Register automata with linear arithmetic

Yu-Fang Chen*, Ondřej Lengál†, Tony Tan‡ and Zhilin Wu§

* Institute of Information Science, Academia Sinica, Taiwan
† FIT, Brno University of Technology, IT4Innovations Centre of Excellence, Czech Republic
‡ Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
§ State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China

***Abstract*—We propose a novel automata model over the alphabet of rational numbers, which we call *register automata over the rationals* ($\mathbf{RA}_{\mathbb{Q}}$). It reads a sequence of rational numbers and outputs another rational number. $\mathbf{RA}_{\mathbb{Q}}$ is an extension of the well-known *register automata* (RA) over infinite alphabets, which are finite automata equipped with a finite number of registers/variables for storing values. Like in the standard RA, the $\mathbf{RA}_{\mathbb{Q}}$ model allows both equality and ordering tests between values. It, moreover, allows to perform linear arithmetic between certain variables. The model is quite expressive: in addition to the standard RA, it also generalizes other well-known models such as affine programs and arithmetic circuits.**

The main feature of $\mathbf{RA}_{\mathbb{Q}}$ is that despite the use of linear arithmetic, the so-called *invariant problem*—a generalization of the standard non-emptiness problem—is decidable. We also investigate other natural decision problems, namely, *commutativity*, *equivalence*, and *reachability*. For deterministic $\mathbf{RA}_{\mathbb{Q}}$, commutativity and equivalence are polynomial-time inter-reducible with the invariant problem.

I. Introduction

Motivated by various needs and applications, there have occurred many studies on languages over infinite alphabets. To name a few, typical applications include database systems, program analysis and verification, programming languages and theory by itself. See, e.g., [1]–[9] and the references therein. One of the most popular models are arguably register automata (RA) [10], [11]. Briefly, an RA is a finite automaton equipped with a finite number of registers, where each register can store one symbol at a time. The automaton then moves from state to state by comparing the input symbol with those in its registers, and at the same time may decide to update the content of its registers by storing a new symbol into one of its registers, and thus, "forgetting" the previously stored symbol. The simplicity and naturality of RA obviously contribute to their appeal.

So far, the majority of research in this direction has focused on models where the only operations allowed on the input symbols are equality and order relation. For many purposes, this abstraction is good enough. For example, relational algebra-based queries, often used in database systems, involve only equality tests [12]. For many simple but common queries, such as counting the number of elements or summing up the values in a list, at least some arithmetic is, however, required.

It is a folklore belief that allowing RA to perform even the simplest form of arithmetic on their registers will immediately yield undecidability for the majority of interesting decision problems. The evidence is that such RA subsume simple two-counter machines, which are already Turing-complete [13]. Indeed, the belief holds not only for RA, but for the majority (if not all) of the models of languages over infinite alphabets.

In this paper, we propose a novel automaton model over the rational numbers $\mathbb{Q}$, named *register automata over the rationals* ($\mathrm{RA}_{\mathbb{Q}}$). Like in standard RA, an $\mathrm{RA}_{\mathbb{Q}}$ is equipped with a finite number of variables (registers), each of them is able to store a value.* The $\mathrm{RA}_{\mathbb{Q}}$ model allows to test order and perform linear arithmetic between some variables, yet keeps several interesting decision problems decidable. The key idea is the partitioning of variables into two sets, *control variables* and *data variables*, which is inspired by the work of Alur and Černý [14]. *Control variables* can be used in transition guards for order ($\leqslant$) comparison and can be assigned a value either from the input or from another control variable. In contrast, *data variables* can store a value obtained from a linear combination of the values of all variables and the value from the input, but cannot be used in transition guards. In a final state, an $\mathrm{RA}_{\mathbb{Q}}$ outputs a rational number obtained by a linear combination of the values of all variables (non-final states have no output). Due to nondeterminism, it is possible that different computation paths for the same input word produce different output values. $\mathrm{RA}_{\mathbb{Q}}$ can be used to model, e.g., the following aggregate functions: finding the smallest and the largest elements, finding the k-th largest element, counting the number of elements above a certain threshold, summing all elements, or counting the number of occurrences of the largest element in a list.

The $\mathrm{RA}_{\mathbb{Q}}$ model is a very general model that captures and simulates at least three other well-known models. The first and obvious one is the standard RA studied in [9]–[11], [15]. An RA is simply an $\mathrm{RA}_{\mathbb{Q}}$ without data variables that only allows equality test of control variables and in a final state outputs the constant 1. The second one is the affine program (AP) model defined by Karr [16], which is commonly used as a standard abstract domain in static program analysis [17], [18]. An AP is a special case of an $\mathrm{RA}_{\mathbb{Q}}$ where control variables as well as values of the input are ignored. Finally, $\mathrm{RA}_{\mathbb{Q}}$ can also simulate (division-free) arithmetic circuits (AC)

*Though normally called registers, for reasons that will be apparent later, we will refer to them as *variables* in this paper.

978-1-5090-3018-7/17/$31.00

without indeterminates. Originally, AC were introduced as a model for studying algebraic complexity [19], [20], but recently gain prominence as a model for analysing the complexity of numerical analysis [21] due to its succinct representation of numbers. We show that $RA_{\mathbb{Q}}$ can be used to represent numbers using roughly only twice as many transitions as the number of edges in the AC that represents the same number.

We study several decision problems for $RA_{\mathbb{Q}}$. The first one is the so-called *invariant* problem, which asks if the set of reachable configurations of a given $RA_{\mathbb{Q}}$ at a given state is *not* contained in a given *affine space*.† This is a typical decision problem in AP, where one would like to find out the relations among the variables when the program reaches a certain state [16], [22]. We show that the invariant problem for $RA_{\mathbb{Q}}$ is polynomial-time inter-reducible with another decision problem called the *non-zero* problem, which asks if a given $RA_{\mathbb{Q}}$ can output a non-zero value for some input word. Note that the non-zero problem is a generalization of the non-emptiness problem of RA, if we assume that an RA outputs the constant 1 in its final states. We show that the non-zero problem is decidable in exponential time (in the number of control variables, while polynomial in other parameters). Our algorithm is based on the well-known Karr's algorithm [16], [22] for deciding the same problem for AP.

We should remark that the exponential complexity is in the bit model, i.e., rational numbers are represented in their bit forms. If we assume that each rational number occupies only a constant space, e.g., the Blum-Shub-Smale model [23], the non-zero problem is PSPACE-complete, which matches the non-emptiness problem of standard RA [9].

In addition, we also prove a small model property on the length of the shortest word leading to a non-zero output. From that, we derive a polynomial space algorithm for the non-zero problem for the so-called *copyless* $RA_{\mathbb{Q}}$, i.e., $RA_{\mathbb{Q}}$ where reassignments to data variables are copyless‡. In fact, the non-zero problem becomes PSPACE-complete. It should be remarked that copyless $RA_{\mathbb{Q}}$ already subsume standard RA.

The separation of control and data variables is the key to make the non-zero problem decidable. In fact, allowing $RA_{\mathbb{Q}}$ to access just the *least significant bit* of their data variables is already enough to make them Turing-complete, and so is allowing order comparison between data variables. Without control variables, $RA_{\mathbb{Q}}$ become AP, positioning their invariant problem in PTIME [16], [22]. $RA_{\mathbb{Q}}$ without data variables are copyless, which makes their invariant problem PSPACE-complete (as mentioned above).

We also study the *commutativity* and *equivalence* problems for $RA_{\mathbb{Q}}$. The former asks whether a given $RA_{\mathbb{Q}}$ is commutative. A commutative $RA_{\mathbb{Q}}$ is an $RA_{\mathbb{Q}}$ that, given a word w as its input, outputs the same value on any permutation of w. The latter problem asks if two $RA_{\mathbb{Q}}$ are essentially the same, i.e., for every input word, the two $RA_{\mathbb{Q}}$ output the same set of values. The equivalence problem is known to be undecidable already for RA [15] via a reduction from *Post correspondence problem* (PCP). The same reduction can be used to show that the commutativity problem for RA is also undecidable. For deterministic $RA_{\mathbb{Q}}$, we show that the commutativity, equivalence, and invariant problems are inter-reducible to each other in polynomial time. Thus, for deterministic copyless $RA_{\mathbb{Q}}$ (and therefore also for deterministic RA), all problems mentioned above can be decided in polynomial space, and are, in fact, PSPACE-complete.

Finally, we also study the *reachability* problem for $RA_{\mathbb{Q}}$. This problem asks if a given $RA_{\mathbb{Q}}$ can output 0 for some input word. We show that although the reachability problem is undecidable in general, even when the $RA_{\mathbb{Q}}$ is deterministic, it is in NEXPTIME for nondeterministic copyless $RA_{\mathbb{Q}}$ with non-strict transition guards§. The decision procedure is obtained by a reduction to the configuration coverability problem of *rational vector addition systems with states* ($\mathbb{Q}$-VASS). Since there is an exponential blow-up in the reduction and the configuration coverability problem of $\mathbb{Q}$-VASS is in NP, we get a nondeterministic exponential-time decision procedure for the reachability problem of copyless $RA_{\mathbb{Q}}$ with non-strict transition guards.

An overview of the results obtained in this paper can be found in Table I. All decision problems we consider are natural and have corresponding applications. The invariant, equivalence, and reachability problems are standard decision problems considered in formal verification. RA and $RA_{\mathbb{Q}}$ are natural models of Reducer programs [3], [4] in the MapReduce paradigm [24], where commutativity is an important property required for Reducers [3], [25], [26].

Lastly, let us explain the main differences between the decision procedures for RA and those presented for $RA_{\mathbb{Q}}$. The non-emptiness and reachability problems for RA can essentially be reduced to the reachability problem in a finite-state system, where one can bound the number of data values and consider a finite alphabet. The commutativity and equivalence problems for deterministic RA can then be reduced to the non-emptiness problem. On the other hand, due to the use of arithmetic operations, similar techniques are no longer applicable in $RA_{\mathbb{Q}}$, thus, a different set of tools is then required such as Karr's algorithm and those from algebra and linear programming as used in this paper.

Organization: We review some basic linear algebra tools and Karr's algorithm in Section II. In Section III, we present the formal definition of $RA_{\mathbb{Q}}$. We discuss the invariant and non-zero problems in Section IV, and the commutativity and equivalence problems in Section V. In Section VI we discuss the reachability problem. We conclude with some discussions on related works and remarks in Sections VII and VIII. All missing technical details and proofs can be found in the long version of the paper [27].

†Formal definitions will be presented later on, including the representation of the given affine space.

‡The copyless constraint of $RA_{\mathbb{Q}}$ is inspired by and in the same flavour of the one for streaming transducers in [14].

§A guard is non-strict if it *does not* contain negations, i.e., it is a positive Boolean combination of inequalities $z \leqslant z'$.

TABLE I

OVERVIEW OF THE RESULTS (SV- means *single-valued*, CL- means *copyless*, NSTG- means *with non-strict transition guards*, reachability for (deterministic) RA means *state reachability*, -c means *complete*, UNDEC means *undecidable*)

Model	Non-zero (Emptiness)	Equivalence	Commutativity	Reachability
RA [9]	PSPACE-c [9]	UNDEC [15]	UNDEC (Thm. 5)	PSPACE-c [9]
deterministic RA [9]	PSPACE-c [9]	PSPACE-c [9]	PSPACE-c (Cor. 2)	PSPACE-c [9]
$\text{RA}_{\mathbb{Q}}$	EXPTIME (Thm. 2)	UNDEC (Thm. 5)	UNDEC (Thm. 5)	UNDEC (Thm. 7)
SV-$\text{RA}_{\mathbb{Q}}$	EXPTIME (Thm. 2)	UNDEC (Thm. 5)	UNDEC (Thm. 5)	UNDEC (Thm. 7)
CL-$\text{RA}_{\mathbb{Q}}$	PSPACE-c (Thm. 4)	UNDEC (Thm. 5)	UNDEC (Thm. 5)	?
deterministic $\text{RA}_{\mathbb{Q}}$	EXPTIME (Thm. 2)	EXPTIME (Cor. 1)	EXPTIME (Cor. 1)	UNDEC (Thm. 7)
deterministic CL-$\text{RA}_{\mathbb{Q}}$	PSPACE-c (Thm. 4)	PSPACE-c (Cor. 1)	PSPACE-c (Cor. 1)	?
NSTG-CL-$\text{RA}_{\mathbb{Q}}$	PSPACE-c (Thm. 4)	?	?	NEXPTIME (Thm. 8)

II. PRELIMINARIES

In this paper, a *word* w is a finite sequence of rational numbers $w = d_1 \cdots d_n \in \mathbb{Q}^*$. The *length* of w is n, denoted by $|w|$. The term *data value*, or *value* for short, means a rational number. Matrices and vectors are over the rational numbers $\mathbb{Q}$, where $\mathbb{Q}^{m\times n}$ and $\mathbb{Q}^k$ denote the sets of matrices of size $m \times n$ and column vectors of size k (i.e., $\mathbb{Q}^k = \mathbb{Q}^{k\times 1}$), respectively. All vectors in this paper are understood as column vectors.

We use $A, B, \dots$ to denote matrices, where $A(i, j)$ is the component in row i and column j of matrix A. We denote the transpose of A by A^t, and the determinant of a square matrix A by $\det(A)$. We use $\vec{a}, \vec{b}, \vec{u}, \vec{v}, \dots$ to denote vectors, where $\vec{u}(i)$ is the i-th component of vector $\vec{u}$ (numbered from 1).

When $\vec{u} \in \mathbb{Q}^k$ and $\vec{v} \in \mathbb{Q}^l$, we write $\begin{bmatrix}\vec{u}\\ \vec{v}\end{bmatrix}$ to denote a vector in $\mathbb{Q}^{k+l}$ composed as the concatenation of $\vec{u}$ and $\vec{v}$. Abusing the notation, we write 0 to also denote both the zero vector and the zero matrix.

For two vectors $\vec{u}, \vec{v} \in \mathbb{Q}^k$, we write $\vec{u} \geqslant \vec{v}$ when $\vec{u}(i) \geqslant \vec{v}(i)$ for each component $i = 1, \dots, k$. The dot product of $\vec{u}$ and $\vec{v}$ is denoted by $\vec{u}\cdot\vec{v}$.

Affine spaces: Recall that a *vector space* $\mathbb{V}$ in $\mathbb{Q}^k$ is a subset of $\mathbb{Q}^k$ that forms a group under addition $+$ and is closed under scalar multiplication, i.e., for all $\vec{v} \in \mathbb{V}$ and $\alpha \in \mathbb{Q}$, it holds that $\alpha\vec{v} \in \mathbb{V}$. The dimension of $\mathbb{V}$ is denoted by $\dim(\mathbb{V})$. The *orthogonal complement* of $\mathbb{V}$ is the vector space $\mathbb{V}^\perp = \{\vec{u} \mid \vec{u}\cdot\vec{v} = 0 \text{ for every } \vec{v} \in \mathbb{V}\}$. It is known that $\dim(\mathbb{V}^\perp) + \dim(\mathbb{V}) = k$.

An *affine space* $\mathbb{A}$ in $\mathbb{Q}^k$ is a set of the form $\vec{a} + \mathbb{V}$, where $\vec{a} \in \mathbb{Q}^k$ and $\mathbb{V}$ is a vector space in $\mathbb{Q}^k$. Here, $\vec{a}+\mathbb{V}$ denotes the set $\{\vec{a} + \vec{u} \mid \vec{u} \in \mathbb{V}\}$. The dimension of $\mathbb{A}$, denoted $\dim(\mathbb{A})$, is defined as $\dim(\mathbb{V})$.

A vector $\vec{u}$ is an *affine combination* of $V = \{\vec{a}_1, \dots, \vec{a}_n\}$, if there are $\lambda_1, \dots, \lambda_n \in \mathbb{Q}$ such that $\sum_{i=1}^n \lambda_i = 1$ and $\vec{u} = \sum_{i=1}^n \lambda_i \vec{a}_i$. We use $\mathsf{aff}(V)$ to denote the space of all affine combinations of V. It is known that for every affine space $\mathbb{A}$, there is a set V of size $\dim(\mathbb{A}) + 1$ such that $\mathsf{aff}(V) = \mathbb{A}$.

An *affine transformation* $T : \mathbb{Q}^k \to \mathbb{Q}^l$ is defined by a matrix $M \in \mathbb{Q}^{l\times k}$ and a vector $\vec{a} \in \mathbb{Q}^l$, such that $T\vec{x} = M\vec{x} + \vec{a}$. When $\vec{a} = 0$, T is called a *linear transformation*. From basic linear algebra, when $k = l$, it holds that T is a one-to-one mapping iff $\det(M) \neq 0$.

For convenience, we simply write *transformation* to mean affine transformation. Note that composing two transformations T_1 and T_2 yields another transformation $\vec{x} \mapsto T_2T_1\vec{x}$, where $\vec{x} \mapsto T_2T_1\vec{x}$ denotes a function that maps $\vec{x}$ to $T_2T_1\vec{x}$.

The following two lemmas will be useful.

Lemma 1. *Let $\mathbb{A} \subseteq \mathbb{Q}^k$ be an affine space and $T : \mathbb{Q}^{k+1} \to \mathbb{Q}^k$ be a transformation. Suppose there is a vector $\vec{v} \in \mathbb{Q}^k$ and values $d_1, d_2 \in \mathbb{Q}$, where $d_1 \neq d_2$, such that both $T\begin{bmatrix}\vec{v}\\ d_1\end{bmatrix}$ and $T\begin{bmatrix}\vec{v}\\ d_2\end{bmatrix}$ are in $\mathbb{A}$. Then, $T\begin{bmatrix}\vec{v}\\ d\end{bmatrix} \in \mathbb{A}$ for every $d \in \mathbb{Q}$.*

Lemma 2. *Let $T_1, \dots, T_m$ be transformations and $\vec{u}_1, \vec{u}_2, \dots, \vec{u}_{m+1}$ be vectors such that $\vec{u}_{i+1} = T_i\vec{u}_i$ for every $i = 1, \dots, m$. Let $\mathbb{H}$ be an affine space such that $\vec{u}_{m+1} \notin \mathbb{H}$ and $m \geqslant \dim(\mathbb{H}) + 2$. Then, there is a set of indices $J = \{j_1, \dots, j_n\}$ such that $1 \leqslant j_1 < j_2 < \cdots < j_n \leqslant m$, $|J| \leqslant \dim(\mathbb{H}) + 1$, and $T_{j_n}T_{j_{n-1}} \cdots T_{j_1}\vec{u}_1 \notin \mathbb{H}$.*

Affine programs: An *affine program* (AP) with n variables is a tuple $\mathcal{P} = (S, s_0, \mu)$, where S is a finite set of states, $s_0 \in S$ is the initial state, and μ is a finite set of transitions of the form (s_1, T, s_2), where $s_1, s_2 \in S$ and $T : \mathbb{Q}^n \to \mathbb{Q}^n$ is a transformation. Intuitively, $\mathcal{P}$ represents a program with n rational variables, say $z_1, \dots, z_n$. A transition $(s_1, T, s_2) \in \mu$ means that the program can move from state s_1 to s_2 while reassigning the contents of variables via $\vec{z} \mapsto T\vec{z}$, where $\vec{z}$ denotes the column vector of the variables $z_1, \dots, z_n$.

A *configuration* of $\mathcal{P}$ is a pair $(s, \vec{u}) \in S \times \mathbb{Q}^n$ where s is a state of $\mathcal{P}$ and $\vec{u}$ represents the contents of its variables. An *initial* configuration is a configuration $(s_0, \vec{u})$. A path in $\mathcal{P}$ from a configuration $(s, \vec{u})$ to a configuration $(s', \vec{v})$ is a sequence of transitions $(p_0, T_1, p_1), \dots, (p_{m-1}, T_m, p_m)$ of $\mathcal{P}$ such that $p_0 = s$, $p_m = s'$, and $\vec{v} = T_m \cdots T_1\vec{u}$.

The *AP invariant* problem is defined as follows: *Given an AP $\mathcal{P}$, a vector $\vec{u} \in \mathbb{Q}^n$, a state $s' \in S$, and an affine space $\mathbb{H}$, decide if there is a path in $\mathcal{P}$ from $(s_0, \vec{u})$ to $(s', \vec{v})$ for some $\vec{v} \notin \mathbb{H}$.* If there is such a path, then $\mathbb{H}$ is not an invariant for s' in $\mathcal{P}$ w.r.t. the initial configuration $(s_0, \vec{u})$. The input affine space $\mathbb{A} = \vec{a} + \mathbb{V}$ can be represented either as a pair $(\vec{a}, V)$ where V is a basis of $\mathbb{V}$, or as a set of vectors V where $\mathsf{aff}(V) = \mathbb{A}$. Either representation is fine as one can be easily transformed to the other.

The AP invariant problem can be solved in a polynomial time by the so-called Karr's algorithm [16], [22]. The main idea of Karr's algorithm is to compute, for every state $s \in S$, a set of vectors V_s such that the existence of a path from $(s_0, \vec{u})$ to $(s, \vec{v})$ implies $\vec{v} \in \mathsf{aff}(V_s)$. The algorithm works as follows: At the beginning, it sets $V_{s_0} = \{\vec{u}\}$ and $V_s = \emptyset$ for all other $s \neq s_0$. Then, using a worklist algorithm, it starts propagating the values of V_s over transitions such that for each transition $(s_1, T, s_2) \in \mu$ and each vector $\vec{v} \in V_{s_1}$ that has not been processed before, it adds the vector $T\vec{v}$ into V_{s_2}, if $T\vec{v} \notin \mathsf{aff}(V_{s_2})$. It holds that there is a path from $(s_0, \vec{u})$ to $(s', \vec{v})$, for some $\vec{v} \notin \mathbb{H}$, iff $V_{s'} \nsubseteq \mathbb{H}$. Note that we can limit the cardinality of V_s to be at most $n+1$, hence the algorithm runs in a polynomial time. We refer the reader to [16], [22] for more details.

Remark 1. *From Karr's algorithm, we can infer a small model property for the invariant problem. That is, if there is a path from $(s_0, \vec{u})$ to $(s', \vec{v})$, for some $\vec{v} \notin \mathbb{H}$, then there is such a path of length at most $(n+1)|S|$. Such a bound can also be derived in a more straightforward manner via Lemma 2, which we believe is interesting on its own. In fact, if all transformations in an AP are one-to-one, the bound $(n+1)|S|$ can be lowered to $(\dim(\mathbb{H})+2)|S|$, which can be particularly useful when $\dim(\mathbb{H})$ is small.*

III. Register automata over the rationals ($\mathrm{RA}_{\mathbb{Q}}$)

In the following, we fix $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_l\}$, two disjoint sets of variables called *control* and *data* variables, respectively. The vector $\vec{x}$ always denotes a vector of size k where $\vec{x}(i)$ is x_i. Likewise, $\vec{y}$ is of size l and $\vec{y}(i) = y_i$. We also reserve a special variable $\mathsf{cur} \notin X \cup Y$ to denote the data value currently read by the automaton.

Each variable in $X \cup Y$ can store a data value (these variables are sometimes called *registers*). When a vector $\vec{u} \in \mathbb{Q}^{k+l}$ is used to represent the contents of variables in $X \cup Y$, the first k components of $\vec{u}$ represent the contents of control variables, denoted by $\vec{u}(X)$, and the last l components represent the contents of data variables, denoted by $\vec{u}(Y)$. We also use $\vec{u}(x_i)$ and $\vec{u}(y_j)$ to denote the contents of x_i and y_j in $\vec{u}$, respectively.

A *linear constraint* over $X \cup \{\mathsf{cur}\}$ is a Boolean combination of atomic formulas of the form $z \leqslant z'$, where $z, z' \in X \cup \{\mathsf{cur}\}$. We write $\mathcal{C}(X, \mathsf{cur})$ to denote the set of all linear constraints over $X \cup \{\mathsf{cur}\}$. For convenience, we use $z < z'$ as an abbreviation of $\neg(z' \leqslant z)$. In the following, $\mathbb{P}^{k\times(k+1)}$ denotes the set of all 0-1 matrices in which the number of 1's in each row is exactly one. Intuitively, a matrix $A \in \mathbb{P}^{k\times(k+1)}$ denotes a mapping from $\{x_1, \ldots, x_k\}$ to $\{x_1, \ldots, x_k, \mathsf{cur}\}$.

Definition 1. *A* register automaton over the rationals *($RA_{\mathbb{Q}}$) with control and data variables (X, Y) is a tuple $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta\rangle$ defined as follows:*

- *Q is a finite set of states, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states.*
- *$\vec{u}_0 \in \mathbb{Q}^{k+l}$ is the initial contents of variables in $X \cup Y$.*
- *δ is a set of transitions whose elements are of the form*

$$t: \quad (p, \varphi(\vec{x}, \mathsf{cur})) \quad \rightarrow \quad (q, A, B, \vec{b}), \qquad (1)$$

where $p, q \in Q$ are states, $\varphi(\vec{x}, \mathsf{cur})$ is a linear constraint from $\mathcal{C}(\vec{x}, \mathsf{cur})$, and $A \in \mathbb{P}^{k\times(k+1)}$, $B \in \mathbb{Q}^{l\times(k+l+1)}$, $\vec{b} \in \mathbb{Q}^l$. The formula $\varphi(\vec{x}, \mathsf{cur})$ is called the guard *of t and the triple $(A, B, \vec{b})$ its* variable reassignment.

- *ζ is a mapping that maps each final state q_f to a linear function/expression $g(\vec{x}, \vec{y}) = \vec{a}\cdot\vec{x} + \vec{b}\cdot\vec{y} + c$, where $\vec{a} \in \mathbb{Q}^k$, $\vec{b} \in \mathbb{Q}^l$, and $c \in \mathbb{Q}$.*

The intuitive meaning of the transition in (1) is as follows. Suppose the contents of variables in $\vec{x}$ and $\vec{y}$ are $\vec{u}$ and $\vec{v}$, respectively. If $\mathcal{A}$ is in state p, currently reading data value c, and the guard $\varphi(\vec{u}, c)$ holds, then $\mathcal{A}$ can enter state q and reassign the variables $\vec{x}$ with $A\begin{bmatrix}\vec{u}\\ c\end{bmatrix}$ and $\vec{y}$ with $B\begin{bmatrix}\vec{u}\\ \vec{v}\\ c\end{bmatrix} + \vec{b}$.

Note that the matrix representation of the reassignment can be equivalently written as (i) a reassignment of each control variable in X with a variable in $X \cup \{\mathsf{cur}\}$, and (ii) a reassignment of each data variable in Y with a linear combination of variables in $X \cup Y \cup \{\mathsf{cur}\}$ and constants. We will therefore sometimes write the matrices A, B, and the vector $\vec{b}$ as reassignments to variables of the form $\{x_1 := r_1; \ldots; x_k := r_k; y_1 := s_1; \ldots; y_l := s_l\}$ or $\{\vec{x} := f(\vec{x}, \mathsf{cur});\ \vec{y} := g(\vec{x}, \vec{y}, \mathsf{cur})\}$. For readability, we omit identity reassignments such as $x_i := x_i$ or $y_j := y_j$ from the first form since the values of these variables do not change. A variable $x_i \in X$ is said to be *read-only* if, for each transition t, the reassignment to x_i in t is always of the form $x_i := x_i$.

Remark 2. *Note that in the definition above, the guards only allow comparisons among the current data value and the contents of variables in $\vec{x}$. One can easily generalize the guards so that comparisons with constants are allowed. Such a generalization does not affect the expressive power of $RA_{\mathbb{Q}}$, since every such a constant c can be stored into a fresh read-only control variable x_c in the initial assignment $\vec{u}_0$. This notation is chosen for technical convenience. On the other hand, for readability, in some of the examples later on, we do use comparisons with constants, which, strictly speaking, should be taken as comparisons with read-only control variables.*

A *configuration* of $\mathcal{A}$ is a pair $(q, \vec{u}) \in Q \times \mathbb{Q}^{k+l}$, where $\vec{u}$ denotes the contents of the variables. The *initial configuration* of $\mathcal{A}$ is $(q_0, \vec{u}_0)$, while final configurations are those with a final state in the left-hand component. A transition $t = (p, \varphi(\vec{x}, \mathsf{cur})) \rightarrow (q, A, B, \vec{b})$ and a value d entail a binary relation $(p, \vec{u}) \vdash_{t,d} (q, \vec{v})$, if

- $\varphi(\vec{u}(X), d)$ holds and
- $\vec{v}(X) = A\begin{bmatrix}\vec{u}(X)\\ d\end{bmatrix}$ and $\vec{v}(Y) = B\begin{bmatrix}\vec{u}(X)\\ \vec{u}(Y)\\ d\end{bmatrix} + \vec{b}$.

For a sequence of transitions $P = t_1 \cdots t_n$, we write $(q_0, \vec{u}_0) \vdash_P (q_n, \vec{u}_n)$ if there is a word $d_1 \cdots d_n$ such that $(q_0, \vec{u}_0) \vdash_{t_1,d_1} (q_1, \vec{u}_1) \vdash_{t_2,d_2} \cdots \vdash_{t_n,d_n} (q_n, \vec{u}_n)$. In this case, we say that $d_1 \cdots d_n$ *is compatible with* $t_1 \cdots t_n$. As usual, we write $(q_0, \vec{u}_0) \vdash^* (q_n, \vec{u}_n)$ if there exists a sequence of transitions P such that $(q_0, \vec{u}_0) \vdash_P (q_n, \vec{u}_n)$.

For an input word $w = d_1 \cdots d_n$, a *run* of $\mathcal{A}$ on w is a sequence $(q_0, \vec{u}_0) \vdash_{t_1,d_1} (q_1, \vec{u}_1) \vdash_{t_2,d_2} \cdots \vdash_{t_n,d_n} (q_n, \vec{u}_n)$, where $(q_0, \vec{u}_0)$ is the initial configuration and $t_1, \ldots, t_n \in \delta$. In this case, we also say $(q_0, \vec{u}_0) \vdash_{\mathcal{A},w} (q_n, \vec{u}_n)$. The run is *accepting* if $q_n \in F$, in which case $\mathcal{A}$ outputs the value $g(\vec{u}_n)$, where $\zeta(q_n) = g$, and we say that $\mathcal{A}$ accepts w. We write $\mathcal{A}(w)$ to denote the set of all outputs of $\mathcal{A}$ on w (i.e., if $\mathcal{A}$ does not accept w, we write $\mathcal{A}(w) = \emptyset$).

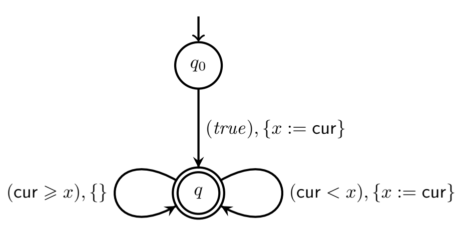
We say that $\mathcal{A}$ is *deterministic* if, for any state p and a pair of transitions $(p, \varphi(\vec{x}, \mathsf{cur})) \to (q, A, B, \vec{b})$ and $(p, \varphi'(\vec{x}, \mathsf{cur})) \to (q', A', B', \vec{b}')$ starting in p, the formula $\varphi(\vec{x}, \mathsf{cur}) \wedge \varphi'(\vec{x}, \mathsf{cur})$ is unsatisfiable. $\mathcal{A}$ is *complete* if, for every state p, the disjunction of guards on all transitions starting in p is valid. We say that $\mathcal{A}$ is *single-valued* if for every word w, $|\mathcal{A}(w)| \leqslant 1$. Note that different input words may yield different outputs. Evidently, every deterministic $\mathrm{RA}_{\mathbb{Q}}$ is single-valued.

We call $\mathcal{A}$ *copyless* if the reassignment of its data variables is of the form $\vec{y} := A\vec{y} + f(\vec{x}, \mathsf{cur})$, where f is a linear function and A is a 0-1 matrix where each column contains at most one 1. The intuition is that each variable y_i appears at most once in the right-hand side of the reassignment. For example, when $l = 2$, the reassignment $\{y_1 := y_1 + y_2; y_2 := 2x_1\}$ is copyless, while $\{y_1 := y_1 + y_2; y_2 := y_1\}$ is not, since y_1 appears twice in the right-hand side. Our definition of copyless is similar to the one for streaming transducers in [14].

Note that the standard register automata (RA) studied in [9], [11], [15] can be seen as a special case of $\mathrm{RA}_{\mathbb{Q}}$ without the data variables Y. Moreover, we can view the output function in each final state of an RA as a constant function that always outputs 1. Then, a standard RA can be seen as a single-valued as well as copyless $\mathrm{RA}_{\mathbb{Q}}$. Also note that affine programs in the sense of Karr [16], [22] are also a special case of $\mathrm{RA}_{\mathbb{Q}}$ in which control variables and input words are ignored.

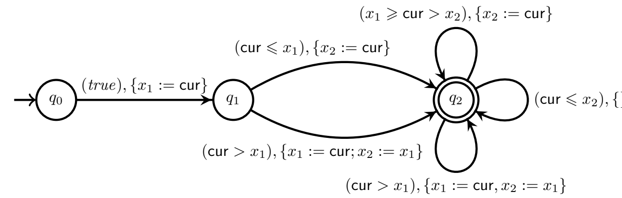
We present some typical aggregate functions that can be computed with $\mathrm{RA}_{\mathbb{Q}}$.

Computing the minimal value: The $\mathrm{RA}_{\mathbb{Q}}$ has one control variable x, as pictured below. The output function is $\zeta(q) = x$. The transition is pictured as $\varphi(\vec{x}, \mathsf{cur}), \{M\}$, where $\varphi(\vec{x}, \mathsf{cur})$ is the guard and M denotes the variable reassignment.



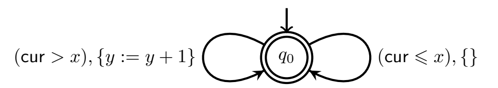
Intuitively, the $\mathrm{RA}_{\mathbb{Q}}$ starts by storing the first value in x. Every subsequent value cur is then compared with x and if $\mathsf{cur} < x$, it is stored in x via the reassignment $x := \mathsf{cur}$.

Computing the second largest element: The $\mathrm{RA}_{\mathbb{Q}}$ has control variables x_1 and x_2 and its output function is $\zeta(q_2) = x_2$.



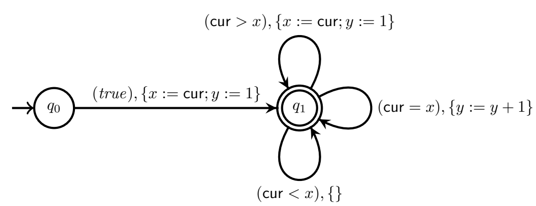
Intuitively, the $\mathrm{RA}_{\mathbb{Q}}$ stores the first two values in x_1 and x_2 in a decreasing order. Each subsequent value cur is compared with x_1 and x_2, which are updated if necessary.

Computing the number of elements larger than M: The $\mathrm{RA}_{\mathbb{Q}}$ has one control variable x and one data variable y with initial values M and 0, respectively. The output function is $\zeta(q_0) = y$.



Intuitively, each input value cur is compared with x. If $\mathsf{cur} > x$, the $\mathrm{RA}_{\mathbb{Q}}$ increments y by 1 via the reassignment $y := y+1$.

Computing the number of occurrences of the maximal element: The $\mathrm{RA}_{\mathbb{Q}}$ has one control variable x and one data variable y, with the initial value 0. The output function is $\zeta(q_1) = y$.



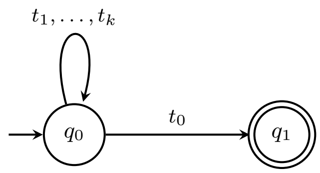
Intuitively, it stores the first value in x and reassigns $y := 1$. Every subsequent value cur is then compared with x. If it is the new largest element, it will be stored in x and the contents of y is reset to 1.

Proposition 1 below will be useful later on. Intuitively, it states that for a fixed sequence of transitions $t_1 \cdots t_n$, the contents of variables of $\mathcal{A}$ are a linear combination of the values in the input word $d_1 \cdots d_n$, provided that $d_1 \cdots d_n$ is compatible with $t_1 \cdots t_n$. Its proof can be done by a straightforward induction on n and is, therefore, omitted.

Proposition 1. (Linearity of $\mathrm{RA}_{\mathbb{Q}}$) *Let $\mathcal{A}$ be an $\mathrm{RA}_{\mathbb{Q}}$ over (X, Y). For every sequence $t_1 \cdots t_n$ of transitions of $\mathcal{A}$, there is a matrix $M \in \mathbb{Q}^{(k+l) \times n}$ and a vector $\vec{a} \in \mathbb{Q}^{k+l}$ such that for every word $d_1 \cdots d_n$ compatible with $t_1 \cdots t_n$, where $(q_0, \vec{u}_0) \vdash_{t_1,d_1} \cdots \vdash_{t_n,d_n} (q_n, \vec{u}_n)$, it holds that*

$$\vec{u}_n \quad = \quad M \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} + \vec{a}.$$

The following example shows that $\mathrm{RA}_{\mathbb{Q}}$ can be used to represent positive integers succinctly. Let p and n be positive integers, k be an integer such that $k = \lceil \log n \rceil$, and $\mathcal{A}$ be an $\mathrm{RA}_{\mathbb{Q}}$ as illustrated below.



$\mathcal{A}$ is over $X = \{x_1, \ldots, x_k\}$ and $Y = \{y\}$. The initial state of $\mathcal{A}$ is q_0 and q_1 is its final state. The initial contents of the variables are $(b_1, \ldots, b_k, 1)$, where $b_k \cdots b_1$ is the binary representation of n, i.e., $n = \sum_{i=1}^{k} b_i 2^{i-1}$.

The transition t_0 is $(q_0, \bigwedge_{i=1}^{k} x_i = 0) \to (q_1, \{\})$. For each $i = 1, \ldots, k$, the transition t_i is defined as follows:

$$\begin{aligned}&(q_0, x_i = 1 \wedge \bigwedge_{j=1}^{i-1} x_j = 0)\\ &\to (q_0, \{x_1 := 1; \ldots; x_{i-1} := 1; x_i := 0; y := p \cdot y\}).\end{aligned}$$

Recall that when a variable's value in a reassignment is not specified, its value stays the same. Therefore, in t_0, the contents of all variables stay the same, while in t_i, the contents of $x_{i+1}, \ldots, x_k$ stay the same. We define the output function $\zeta(q_1)$ to output y.

Intuitively, the contents of variables $\vec{x} = (x_1, \ldots, x_k)$ represent a number between n and 0 in binary, where the least significant bit is stored in x_1. $\mathcal{A}$ starts with $\vec{x}$ containing the binary representation of n, and iterates through all integers from n down to 1. On each iteration, it takes one of the transitions $t_1, \ldots, t_k$ that "decrements" the number represented by $\vec{x}$, and multiplies the contents of y by p. When the number in $\vec{x}$ reaches 0, it takes transition t_0 and moves to state q_1. Note that the outcome of $\mathcal{A}$ does not depend on the input word and it always output p^n regardless on the input. Moreover, $\mathcal{A}$ has only $\lceil \log(n) \rceil + 1$ transitions. In fact, one can obtain an $\mathrm{RA}_{\mathbb{Q}}$ that always outputs $p_1^{n_1} \cdots p_k^{n_k}$ by constructing one $\mathrm{RA}_{\mathbb{Q}}$ for each $p_i^{n_i}$ and composing them sequentially. The final $\mathrm{RA}_{\mathbb{Q}}$ has at most $\sum_{i=1}^{k}(\lceil \log(n_i) \rceil + 1)$ transitions.

Motivated by the example above, we say that *an $\mathrm{RA}_{\mathbb{Q}}$ $\mathcal{A}$ represents a positive integer n* if it has exactly one possible output n. With this representation, $\mathrm{RA}_{\mathbb{Q}}$ can simulate arithmetic circuits as stated below.

Theorem 1. *For every arithmetic circuit C (division-free and without indeterminates), there is an $\mathrm{RA}_{\mathbb{Q}}$ $\mathcal{A}$ that represents the same number as C with the number of transitions linearly proportional to the number of edges in C. If C is additive or multiplicative, $\mathcal{A}$ uses only one data variable. Moreover, $\mathcal{A}$ can be constructed in time linear in the size of C.*

The number of transitions in $\mathcal{A}$ is roughly twice the number of edges in C, plus the number of constants in C.

IV. The invariant problem for $\mathrm{RA}_{\mathbb{Q}}$

In this section, we will, in the same spirit as Karr [16], consider the *invariant* problem for $\mathrm{RA}_{\mathbb{Q}}$. For an $\mathrm{RA}_{\mathbb{Q}}$ $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ and a state $q \in Q$, define $\mathsf{vec}(\mathcal{A}, q) = \{\vec{v} \mid (q_0, \vec{u}_0) \vdash^* (q, \vec{v})\}$, i.e., $\mathsf{vec}(\mathcal{A}, q)$ contains all vectors representing the contents of control and data variables when $\mathcal{A}$ reaches state q. The *invariant* problem is defined as: *Given an $\mathrm{RA}_{\mathbb{Q}}$ $\mathcal{A}$, a state q of $\mathcal{A}$, and an affine space $\mathbb{H}$, decide whether* $\mathsf{vec}(\mathcal{A}, q) \not\subseteq \mathbb{H}$. Again, an affine space $\mathbb{A} = \vec{a} + \mathbb{V}$ can be represented either as a pair $(\vec{a}, V)$ where V is a basis of $\mathbb{V}$, or as a set V where $\mathsf{aff}(V) = \mathbb{A}$. The invariant problem is tightly related to the *must-constancy* problem for programs [28], which asks, for a given program location ℓ, a given variable z and a given constant c, whether the value of z in ℓ must be equal to c.

Instead of the invariant problem, it will be more convenient to consider another, but equivalent, problem, which we call the *non-zero* problem, defined as follows. *Given an $\mathrm{RA}_{\mathbb{Q}}$ $\mathcal{A}$, decide whether there is w such that $\mathcal{A}(w) \not\subseteq \{0\}$, i.e., whether $\mathcal{A}$ outputs a non-zero value on some word w.* We abuse notation and simply write $\mathcal{A}(w) \neq 0$ to denote that there is $c \in \mathcal{A}(w)$ such that $c \neq 0$. The non-zero problem can, therefore, be written as: *Given an $\mathrm{RA}_{\mathbb{Q}}$ $\mathcal{A}$, decide whether there is w such that $\mathcal{A}(w) \neq 0$.*

The two problems are, in fact, Karp inter-reducible. The reduction from the non-zero problem to the invariant problem is as follows. Let $\mathcal{A}$ be the input to the non-zero problem, $q_1, \ldots, q_m$ be the final states of $\mathcal{A}$, and ζ be the mapping that specifies the output functions for the final states. Let $\mathcal{A}'$ be the $\mathrm{RA}_{\mathbb{Q}}$ obtained by adding a new state q_f into $\mathcal{A}$, and for every q_i adding the following transition: $(q_i, \mathit{true}) \to (q_f, \{y_1 := \zeta(q_i)\})$. $\mathcal{A}'$ has only one final state q_f, whose output function yields y_1. The reduction follows by the fact that there is w such that $\mathcal{A}(w) \neq 0$ iff $\mathsf{vec}(\mathcal{A}, q_f) \not\subseteq \mathbb{H}$, where $\mathbb{H}$ is the space of the solutions $\zeta(q_f)(\vec{x}, \vec{y}) = 0$.

Vice versa, the invariant problem reduces to the non-zero problem as follows. Let $\mathcal{A}$, q, and $\mathbb{H} = \vec{a} + \mathbb{V}$ be the input to the invariant problem. Let $\{\vec{v}_1, \ldots, \vec{v}_m\}$ be a basis of $\mathbb{V}^{\perp}$, the orthogonal complement of $\mathbb{V}$, which can be obtained by Gaussian elimination on a basis of $\mathbb{V}$ in polynomial time. Let $\mathcal{A}'$ be the $\mathrm{RA}_{\mathbb{Q}}$ obtained by adding the following into $\mathcal{A}$:

- $m + 1$ new states $q_1, \ldots, q_m$ and p,
- $m + 1$ new data variables $y_1, \ldots, y_m$ and z,
- for each q_i, the pair of transitions $(q, \mathit{true}) \to (q_i, \{y_i := (\begin{bmatrix}\vec{x}\\ \vec{y}\end{bmatrix} - \vec{a}) \cdot \vec{v}_i\})$ and $(q_i, \mathit{true}) \to (p, \{z := y_i\})$.

Further, set p as the only final state of $\mathcal{A}'$ and set its output function to yield z. The reduction follows from the fact that $\vec{u} \in \mathbb{H}$ iff $(\vec{u} - \vec{a}) \cdot \vec{v}_i = 0$ for every $i = 1, \ldots, m$, thus, $\mathsf{vec}(\mathcal{A}, q) \not\subseteq \mathbb{H}$ iff there is a word w such that $\mathcal{A}'(w) \neq 0$.

A. The algorithm and a small model property

In this section, we present an exponential-time algorithm for the non-zero problem of $\mathrm{RA}_{\mathbb{Q}}$. Let $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ be the input $\mathrm{RA}_{\mathbb{Q}}$ over (X, Y), where $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_l\}$. The main idea of the presented algorithm is to transform $\mathcal{A}$ into an affine program $\mathcal{P}_{\mathcal{A}}$ and analyse $\mathcal{P}_{\mathcal{A}}$ using Karr's algorithm.

We start with some necessary definitions. An *ordering* of X is a total preorder ϕ on X, i.e., $\phi = z_1 \circledast_1 z_2 \circledast_2 \cdots \circledast_{k-1} z_k$, where $(z_1, \ldots, z_k)$ is a permutation of $(x_1, \ldots, x_k)$ and each $\circledast_i$ is either $<$ or $=$. An ordering ϕ is *consistent* with a transition $(p, \varphi(\vec{x}, \mathsf{cur})) \to (q, A, B, \vec{b})$, if $\varphi(\vec{x}, \mathsf{cur}) \wedge \phi$ is satisfiable.

We say that an ordering ϕ holds in a configuration $(q, \vec{u})$ if it holds when we substitute $(x_1, \ldots, x_k)$ with $\vec{u}(X)$. In this case, we say that the ordering of $(q, \vec{u})$ is ϕ.

The construction of $\mathcal{P}_{\mathcal{A}}$ is based on the following lemma.

Lemma 3. *Let $\mathbb{H}$ be an affine space and*

$$(q_1, \vec{u}_1) \vdash_{t_1,d_1} \cdots\cdots \vdash_{t_m,d_m} (q_{m+1}, \vec{u}_{m+1})$$

be a run of $\mathcal{A}$ on a word $d_1 \cdots d_m$ such that $\vec{u}_{m+1} \notin \mathbb{H}$. Then there is a run of $\mathcal{A}$ on a word $c_1 \cdots c_m$, say

$$(q_1, \vec{v}_1) \vdash_{t_1,c_1} \cdots\cdots \vdash_{t_m,c_m} (q_{m+1}, \vec{v}_{m+1}),$$

such that $\vec{u}_1 = \vec{v}_1$, $\vec{v}_{m+1} \notin \mathbb{H}$, and for every $i = 1, \ldots, m+1$ the following holds:

(a) $(q_i, \vec{u}_i)$ and $(q_i, \vec{v}_i)$ have the same ordering.
(b) If $d_i = \vec{u}_i(X)(j)$ for some j, then $c_i = \vec{v}_i(X)(j)$.
(c) If $d_i < \vec{u}_i(X)(j)$, where $\vec{u}_i(X)(j)$ is the minimal value in $\vec{u}_i(X)$, then either $c_i = \vec{v}_i(X)(j) - 1$ or $c_i = \vec{v}_i(X)(j) - 2$.
(d) If $d_i > \vec{u}_i(X)(j)$, where $\vec{u}_i(X)(j)$ is the maximal value in $\vec{u}_i(X)$, then either $c_i = \vec{v}_i(X)(j) + 1$ or $c_i = \vec{v}_i(X)(j) + 2$.
(e) If $\vec{u}_i(X)(j) < d_i < \vec{u}_i(X)(j')$, where $\vec{u}_i(X)(j)$ is the maximal value in $\vec{u}_i(X)$ less than d_i and $\vec{u}_i(X)(j')$ is the minimal value in $\vec{u}_i(X)$ greater than d_i, then either $c_i = \frac{1}{3}\vec{v}_i(X)(j) + \frac{2}{3}\vec{v}_i(X)(j')$ or $c_i = \frac{2}{3}\vec{v}_i(X)(j) + \frac{1}{3}\vec{v}_i(X)(j')$.

Intuitively, Lemma 3 states that for a run $(q_1, \vec{u}_1) \vdash_{\mathcal{A},w} (q_{m+1}, \vec{u}_{m+1})$ on a word $w = d_1 \cdots d_m$ such that $\vec{u}_{m+1}$ does not belong to the affine space $\mathbb{H}$, we can assume that each d_i is a linear combination of components in $\vec{u}_i$. We note that the choice of the constants $\pm 1, \pm 2, \frac{1}{3}, \frac{2}{3}$ in items (c)-(e) is arbitrary and was made to ensure that there are at least two possible different values for c_i, since by Lemma 1, one of them is guaranteed to hit outside $\mathbb{H}$. Other constants satisfying the right conditions would work, too.

With Lemma 3, we then transform $\mathcal{A}$ to an affine program $\mathcal{P}_{\mathcal{A}}$ and apply Karr's algorithm on $\mathcal{P}_{\mathcal{A}}$ to decide the non-zero problem. Essentially, the set of states in $\mathcal{P}_{\mathcal{A}}$ is the Cartesian product of Q and the set of orderings of $X \cup \{\mathsf{cur}\}$. The number of variables in $\mathcal{P}$ is $k + l$. There are altogether $2^k(k+1)!$ orderings of $X \cup \{\mathsf{cur}\}$, so the algorithm runs in an exponential time, as stated in Theorem 2 below.

Theorem 2. *The non-zero problem for RA$_{\mathbb{Q}}$ is in* EXPTIME.

Note that the number of states in the affine program $\mathcal{P}_{\mathcal{A}}$ is $|Q|2^k(k+1)!$. By Remark 1, we can obtain a similar small model property for RA$_{\mathbb{Q}}$, as stated below.

Theorem 3. (A small model property for RA$_{\mathbb{Q}}$) *If there is a word $w \in \mathbb{Q}^*$ such that $\mathcal{A}(w) \neq 0$, then there is a word $w' \in \mathbb{Q}^*$ such that $\mathcal{A}(w') \neq 0$ and $|w'| \leqslant |Q|(k+l+1)2^k(k+1)!$.*

One can also prove Theorem 3 without relying on Karr's algorithm. Moreover, we can show that the exponential bound is, in fact, tight (see [27] for proofs of both claims).

We should remark that the exponential complexity is in the bit model, i.e., rational numbers are represented in their bit forms. As stated in Theorem 1, an RA$_{\mathbb{Q}}$ can simulate an arithmetic circuit and store in its data variables values that are doubly-exponentially large (w.r.t. the number of control variables), which occupy an exponential space. For example, if the initial value of a data variable y is 1 and every transition contains the reassignment $y := 2y$, the final value of y may be up to $2^{|Q|(k+l+1)2^k(k+1)!}$. However, if we assume that rational numbers between -1 and 1 occupy only constant space, the non-zero problem is in PSPACE (by guessing a path of exponential length as in Theorem 3), which matches the non-emptiness problem of standard RA [9].

B. Polynomial-space algorithm for copyless RA$_{\mathbb{Q}}$

In the following, let $\mathcal{A}$ be a copyless RA$_{\mathbb{Q}}$ with k control variables and l data variables. W.l.o.g., we assume that every transition of $\mathcal{A}$ is of the form $(p, \varphi(\vec{x}, \mathsf{cur})) \to (q, A, B, 0)$, i.e., $\vec{b} = 0$ (every RA$_{\mathbb{Q}}$ can be transformed to this form by adding new control variables to store the non-zero constants in $\vec{b}$). Recall that copyless RA$_{\mathbb{Q}}$ are still a generalization of standard RA, thus, the non-zero problem is PSPACE-hard. In the following we will show that the non-zero problem is in PSPACE. We need the following lemma.

Lemma 4. *Let $\mathbb{H}$ be an affine space and*

$$(q_1, \vec{u}_1) \vdash_{t_1,d_1} \cdots\cdots \vdash_{t_m,d_m} (q_{m+1}, \vec{u}_{m+1}),$$

be a run of $\mathcal{A}$ on a word $d_1 \cdots d_m$ such that $\vec{u}_{m+1} \notin \mathbb{H}$. Then, there exists a run of $\mathcal{A}$ on a word $c_1 \cdots c_m$, say

$$(q_1, \vec{v}_1) \vdash_{t_1,c_1} \cdots\cdots \vdash_{t_m,c_m} (q_{m+1}, \vec{v}_{m+1}),$$

such that $\vec{u}_1 = \vec{v}_1$, $\vec{v}_{m+1} \notin \mathbb{H}$, and for every $i = 1, \ldots, m+1$, if c_i does not appear in $\vec{v}_i(X)$, then $c_i \neq c_j$ for every $j \leqslant i-1$.

Intuitively, Lemma 4 states that for the non-zero problem, it is sufficient to consider only words $c_1 \cdots c_m$ such that if $\mathcal{A}$ encounters a value c_i that is not in its control variables, then c_i is indeed new, i.e., it has not appeared in $c_1 \cdots c_{i-1}$. Another way of looking at it is that once $\mathcal{A}$ "forgets" a value c_i, i.e., c_i no longer appears in its control variables, then c_i will never appear again in the future.

We start with the following observation. Let $w = d_1 \cdots d_m$ be a word. Suppose $(q_0, \vec{u}_0) \vdash_{\mathcal{A},w} (q_m, \vec{u}_m)$, where q_m is a final state. By linearity of RA$_{\mathbb{Q}}$ (cf. Proposition 1), $\vec{u}_m = M[d_1 \cdots d_m]^t + \vec{a}$, for some M and $\vec{a}$. Let $\zeta(q_m)(\vec{x}, \vec{y}) = \vec{c} \cdot \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} + b$. Then, for some $\alpha_1, \ldots, \alpha_m, \beta$,

$$\zeta(q_m)(\vec{u}_m) \quad = \quad \alpha_1 d_1 + \cdots + \alpha_m d_m + \beta.$$

Suppose $d'_1, \ldots, d'_n$ are the distinct values occurring in $d_1 \cdots d_m$. Therefore, for some $\alpha'_1, \ldots, \alpha'_n$, it holds that

$$\zeta(q_m)(\vec{u}_m) \quad = \quad \alpha'_1 d'_1 + \cdots + \alpha'_n d'_n + \beta.$$

We can assume that the values $d'_1, \ldots, d'_k$ are the initial contents of control variables. For simplicity, we can also assume that the initial contents of control variables are pairwise different and that all initial values stored in control variables occur in $d_1 \cdots d_m$. We observe the following:

- If there is $i > k$ such that $\alpha'_i \neq 0$, then we can assume that $\zeta(q_m)(\vec{u}_m) \neq 0$. To show why, suppose to the contrary that $\zeta(q_m)(\vec{u}_m) = 0$. From the assumption, the value d'_i does not appear in the initial contents of control variables. When d'_i first appears in the input word, by density of rational numbers, we can increase d'_i by some small number $\epsilon > 0$, i.e., replace d'_i with $d'_i + \epsilon$, and still obtain a run from q_0 to q_m. The output will now be
$$\alpha'_1 d'_1 + \cdots + \alpha'_i(d'_i + \epsilon) + \cdots + \alpha'_n d'_n + \beta = \alpha'_i \epsilon,$$
which will be non-zero, since both ϵ and α'_i are non-zero.
- If, for all $i > k$, it holds that $\alpha'_i = 0$, then $\zeta(q_m)(\vec{u}_m) \neq 0$ if and only if $\alpha'_1 d'_1 + \cdots + \alpha'_k d'_k + \beta \neq 0$.

Note that our observation above holds for general RA$_{\mathbb{Q}}$. In general, the number of bits for storing α'_i can be exponentially large, but as we will see later, is polynomially bound for copyless RA$_{\mathbb{Q}}$.

The algorithm works by simulating a run of $\mathcal{A}$ of length at most $|Q|(k + l + 1)2^k(k + 1)!$ starting from the initial configuration. During the simulation, when $\mathcal{A}$ assigns new values into control variables, the algorithm only remembers the ordering of control variables, not the actual data values assigned. Such an ordering is sufficient to simulate a run. The algorithm will then try to nondeterministically guess the first position of the word where a value d'_i such that $i > k$ and $\alpha'_i \neq 0$ occurs. Again, the algorithm does not guess the actual value d'_i but, instead, only remembers the names of the control variables that d'_i is assigned to.

In the rest of the simulation, the algorithm keeps track of how many "copies" of d'_i have been added to each data variable. For example, suppose d'_i is stored in a control variable x_j and the reassignment for y in a transition t is of the form $y := y + y' + c'x_j$. Then, the number of copies of d'_i in y is obtained by adding the number of copies of d'_i in y and y', plus c' copies of d'_i. Note that due to being copyless, the assignment to y' in t cannot use the original value of y', which is lost.

When the value d'_i is forgotten in $\mathcal{A}$, i.e., d'_i is not stored in any control variable any more, we can assume d'_i will not appear again in the input word (by Lemma 4). During the simulation, the algorithm keeps for every data variable y a track of how many copies of d'_i are stored in y. Every time the algorithm reaches a final state, it applies the output function of the state and checks whether $\alpha'_i \neq 0$.

Based on the property that $\mathcal{A}$ is copyless, we notice that in one step, the sum of *all* data variables may increase by at most $f(\vec{x}, \mathsf{cur})$, for some linear function f, where the constants in f come from those in the transition. By Theorem 3, during any run, the number of bits occupied by the sum of the "coefficients" of d'_i in *all* data variables is at most $c \cdot \log(|Q|(k + l + 1)2^k(k + 1)!)$, where c is the sum of all bits occupied by the constants in the transitions. Thus, each α'_i occupies only a polynomial space.

If for all $i > k$ it holds that $\alpha'_i = 0$, the algorithm counts $\alpha'_1, \ldots, \alpha'_k$ instead. Recall that $d'_1, \ldots, d'_k$ are the data values of the initial contents of control variables. The algorithm performs the counting during the simulation of a run in a similar way as above. When d'_i no longer appears in any control variable, the counting stops and the simulation simply continues by remembering the state and the ordering of control variables. When a final state is reached, the algorithm verifies that $\alpha'_1 d'_1 + \cdots + \alpha'_k d'_k + \beta \neq 0$. Again, each α'_i occupies only a polynomial space, thus, the whole algorithm runs in a polynomial space. Since the non-emptiness problem for standard RA is already PSPACE-hard, we conclude with the following theorem.

Theorem 4. *The non-zero problem for copyless RA$_{\mathbb{Q}}$ is* PSPACE*-complete.*

It is tempting to directly use Theorem 3 to prove Theorem 4 by simulating the run directly instead of tracing the coefficients of input values. In doing so, however, the number of bits may increase in each step. For example, suppose an RA$_{\mathbb{Q}}$ has two control variables x_1 and x_2 storing 0.01 and 0.1 (in binary), respectively, and its transitions have the guard $x_1 < \mathsf{cur} < x_2$ with the reassignment $\{x_1 := \mathsf{cur}\}$. Straightforward guessing by adding one bit 1 at the end of x_1 will result in the number of bits in x_1 increasing by one in each step. A similar thing can happen if guessing a path in the affine program generated from Lemma 3: the numbers cannot be represented in a polynomial space because of the multiplication with $\frac{1}{3}$ and $\frac{2}{3}$.

Furthermore, note that the algorithm is correct also for general (i.e., not only copyless) RA$_{\mathbb{Q}}$. The restriction to copyless RA$_{\mathbb{Q}}$ allows us to guarantee that the space used by the algorithm is polynomial. If we applied the algorithm to general RA$_{\mathbb{Q}}$, it would require an exponential space.

C. Some remarks on the non-zero problem

In this section, we have shown that the non-zero problem for RA$_{\mathbb{Q}}$ is in EXPTIME, and the problem itself is a generalization of the non-emptiness problem for standard RA. Our algorithm relies heavily on the fact that the variables are partitioned into two groups: control variables, which "control" the computation flow, and data variables, which accumulate data about the input word. Without control variables, an RA$_{\mathbb{Q}}$ is similar to an affine program, thus the non-zero problem drops to PTIME. Without data variables, the non-zero problem becomes PSPACE-complete, as an RA$_{\mathbb{Q}}$ without data variables is a special case of a copyless RA$_{\mathbb{Q}}$ but still a generalization of a standard RA.

It is also important that RA$_{\mathbb{Q}}$ have no access to data variables at all. If we allow comparison between two data variables, the non-zero problem becomes undecidable. In fact, even if we allow RA$_{\mathbb{Q}}$ to access only one bit of information from data variables, say, the least significant bit of the integer part of a rational number, RA$_{\mathbb{Q}}$ can simulate Turing machines. In particular, the contents of a Turing machine (two-way infinite) tape can be represented as a rational number. For example, if the contents of the tape is $\sqcup^\omega 0\underline{0}1 \sqcup^\omega$ (with the underline indicating the position of the head and $\sqcup$ denoting a blank space), its representation by a rational number can be e.g. 1010.11, where $0, 1$, and $\sqcup$ are encoded by $10, 11$, and 00, respectively. The head moving right and left can be simulated by multiplying the data variable by 4 and $\frac{1}{4}$, respectively.

V. THE EQUIVALENCE AND COMMUTATIVITY PROBLEMS

In this section we study the equivalence and the commutativity problems. The equivalence problem is defined as follows: *Given two RA$_{\mathbb{Q}}$ $\mathcal{A}$ and $\mathcal{A}'$, decide if $\mathcal{A}(w) = \mathcal{A}'(w)$ for all words w.* On the other hand, the commutativity problem is defined as follows: *Given an RA$_{\mathbb{Q}}$ $\mathcal{A}$, decide if for all words w and w' such that $w' \in perm(w)$, it holds that $\mathcal{A}(w) = \mathcal{A}(w')$, where $perm(w)$ is the set of all permutations of the word w.*

Theorem 5. *For single-valued RA$_{\mathbb{Q}}$, the commutativity and equivalence problems are both undecidable.*

Theorem 5 can be proved by a reduction similar to the one used in [15] for proving undecidability of the equivalence problem for standard RA. For deterministic RA$_{\mathbb{Q}}$, however, both problems become inter-reducible (via a Cook reduction) with the non-zero problem, as stated below.

Theorem 6. *For deterministic RA$_{\mathbb{Q}}$, the equivalence problem, the commutativity problem, the non-zero problem, and the invariant problem are all inter-reducible in polynomial time.*

In the following paragraph, we present the main ideas of the proofs.

From non-zero to invariant and vice versa: Note that the Karp reductions from Section IV cannot be used, because they construct nondeterministic RA$_{\mathbb{Q}}$. Instead, we modify them into Cook reductions such that for every final state of the RA$_{\mathbb{Q}}$ in the non-zero problem, we create one invariant test. On the other hand, for the invariant problem, suppose that $\mathbb{H} = \vec{a} + \mathbb{V}$ and let us take a basis $\{\vec{v}_1, \ldots, \vec{v}_m\}$ for $\mathbb{V}^{\perp}$ (the orthogonal complement of $\mathbb{V}$). Then for each $\vec{v}_i$ in the basis, we create a new RA$_{\mathbb{Q}}$ with a single final state with a corresponding output function. Notice that the reductions preserve the (deterministic) structure of the RA$_{\mathbb{Q}}$.

From equivalence to non-zero: The proof is by a standard product construction. Given two deterministic RA$_{\mathbb{Q}}$ $\mathcal{A}_1$ and $\mathcal{A}_2$ (w.l.o.g. we assume they are both complete), we can construct in a polynomial time a deterministic RA$_{\mathbb{Q}}$ $\mathcal{A}$ such that $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent iff $\mathcal{A}(w) = 0$ for all w. The states of $\mathcal{A}$ are of the form (q_1, q_2), where q_1 is a state from $\mathcal{A}_1$ and q_2 from $\mathcal{A}_2$. A state (q_1, q_2) is final iff at least one of q_1 and q_2 is final, and the output function is defined as either (i) the difference of the outputs of q_1 and q_2 if both q_1 and q_2 are final, or (ii) the constant 1 if exactly one of them is final.

From non-zero to commutativity: Let $\mathcal{A}$ be a deterministic RA$_{\mathbb{Q}}$. We assume w.l.o.g. that for all w, $|\mathcal{A}(w)| = 1$, i.e., $\mathcal{A}$ outputs a value on all inputs w. We construct a deterministic RA$_{\mathbb{Q}}$ $\mathcal{A}'$ with outputs defined as follows:

- For words where the first and the second values are 1 and 2 respectively, i.e., words of the form $v = 12w$, we define $\mathcal{A}'(v) = \mathcal{A}(w)$
- For all other words, $\mathcal{A}'$ outputs 0.

The construction of $\mathcal{A}'$ takes only linear time by adding two new states that check the first two values and a new final state that outputs the constant 0 for words not in the form of $12w$. If there is a word w such that $\mathcal{A}(w) \neq 0$, then $\mathcal{A}'(12w) \neq 0$, so $\mathcal{A}'$ is not commutative (because $\mathcal{A}'(21w) = 0$). On the other hand, if $\mathcal{A}'$ is not commutative, it means that there is an input for which the output is not 0.

From commutativity to equivalence: The idea of the proof is similar to the one used in [25] to prove decidability of the commutativity problem of two-way finite automata. A similar idea was also used in [3] for the same problem over *symbolic numerical transducers*, which are a strict subclass of RA$_{\mathbb{Q}}$.

We define two permutation functions π_1 and π_2 on words as follows: let $\pi_1(d_1 d_2 \cdots d_n) = d_2 d_1 \cdots d_n$ (swap the first two symbols) and $\pi_2(d_1 d_2 \cdots d_n) = d_2 \cdots d_n d_1$ (move the first symbol to the end of the input word). It is known that every permutation is a composition of π_1 and π_2 [29].

Given a deterministic RA$_{\mathbb{Q}}$ $\mathcal{A}$, it holds that $\mathcal{A}$ is commutative iff the following equations hold for every word w:

$$\mathcal{A}(w) \ = \ \mathcal{A}(\pi_1(w)) \ = \ \mathcal{A}(\pi_2(w))$$

As a consequence, we can reduce the commutativity problem to the equivalence problem by constructing deterministic RA$_{\mathbb{Q}}$ $\mathcal{A}_1$ and $\mathcal{A}_2$ such that for every word w, $\mathcal{A}_1(w) = \mathcal{A}(\pi_1(w))$ and $\mathcal{A}_2(w) = \mathcal{A}(\pi_2(w))$.

While the construction of $\mathcal{A}_1$ is straightforward, the construction of $\mathcal{A}_2$ is more involved. The standard way to construct $\mathcal{A}_2$ involves nondeterminism to "guess" that the next transition is the last one. However, here we require $\mathcal{A}_2$ to be deterministic. Our trick is to use a new set of variables to simulate the process of guessing in a deterministic way.

Corollary 1. *The commutativity and equivalence problems for deterministic RA$_{\mathbb{Q}}$ are in* EXPTIME. *They become* PSPACE-*complete for deterministic copyless RA$_{\mathbb{Q}}$.*

All upper bounds follow from the results in Section IV. The PSPACE-hardness can be obtained using a reduction similar to the one in [9]. Moreover, we can use the ideas in the proof of Theorem 6 also for standard RA to obtain the following corollary.

Corollary 2. *The commutativity problem for deterministic RA is* PSPACE-*complete.*

VI. THE REACHABILITY PROBLEM

The reachability problem is defined as follows: *Given an RA$_{\mathbb{Q}}$ $\mathcal{A}$, decide if there is a word w such that* $0 \in \mathcal{A}(w)$. The reachability problem is tightly related to the *may-constancy* problem for programs [28], which asks, for a given program location ℓ, a given variable z, and a given constant c, whether the value of z in ℓ may be equal to c, that is, there is an execution path leading to ℓ such that the value of z in ℓ is c.

Theorem 7. *The reachability problem for RA$_{\mathbb{Q}}$ is undecidable, even for deterministic RA$_{\mathbb{Q}}$.*

The proof of Theorem 7 is obtained by a reduction from PCP [30]. On the other hand, we show that for copyless $\mathrm{RA}_{\mathbb{Q}}$ with non-strict guards, the reachability problem is decidable.

Let X be a set of control variables. A transition guard $\varphi(\vec{x}, \mathsf{cur})$ is *non-strict* if it does not contain negations, i.e., it is a positive Boolean combination of inequalities $z \leqslant z'$ for $z, z' \in X \cup \{\mathsf{cur}\}$. An $\mathrm{RA}_{\mathbb{Q}}$ $\mathcal{A}$ is said to have *non-strict transition guards* if the guard in each transition of $\mathcal{A}$ is non-strict.

Theorem 8. *The reachability problem for (nondeterministic) copyless $\mathrm{RA}_{\mathbb{Q}}$ with non-strict transition guards is in* NEXPTIME.

The rest of this section is devoted to the proof of Theorem 8. Suppose $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta\rangle$ is a copyless $\mathrm{RA}_{\mathbb{Q}}$ with non-strict transition guards over (X, Y), where $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_l\}$. Let $\mathcal{N}$ be the set of constants appearing in $\vec{u}_{0(X)}$. For simplicity, we assume that all control variables initially contain different values.

Suppose there is a word $w = d_1 \cdots d_n$ that leads to a zero output. Let $(q_0, \vec{u}_0) \vdash_{t_1,d_1} (q_1, \vec{u}_1) \vdash_{t_2,d_2} \cdots \vdash_{t_n,d_n} (q_n, \vec{u}_n)$ be the run of $\mathcal{A}$ on w. By Proposition 1, there are M and $\vec{b}$ such that

$$\vec{u}_n \quad = \quad M \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} + \vec{b}.$$

The values $d_1, \ldots, d_n$ satisfy a set of inequalities imposed by the transitions $t_1, \ldots, t_n$. Let $\Phi(\vec{z})$ denote the conjunction of those inequalities, where $\vec{z} = (z_1, \ldots, z_n)^t$ are variables representing the data values $d_1, \ldots, d_n$. For simplicity, we assume that the guards in $t_1, \ldots, t_n$ contain *no disjunctions*, which means that the set of points (vectors) satisfying $\Phi(\vec{z})$ is a convex polyhedron.

Suppose the output function of q_n is $\zeta(q_n) = \vec{a} \cdot \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} + a'$. We define the following function:

$$f(\vec{z}) \quad = \quad \vec{a} \cdot M \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} + \vec{a} \cdot \vec{b} + a'.$$

Thus, by our assumption that $d_1 \cdots d_n$ leads to zero, we have:

$$f((d_1, \ldots, d_n)^t) = 0 \quad \wedge \quad \Phi((d_1, \ldots, d_n)^t) = \mathsf{true},$$

which is equivalent to:

$$\exists \vec{z}_1, \vec{z}_2 \in \mathbb{Q}^n : f(\vec{z}_1) \leqslant 0 \leqslant f(\vec{z}_2) \wedge \Phi(\vec{z}_1) \wedge \Phi(\vec{z}_2). \qquad (2)$$

Observe that (2) holds iff the following two constraints hold simultaneously:

[F1] the infimum of $f(\vec{z})$ w.r.t. $\Phi(\vec{z})$ is $\leqslant 0$,
[F2] the supremum of $f(\vec{z})$ w.r.t. $\Phi(\vec{z})$ is $\geqslant 0$.

From the Simplex algorithm for linear programming [31], we know that the points that yield the optimum, i.e., the infimum and the supremum, are at the "corner" points of convex polyhedra. The constraints in $\Phi(\vec{z})$ only contain the constants from $\mathcal{N}$ (as a result of the fact that the initial contents of control variables are a fixed vector of constants), so the corner points of the convex polyhedron of $\Phi(\vec{z})$ only take values from the set $\mathcal{N} \cup \{-\infty, +\infty\}$.

To establish constraints **F1** and **F2**, it is sufficient to find two corner points $\vec{z}_1$ and $\vec{z}_2$ such that $f(\vec{z}_1) \leqslant 0 \leqslant f(\vec{z}_2)$. To find these two points, we will construct a corresponding $\mathbb{Q}$-VASS (rational vector addition systems with states), where the configuration reachability can be decided in NP.

In the following, we shows how to construct the $\mathbb{Q}$-VASS from $\mathcal{A}$. For simplicity of presentation, we make the following assumptions:

- $\mathcal{A}$ is *order-preserving* on X. That is, at all times the contents of control variables must satisfy the constraint
$$x_1 \;\leqslant\; x_2 \;\leqslant\; \cdots \;\leqslant\; x_k.$$
- The reassignments of data variables are of the form $y_j := y_j + f(\vec{x}, \mathsf{cur})$ for each $y_j \in Y$.

The construction can be generalized to arbitrary copyless $\mathrm{RA}_{\mathbb{Q}}$ with non-strict guards without the two assumptions.

Moreover, we can "split" each transition of $\mathcal{A}$ into several ones by pinpointing the place of cur w.r.t. the linear order $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_k$, so that the guard in each transition is of the form: $\mathsf{cur} = x_i$, $\mathsf{cur} \leqslant x_1$, $x_i \leqslant \mathsf{cur} \leqslant x_{i+1}$, or $x_k \leqslant \mathsf{cur}$.

Let $\vec{u}_{0(X)} = (c_1, \ldots, c_k)^t$. Then $\mathcal{N} = \{c_1, \ldots, c_k\}$ and $c_1 < \cdots < c_k$. Let $\mathcal{N}_\infty = \{-\infty, +\infty\} \cup \mathcal{N}$. A *specification* is a mapping η from X to $\mathcal{N}_\infty$ that respects the ordering of $\mathcal{N}_\infty$, i.e., for $i \leqslant j$, $\eta(x_i) \leqslant \eta(x_j)$. Intuitively, η encodes the value of x_i in a corner point. We have $\eta(x_i) = c_j$ when x_i is either assigned to c_j or to a value arbitrarily close to c_j.

We will construct a $2l$-dimensional $\mathbb{Q}$-VASS (S, Δ) with variables $\vec{y}_1 = (y_{1,1}, \ldots, y_{1,l})$ and $\vec{y}_2 = (y_{2,1}, \ldots, y_{2,l})$ as follows. The set of states S of the $\mathbb{Q}$-VASS is $Q \times \{(\eta_1, \eta_2) \mid \eta_1, \eta_2 \text{ are specifications}\}$. A *configuration* is of the form $((q, \eta_1, \eta_2), \vec{y}_1, \vec{y}_2)$, where $(\eta_1, \vec{y}_1)$ and $(\eta_2, \vec{y}_2)$ summarize the information of the components of the two corner points that have been acquired so far (in other words, the input data values that have been read by the $\mathrm{RA}_{\mathbb{Q}}$ so far). The details of the transition relation Δ can be found in [27].

Consider the initial configuration $((q_0, \eta, \eta), \vec{u}_{0(Y)}, \vec{u}_{0(Y)})$, where $\eta(x_i) = \vec{u}_0(x_i)$ for each $x_i \in X$. It holds that there is w such that $0 \in \mathcal{A}(w)$ iff there is a configuration $((q', \eta_1, \eta_2), \vec{v}_1, \vec{v}_2)$ reachable from the initial configuration such that $q' \in F$ and one of the following holds.

$$\zeta(q')(\eta_1(\vec{x}), \vec{v}_1) \;\leqslant\; 0 \;\leqslant\; \zeta(q')(\eta_2(\vec{x}), \vec{v}_2)$$

or

$$\zeta(q')(\eta_2(\vec{x}), \vec{v}_2) \;\leqslant\; 0 \;\leqslant\; \zeta(q')(\eta_1(\vec{x}), \vec{v}_1).$$

The existence of such a configuration can be encoded as configuration reachability in the constructed $\mathbb{Q}$-VASS, which, in turn, can be reduced to satisfiability of an existential Presburger formula.

VII. RELATED WORK

The literature provides many different formal models with registers or arithmetics. Here we just mention those that are closely related to $\mathrm{RA}_{\mathbb{Q}}$. One of the most general models with registers and arithmetics are *counter automata* [32] (over finite alphabets), which are essentially finite automata equipped with a bounded number of registers capable of holding an integer, which can be tested and updated using Presburger-definable relations. General counter automata with two or more registers are Turing-complete [32], which makes any of their non-trivial problems undecidable.

One way of restricting the expressiveness of counter automata to obtain a decidable model are the so-called *integer vector addition systems with states* ($\mathbb{Z}$-VASS) [33], where testing values of registers is forbidden and the only allowed updates to a register are addition or subtraction of a constant from its value. This restriction makes the configuration reachability problem for $\mathbb{Z}$-VASS much easier (NP-complete) and the *equivalence of reachability sets* problem decidable (coNEXPTIME-complete). For completeness, we also mention *vector addition systems with states* (denoted as VASS without the initial $\mathbb{Z}$), where registers can only hold values from $\mathbb{N}$ (and thus transitions that would decrease the current value below zero are disabled). This makes VASS equivalent to Petri nets. In VASS, configuration reachability is EXPSPACE-hard [34] (but decidable [35]) and equivalence is undecidable [36].

Another way of restricting counter automata to decidable subclasses is via their structure. One important subclass of this kind are the so-called *flat counter automata* [37], i.e., counter automata without nested loops, where configuration reachability and equivalence are decidable.

Register automata (RA) [9]–[11], [15]—sometimes also called *finite-memory automata*— is a model of automata over infinite alphabets where registers can store values copied from the input and transition guards can only test equality between the input value and the values stored in registers. For (nondeterministic) RA, the emptiness problem is PSPACE-complete, while the inclusion, equivalence, and universality problems are all undecidable. Register automata can also be extended [38] to allow transition guards to test the order relation between data values (denoted by $\mathrm{RA}_{\leqslant}$), in which case they are able to simulate *timed automata* [39] by encoding timed words with data words. The model of $\mathrm{RA}_{\mathbb{Q}}$ can be seen as an extension of $\mathrm{RA}_{\leqslant}$ with data variables and linear arithmetics on them. There is also another RA model over the alphabet $\mathbb{N}$ with order and successor relations in guards, but no arithmetic on the input word [40].

As mentioned in the introduction, the model of $\mathrm{RA}_{\mathbb{Q}}$ is inspired by the model of *streaming data string transducers* (SDST), proposed by Alur and Černý in [14]. SDST are an extension of deterministic $\mathrm{RA}_{\leqslant}$ with *data string variables (registers)*, which can hold data strings obtained by concatenating some of the input values that have been read so far. There are two major restrictions imposed on the data strings variables of SDST: (i) they are *write-only*, in the sense that they are forbidden to occur in transition guards, and (ii) the reassignments that update them are *copyless*. These two restrictions are essential for obtaining the PSPACE-completeness result of the equivalence problem for SDST.

Cost register automata (CRA) [41] is a model over finite alphabets where a finite number of *cost registers* are used to store values from a (possibly infinite) cost domain, and these cost registers are updated by using the operations specified by *cost grammars*. A cost domain and a cost grammar, together with its interpretation on the cost domain, are called a *cost model*. An example of a cost model is $(\mathbb{Q}, +)$, where the cost domain is $\mathbb{Q}$, the set of rational numbers, and the cost grammar is the set of linear arithmetic expressions on $\mathbb{Q}$, with $+$ interpreted as the addition operation on $\mathbb{Q}$. Decidability and complexity of decision problems for CRA depend on the underlying *cost model*. For instance, the equivalence problem for CRA over the $(\mathbb{Q}, +)$ cost model is decidable in PTIME, while, on the other hand, for CRA over the $(\mathbb{N}, \min, +c)$ cost model (which are equivalent to weighted automata), the equivalence problem becomes undecidable.

The work related closest to $\mathrm{RA}_{\mathbb{Q}}$ are *streaming numerical transducers* (SNT) introduced in our previous work [3] for investigating the commutativity problem of Reducer programs in the MapReduce framework [24]. The model of SNT is a strict subclass of $\mathrm{RA}_{\mathbb{Q}}$ that satisfies several additional constraints; in particular, SNT are copyless and their transition graph is *deterministic* and *generalized flat* (any two loops share at most one state). In [3], by using a completely different proof strategy than in the current paper, we provided an exponential-time algorithm for the non-zero, equivalence, and commutativity problems of SNT. We did not consider the reachability problem for SNT in [3].

Weighted register automata (WRA) [42] is a model that combines register automata with *weighted automata* [43]. Using the framework of this paper, the model of WRA can be seen as a variant of $\mathrm{RA}_{\mathbb{Q}}$ with exactly one data variable that is used to store the weight, with the following differences: (i) the input data values in WRA can be compared using an arbitrary collection of binary data relations in the data domain, and (ii) the data variable can be updated using an arbitrary collection of binary data functions from the data domain to the weight domain. The work [42] focused on the expressibility issues and did not investigate the decision problems.

Finally, let us mention *symbolic automata* and *symbolic transducers* [44]–[46]. They are extensions of finite automata and transducers where guards in transitions are predicates from an alphabet theory (which is a parameter of the model), thus preserving many of their nice properties. Extending these models with registers in a straightforward way yields undecidable models. Imposing a register access policy (such as that a register always holds the previous value) can bring some decision problems back to the realm of decidability [46], [47]. It is an interesting open problem to find a way of combining symbolic automata with $\mathrm{RA}_{\mathbb{Q}}$.

VIII. Concluding Remarks

In this paper, we defined $\mathrm{RA}_{\mathbb{Q}}$ over the rationals. To the best of our knowledge, it is the first such model over infinite alphabets that allows arithmetic on the input word, while keeping some interesting decision problems decidable. We study some natural decision problems such as the invariant/nonzero problem, which is a generalization of the standard nonemptiness problem, as well as the equivalence, commutativity, and reachability problems. $\mathrm{RA}_{\mathbb{Q}}$ is also quite a general model subsuming at least three well-known models, i.e., the standard RA, affine programs, and arithmetic circuits.

It will be interesting to investigate the configuration reachability and coverability problems for copyless $\mathrm{RA}_{\mathbb{Q}}$. Both of them subsume the corresponding problems for $\mathbb{Z}$- and $\mathbb{Q}$-VASS, since such VASS can be viewed as $\mathrm{RA}_{\mathbb{Q}}$ where data variables represent the counters in the VASS. From Theorem 7, we can already deduce that they are undecidable for general $\mathrm{RA}_{\mathbb{Q}}$. We leave the corresponding problems for copyless $\mathrm{RA}_{\mathbb{Q}}$ as future work.

Acknowledgement: We thank Rajeev Alur for valuable discussions and the anonymous reviewers for their helpful suggestions about how to improve the presentation of the paper. Yu-Fang Chen is supported by the MOST grant No. 103-2221-E-001-019-MY3. Ondřej Lengál is supported by the Czech Science Foundation (project 17-12465S), the BUT FIT project FIT-S-17-4014, and the IT4IXS: IT4Innovations Excellence in Science project (LQ1602). Tony Tan is supported by the MOST grant No. 105-2221-E-002-145-MY2. Zhilin Wu is supported by the NSFC grants No. 61472474, 61572478, 61100062, and 61272135.

REFERENCES

[1] L. Segoufin, "Automata and logics for words and trees over an infinite alphabet," in *Proc. of CSL*, 2006.

[2] M. Bojańczyk, "Automata for data words and data trees," in *Proc. of RTA*, 2010.

[3] Y. Chen, L. Song, and Z. Wu, "The commutativity problem of the MapReduce framework: A transducer-based approach," in *Proc. of CAV*, 2016.

[4] F. Neven, N. Schweikardt, F. Servais, and T. Tan, "Distributed streaming with finite memory," in *Proc. of ICDT*, 2015.

[5] F. Aarts, B. Jonsson, J. Uijen, and F. W. Vaandrager, "Generating models of infinite-state communication protocols using regular inference with abstraction," *FMSD*, vol. 46, no. 1, pp. 1–41, 2015.

[6] N. Tzevelekos, "Fresh-register automata," in *Proc. of POPL*, 2011.

[7] L. Libkin, W. Martens, and D. Vrgoč, "Querying graphs with data," *J. ACM*, vol. 63, no. 2, pp. 14:1–14:53, 2016.

[8] O. Grumberg, O. Kupferman, and S. Sheinvald, "Variable automata over infinite alphabets," in *Proc. of LATA*, 2010.

[9] S. Demri and R. Lazić, "LTL with the freeze quantifier and register automata," *ACM Trans. Comput. Log.*, vol. 10, no. 3, 2009.

[10] Y. Shemesh and N. Francez, "Finite-state unification automata and relational languages," *Inf. Comput.*, vol. 114, no. 2, pp. 192–213, 1994.

[11] M. Kaminski and N. Francez, "Finite-memory automata," *Theor. Comput. Sci.*, vol. 134, no. 2, pp. 329–363, 1994.

[12] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[13] M. L. Minsky, *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

[14] R. Alur and P. Černý, "Streaming transducers for algorithmic verification of single-pass list-processing programs," in *Proc. of POPL*, 2011.

[15] F. Neven, T. Schwentick, and V. Vianu, "Finite state machines for strings over infinite alphabets," *ACM Trans. Comput. Log.*, vol. 5, no. 3, pp. 403–435, 2004.

[16] M. Karr, "Affine relationships among variables of a program," *Acta Inf.*, vol. 6, pp. 133–151, 1976.

[17] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proc. of POPL*, 1977.

[18] B. Jeannet and A. Miné, "APRON: A library of numerical abstract domains for static analysis," in *Proc. of CAV*, 2009.

[19] A. Shpilka and A. Yehudayoff, "Arithmetic circuits: A survey of recent results and open questions," *Foundations and Trends in Theoretical Computer Science*, vol. 5, no. 3-4, pp. 207–388, 2010.

[20] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic complexity theory*. Springer, 1997, vol. 315.

[21] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen, "On the complexity of numerical analysis," *SIAM J. Comput.*, vol. 38, no. 5, pp. 1987–2006, 2009.

[22] M. Müller-Olm and H. Seidl, "A note on Karr's algorithm," in *Proc. of ICALP*, 2004.

[23] L. Blum, M. Shub, and S. Smale, "On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines," *Bull. of the Amer. Math. Soc.*, vol. 21, no. 1, pp. 1–46, 1989.

[24] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. of OSDI*, 2004.

[25] Y. Chen, C. Hong, N. Sinha, and B. Wang, "Commutativity of reducers," in *Proc. of TACAS*, 2015.

[26] T. Xiao, J. Zhang, H. Zhou, Z. Guo, S. McDirmid, W. Lin, W. Chen, and L. Zhou, "Nondeterminism in MapReduce considered harmful? an empirical study on non-commutative aggregators in MapReduce programs," in *Proc. of ICSE*, 2014.

[27] Y. Chen, O. Lengál, T. Tan, and Z. Wu, "Register automata with linear arithmetic," `arXiv:1704.03972`, `https://arxiv.org/abs/1704.03972`.

[28] M. Müller-Olm and O. Rüthing, "On the complexity of constant propagation," in *Proc. of ESOP*, 2000.

[29] T. W. Hungerford, *Algebra*, 8th ed. Springer, 2003.

[30] E. L. Post, "A variant of a recursively unsolvable problem," *Bull. of the Amer. Math. Soc.*, vol. 52, pp. 264–268, 1946.

[31] V. Chvátal, *Linear programming*. New York (NY): Freeman, 1983.

[32] M. L. Minsky, "Recursive unsolvability of Post's problem of "Tag" and other topics in the theory of Turing machines," *The Ann. of Math.*, vol. 74, no. 3, pp. 437–455, 1961.

[33] C. Haase and S. Halfon, "Integer vector addition systems with states," in *Proc. of RP*, 2014.

[34] R. J. Lipton, "The reachability problem requires exponential space," Yale University, Tech. Rep. 62, 1976.

[35] E. W. Mayr, "An algorithm for the general Petri net reachability problem," *SIAM J. on Comput.*, vol. 13, no. 3, pp. 441–460, 1984.

[36] M. Hack, "The equality problem for vector addition systems is undecidable," *Theo. Comp. Sci.*, vol. 2, no. 1, pp. 77–95, 1976.

[37] J. Leroux and G. Sutre, "Flat counter automata almost everywhere," in *Proc. of ATVA*, 2005.

[38] D. Figueira, P. Hofman, and S. Lasota, "Relating timed and register automata," *Math. Struct. in Comput. Sci.*, vol. 26, no. 6, pp. 993–1021, 2014.

[39] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[40] B. Brütsch, P. Landwehr, and W. Thomas, "$\mathbb{N}$-memory automata over the alphabet $\mathbb{N}$," in *Proc. of LATA*, 2017.

[41] R. Alur, L. D'Antoni, J. Deshmukh, M. Raghothaman, and Y. Yuan, "Regular functions and cost register automata," in *Proc. of LICS*, 2013.

[42] P. Babari, M. Droste, and V. Perevoshchikov, "Weighted register automata and weighted logic on data words," in *Proc. of ICTAC*, 2016.

[43] M. P. Schützenberger, "On the definition of a family of automata," *Inf. and Cont.*, vol. 4, no. 2–3, pp. 245–270, 1961.

[44] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjørner, "Symbolic finite state transducers: algorithms and applications," in *Proc. of POPL*, 2012.

[45] M. Veanes, "Applications of symbolic finite automata," in *Proc. of CIAA*, 2013.

[46] L. D'Antoni and M. Veanes, "Extended symbolic finite automata and transducers," *FMSD*, vol. 47, no. 1, pp. 93–119, 2015.

[47] C. Czyba, C. Spinrath, and W. Thomas, "Finite automata over infinite alphabets: Two models with transitions for local change," in *Proc. of DLT*, 2015.

[48] K. N. Verma, H. Seidl, and T. Schwentick, "On the complexity of equational Horn clauses," in *Proc. of CADE*, 2005.