriate formalisms for this question.

A ***Transfer-Reset Vector Addition System with States (TR-VASS)*** of m dimensions is a tuple $\mathcal{A} = \langle Q, \delta \rangle$, with Q a set of states and $\delta \subseteq Q \times (\{-1, 0, 1\}^m \cup [m]^2 \cup [m]) \times Q$ a transition relation. Each dimension of $\mathcal{A}$ corresponds to an unbounded counter. Thus, a transition of $\mathcal{A}$ can either update its counters by addition of a vector $\vec{v} \in \{-1, 0, 1\}^m$, or transfer the value of one counter to another, or reset some counter.

Formally, a configuration of $\mathcal{A}$ is a pair $(q, \vec{v}) \in Q \times \mathbb{N}^m$ consisting of a state and a vector of values stored in the counters. The configuration graph of $\mathcal{A}$ is constructed by including an edge $(q, \vec{v}) \to (q', \vec{v}')$ if:

- there is some $(q, \vec{v}'', q') \in \delta$ such that $\vec{v}' = \vec{v} + \vec{v}''$, or
- there is $(q, i, j, q') \in \delta$ such that $\vec{v}' = (\vec{v}[j \mapsto v_i + v_j])[i \mapsto 0]$,
- or there is some $(q, i, q') \in \delta$ such that $\vec{v}' = \vec{v}[i \mapsto 0]$;

where we write v_i for the ith dimension of $\vec{v}$, and $\vec{v}[i \mapsto v']$ for the update of $\vec{v}$ where the i-th counter is set to v'. An ***R-VASS*** is a TR-VASS without transfer transitions.

The ***control-state reachability*** problem for TR-VASSs is defined as follows. Given a TR-VASS $\mathcal{A}$ of m dimensions, a configuration $(q_0, \vec{v}_0)$ and a state q, is there some $\vec{v} \in \mathbb{N}^m$ such that $(q_0, \vec{v}_0) \twoheadrightarrow (q, \vec{v})$? In such a case, we write $(\mathcal{A}, q_0, \vec{v}_0, q) \in \mathsf{Reach}$.

Fact 14 ([10,30,11]) *Control-state reachability for TR-VASSs and R-VASSs is decidable and has non-primitive recursive complexity.*

We next reduce HRA nonemptiness to TR-VASS control-state reachability. Starting w.l.o.g. from an $(m, 0)$-HRA $\mathcal{A}$, we construct a TR-VASS $\mathcal{A}'$ with 2^m dimensions: one dimension $\widetilde{X}$ for each $X \subseteq [m]$. The dimension $\widetilde{\emptyset}$ is used for garbage collecting. We assign to each state of $\mathcal{A}$ a corresponding state in $\mathcal{A}'$ (and also include a stock of dummy states for intermediate transitions) and translate the transitions of $\mathcal{A}$ into transitions of $\mathcal{A}'$ as follows.

[4] Recall that 2-counter machines with increase, decrease and check for zero are Turing complete.

– Each transition with label (X, X') is mapped into a pair of transitions which first decrease counter $\widetilde{X}$ and then increase $\widetilde{X}'$.
– Each reset transition with label X causes a series of transfers: for each counter $\widetilde{Y}$, we do a transfer from $\widetilde{Y}$ to $\widetilde{Y \setminus X}$.

Thus, during computation in each of $\mathcal{A}'$ and $\mathcal{A}$, the value of counter $\widetilde{X}$ matches the number of names which precisely appear in histories X. Since, for checking emptiness, the specific names inside the histories of $\mathcal{A}$ are of no relevance, the above correspondence extends to matching nonemptiness for $\mathcal{A}$ to (final) control-state reachability for $\mathcal{A}'$.

Proposition 15. *Emptiness is decidable for HRAs.*

Doing the opposite reduction we can show that emptiness of even strongly deterministic HRAs is non-primitive recursive. In this direction, each R-VASS $\mathcal{A}$ of m dimensions is simulated by an $(m, 0)$-HRA $\mathcal{A}'$ 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. Using a non-trivial encoding of resets we can ensure that if $\mathcal{A}$ adheres to a particular kind of determinacy conditions (which the machines used in [30] for proving non-primitive recursive complexity do adhere to) then $\mathcal{A}'$ is strongly deterministic.

Proposition 16. *Emptiness for strongly deterministic HRAs is non-primitive recursive.*

We finally consider universality and language containment. Note first that our machines inherit undecidability of these properties from register automata [26]. However, these properties are decidable in the deterministic case, as deterministic HRAs are closed under complementation. In particular, given a deterministic HRA $\mathcal{A}$, the automaton $\mathcal{A}'$ accepting the language $\mathcal{N} \setminus \mathcal{L}(\mathcal{A})$ can be constructed in an analogous way as for deterministic finite-state automata, namely by obfuscating the automaton with all "missing" transitions and swapping final with non-final states (modulo ϵ-transitions). We add the missing transitions as follows. For each state q and each set X such that there are no transitions of the form $q \xrightarrow{Y\!\!\!\!\!\!\rightarrow} \cdot \xrightarrow{X \setminus Y, X'} q'$ in $\mathcal{A}$, we add a transition $q \xrightarrow{X,\emptyset} q_S$ to some sink non-final state q_S.

Proposition 17. *Language containment and universality are undecidable for HRAs. They are decidable for deterministic HRAs, with non-primitive recursive complexity.*

4 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 express meaningful properties and, at the same time, remain at feasible complexity. 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.

Our analysis on emptiness for general HRAs from Section 3 is not applicable to these weaker machines as we now need to take registers into account: the simulation of registers by histories is either not possible or not practical for deriving satisfactory complexity bounds. Additionally, a direct analysis will allow us to reduce instances of counter machine problems to our setting decreasing the complexity size by an exponential, compared to our previous reduction. Solving emptiness for each of the weaker versions of HRAs will involve reduction to a name-free counter machine. In both cases, the reduction shall follow the same concept as in Section 3, namely of simulating computations with names *symbolically*.

4.1 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 18. *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 $q \xrightarrow{X} q' \in \delta$.*

We call such a machine a non-reset (m, n)-HRA. In an analogous fashion, a ***VASS*** of m dimensions (an *m-VASS*) is an R-VASS with no reset transitions. For these machines, control-state reachability is significantly less complex.

Fact 19 *Control-state reachability for VASSs is* EXPSPACE*-complete [21,28], and can be decided in space $O((M+\log|Q|)\cdot 2^{\kappa\, m\log m})$, where Q the set of states of the examined instance, m the vector size, M the maximum initial value and κ a constant [29].*

Closure properties Of the closure constructions of Section 3, those for union and intersection readily apply to non-reset HRAs, while the construction for concatenation needs some minor amendments. 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 13 is a non-reset HRA, which implies that non-reset HRAs are not closed under complementation.

Emptiness We next reduce nonemptiness for non-reset HRAs to control-state reachability for VASSs. Starting from a non-reset (m, n)-HRA $\mathcal{A} = \langle Q, q_0, H_0, \delta, F\rangle$, the reduction maps each non-empty subset of $[m]$ *which appears in* δ to a VASS counter ($Y \subseteq [m]$ appears in δ if there is $(q, X, X', q') \in \delta$ such that $Y \in \{X\cap[m], X'\cap[m]\}$). Thus, the resulting VASS $\mathcal{A}'$ has m' counters, where $m' \leq 2|\delta|$. Although the number of states

of $\mathcal{A}'$ is exponential, as the status of the registers needs to be embedded in states, the dominating factor for state-reachability is m', which is linear in the size of $\mathcal{A}$.

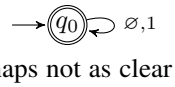
For the converse direction, we reduce reachability for a VASS of 2^m-1 counters to nonemptiness for an $(m,0)$-automaton: we map each counter to a non-empty subset of $[m]$. Note that such a $(2^m$–$1)$-to-m reduction would not work for R-VASSs, hence the different reduction in Proposition 16. This is because resets in HRAs cannot fully capture the behaviour of resets in R-VASSs. In HRAs a reset of a set of histories $\{1,2\}$, say, cannot occur without also resetting histories 1 and 2. In addition, resets necessarily cause virtual transfers of names (e.g. resetting history 1 makes all names appearing precisely in $\{1,2\}$ to appear precisely in history 2).

Proposition 20. *Emptiness checking for non-reset HRAs is* EXPSPACE*-complete.*

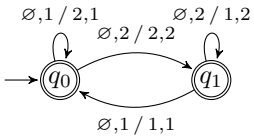
Non-reset HRAs without registers We now show that non-reset HRAs with only histories are as expressive as general non-reset HRAs. The equivalence we prove is weaker than the one we proved for general HRAs: we obtain language equivalence rather than bisimilarity. Our proof below is based on the *colouring technique* of [5]. Before we proceed with the actual result, let us first demonstrate the technique through an example.

Example 21. It is easy to see that the following language

$$\mathcal{L}_5 = \{a_1 \cdots a_n \in \mathcal{N}^* \mid \forall i.\, a_i \neq a_{i+1}\}$$



is recognised by the $(0,1)$-HRA on its right. What is perhaps not as clear is that the $(2,0)$-HRA on the right below, call it $\mathcal{A}$, accepts the same language.



Note first that, by construction, it is not possible for $\mathcal{A}$ to accept the same name in two successive transitions: if we write (X, X') for the labels of incoming transitions to q_0 and (Y, Y') for the outgoing, we cannot match any X' with some Y, and similarly for q_1. This shows $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}_5$. To prove the other inclusion, we need to show that for every word $w = a_1 \cdots a_n \in \mathcal{L}_5$ there is an accepting run in $\mathcal{A}$. For this, it suffices to find a sequence $\ell_1, \cdots, \ell_n$ of labels from the set $\{(\emptyset,1),(\emptyset,2),(1,1),(1,2),(2,1),(2,2)\}$, say $(\ell_i = (X_i, X_i'))$, satisfying:

1. For any i, $X_i' \neq X_{i+1}$.
2. If $a_i = a_j$, $i < j$, and for no $i < k < j$ do we have $a_i = a_k$ then $X_i' = X_j$.
3. For any i, if $a_i \neq a_j$ for all $j < i$ then $X_i = \emptyset$.

The first condition ensures that the sequence corresponds to a valid transition sequence in $\mathcal{A}$, and the other two that the sequence accepts the word $w = a_1 \cdots a_n$. Conditions 1 and 2 determine dependencies between the choices of left and right components in ℓ_is. Let us attach to w dependency pointers as follows: attach a pointer of type 1 (dependency right-to-left) from each a_i to its next occurrence in w, say a_j; from each a_{i+1} attach a type 2 pointer (dependency left-to-right) to a_i. Now note that, as there is no cycle in w which alternates between type 1 and type 2 pointers, it is always possible to produce a valid sequence $\ell_1, \cdots, \ell_n$.

We now state the general result. The proof follows the rationale described above, and is omitted for space limitations. We assume automata with their registers initially empty – the general case can be captured by first applying a construction like $\mathcal{A}$ fix w

of Section 3 (the construction introduces new registers where we would store the initial names, but we can as well use new histories for the same purpose).

Proposition 22. *For each (m, n)-non-reset HRA $\mathcal{A}$ with initially empty registers there is an $(m+3n, 0)$-non-reset HRA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

4.2 Unary HRAs

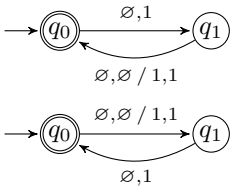
Our second restriction concerns allowing resets but bounding the number of histories to just one. Thus, these automata are closer to the spirit of FRAs and, in fact, extend them by rounding up their history capabilities. We show that these automata require polynomial space complexity for emptiness and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

Definition 23. *A $(1, n)$-HRA is called a* **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 a strict extension.

Example 24. The automata used in Example 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. Following this intuition we can show that $\mathcal{L}_2$ is not recognisable by unary HRAs.

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 $\mathcal{L}(\mathcal{A}_1) = \{a_1 a_1' \cdots a_n a_n' \in \mathcal{N}^* \mid a_1 \cdots a_n \in \mathcal{L}_0\}$ and $\mathcal{L}(\mathcal{A}_2) = \{a_1 a_1' \cdots a_n a_n' \in \mathcal{N}^* \mid a_1' \cdots a_n' \in \mathcal{L}_0\}$, 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 [30,11] from Section 3 provide rather rough bounds. It is therefore useful to do a direct analysis. We reduce nonemptiness for unary HRAs to control-state reachability for R-VASSs of 1 dimension. Although these machines can be seen as close relatives to several other formalisms, like one-counter automata or pushdown automata on a one-letter alphabet, to the best of our knowledge there has been no direct attack of state reachability for them. Our analysis below, which follows standard techniques, yields square minimal-path length, and hence a polynomial complexity for emptiness (N is the size of the input).

Lemma 25. *Control-state reachability for R-VASSs of dimension 1 can be decided in* $\textsc{Space}(\log^2 N)$.

Proposition 26. *Emptiness for unary HRAs can be decided in* $\textsc{Space}((N \log N)^2)$.

5 Connections with existing formalisms

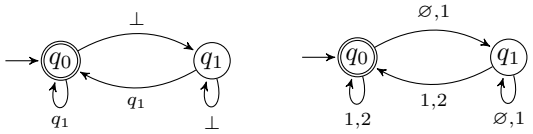
We have already seen that HRAs strictly extend FRAs. In this section we shall draw connections between HRAs and an automata model over infinite alphabets at the limits of decidability, called *Data Automata (DA)*, introduced in [6] in the context of XML theory. DAs operate on *data words*, i.e. over finite sequences of elements from $\mathcal{S} \times \mathcal{N}$, where $\mathcal{S}$ is a finite set of *data tags* and $\mathcal{N}$ is an infinite set of *data values* (but we shall call them *names*). A DA operates in two stages which involve a transducer automaton and a finite-state automaton respectively. Both automata operate on the tag projection of the input, with the second automaton focussing on tags paired with the same name.

For the rest of our discussion we shall abuse data words and treat them simply as strings of names, neglecting data tags. This is innocuous since there are straightforward translations between the two settings.[5] An equivalent formulation of DAs which is closer to our framework is the following [5].

Definition 27. *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 $\mathcal{A}$ like the above 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) and, moreover, the finality conditions for configurations allow us to impose that all names appear in specific histories, if they appear in any. For example, here is a CMA (left below, with $F_1 = F_2 = \{q_0\}$) which recognises $\overline{\mathcal{L}_4}$ of Example 13.



Each name is put in history q_1 when seen for the first time, and to 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 were we able to impose the condition that accepting configurations (q, H) satisfy $a \in H@\{2\}$ for all names $a \in \bigcup_i H(i)$.

[5] A string of names is the same as a data word over a singleton set of data tags; while data tags can be simulated by names in registers of the initial configuration which do not get moved nor copied during the computation.

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, $\mathcal{L}_1$. 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 22 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 28. *For each non-reset HRA $\mathcal{A}$ there is a CMA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

6 Further directions and acknowledgements

Our goal is to apply automata with histories in static and runtime verification. While FRAs have been successful in modelling programs which, at each point during computation, can have access to a bounded memory fragment [33,24], HRAs allow us to express access to unbounded memory, provided that memory locations can be grouped in a bounded number of equivalence classes. Moreover, with HRAs we can express a significantly wider range of properties, closed under complementation-free regular operations, and in particular we can write properties where the history is used in meaningful ways (cf. the scenario of Figure 1). Although the complexity results derived in this paper may seem discouraging at first, they are based on quite specific representations of hard problems; in practice, we expect programs to yield automata of low complexities. Experience with tools based on TR-VASS coverability, like e.g. BFC [17], positively testify in that respect. On the other hand, an extension we envisage to consider is one with restricted emptiness tests, in analogy to e.g. [12].

A connection we would like to investigate is that between our automata and register automata which use alternation. Such machines with one register express behaviours related to HRAs [9] and enjoy some common properties, such as non-primitive recursive complexity for emptiness. Another interesting connection is with *Data Nets* [20], a class of machines which combine Petri nets with infinite alphabets but are not formalised as language acceptors over them. In terms of complexity, data nets seem substantially more involved than reset Petri nets and our machines. Finally, a problem left open in this work is decidability and complexity of bisimilarity. Although it is known that bisimilarity is undecidable for Petri nets [15], the version which seems of relevance towards an undecidability argument for HRAs is that of *visibly* counter automata with labels, i.e. automata which accept labels at each transition, and the action of each transition is determined by its label. The latter problem is not known to be decidable.

We would like to thank Dino Distefano, Petr Jancar, Ranko Lazic, Philippe Schnoebelen, Sylvain Schmitz and anonymous reviewers for fruitful discussions, suggestions and explanations. This work was supported by EPSRC grant H011749 (Grigore) and a Royal Academy of Engineering research fellowship (Tzevelekos).

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 *LICS*, pp. 150–159, 2004.
2. T. Araki and T.Kasami. Some decision problems related to the reachability problem for Petri nets. *Theor. Comput. Sci.*, 3(1):85–104, 1977.
3. M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Log. Meth. Comput. Sci.*, 7(4), 2011.
4. N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *TLCA*, pp. 86–101, 2005.
5. H. Björklund and T. Schwentick. On notions of regularity for data languages *TCS*, 411, 2010.
6. M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS*, pp. 7–16, 2006.
7. A. Bouajjani, S. Fratani, and S. Qadeer. Context-bounded analysis of multithreaded programs with dynamic linked structures. In *CAV*, pp. 207–220, 2007.
8. G. Ciardo. Petri nets with marking-dependent arc cardinality. In *ICATPN*, 1994.
9. S. Demri, R. Lazic. LTL with the freeze quantifier and register automata. *TOCL* 10(3), 2009.
10. C. Dufourd, A. Finkel and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *ICALP*, pp. 103-115, 1998.
11. D. Figueira, S. Figueira, S. Schmitz, and Ph. Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's lemma. In *LICS*, pp. 269–278, 2011.
12. A. Finkel and A. Sangnier. Mixing coverability and reachability to analyze VASS with one zero-test. In *SOFSEM*, pp. 394–406, 2010.
13. M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.
14. R. Grigore, D. Distefano, R. L. Petersen and N. Tzevelekos. Runtime verification based on register automata. To appear in *TACAS*, 2013.
15. P. Jancar. Undecidability of bisimilarity for Petri nets and some related problems. *Theor. Comput. Sci.*, 148(2):281–301, 1995.
16. A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *LICS*, 1999.
17. A. Kaiser, D. Kroening, and T. Wahl. Efficient coverability analysis by proof minimization. In *CONCUR*, 2012. `http://www.cprover.org/bfc/`
18. M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2), 1994.
19. J. Laird. A fully abstract trace semantics for general references. In *ICALP*, 2007.
20. R. Lazic, T. Newcomb, J. Ouaknine, A. W. Roscoe, and J. Worrell. Nets with tokens which carry data. *Fundam. Inform.*, 88(3), 251–274, 2008.
21. R. Lipton. The reachability problem requires exponential space. Tech. Rep. 62, Yale, 1976.
22. A. Manuel and R. Ramanujam. Class counting automata on datawords. *Int. J. Found. Comput. Sci.*, 22(4):863–882, 2011.
23. U. Montanari and M. Pistore. An introduction to History Dependent Automata. *Electr. Notes Theor. Comput. Sci.*, 10, 1997.
24. A. S. Murawski and N. Tzevelekos. Algorithmic nominal game semantics. In *ESOP*, 2011.
25. A. S. Murawski and N. Tzevelekos. Algorithmic games for full ground references. In *ICALP*, pp. 312–324, 2012.
26. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.
27. A. M. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What's *new*? In *MFCS*, pp. 122–141, 1993.
28. C. Rackoff. The covering and boundedness problems for vector addition systems. *TCS*, 1978.
29. L. E. Rosier, H.-C. Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *J. Comput. Syst. Sci.*, 32(1):105–135, 1986.
30. Ph. Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *MFCS*, pp. 616–628, 2010.
31. L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, 2006.
32. I. Stark. *Names and higher-order functions*. PhD thesis, University of Cambridge, 1994.
33. N. Tzevelekos. Fresh-register automata. In *POPL*, pp. 295–306, 2011.