

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* ($RA_{\mathbb{Q}}$). It reads a sequence of rational numbers and outputs another rational number. $RA_{\mathbb{Q}}$ 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 $RA_{\mathbb{Q}}$ 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 $RA_{\mathbb{Q}}$ 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 $RA_{\mathbb{Q}}$, 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* ($RA_{\mathbb{Q}}$). Like in standard RA, an $RA_{\mathbb{Q}}$ is equipped with a finite number of variables (registers), each of them is able to store a value.* The $RA_{\mathbb{Q}}$ 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 (\leq) 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 $RA_{\mathbb{Q}}$ 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. $RA_{\mathbb{Q}}$ 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 $RA_{\mathbb{Q}}$ 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 $RA_{\mathbb{Q}}$ 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 $RA_{\mathbb{Q}}$ where control variables as well as values of the input are ignored. Finally, $RA_{\mathbb{Q}}$ 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.

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 appendix.

[†]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 $z \leq z'$.

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]
RA $_{\mathbb{Q}}$	EXPTIME (Thm. 2)	UNDEC (Thm. 5)	UNDEC (Thm. 5)	UNDEC (Thm. 7)
SV-RA $_{\mathbb{Q}}$	EXPTIME (Thm. 2)	UNDEC (Thm. 5)	UNDEC (Thm. 5)	UNDEC (Thm. 7)
CL-RA $_{\mathbb{Q}}$	PSPACE-c (Thm. 4)	UNDEC (Thm. 5)	UNDEC (Thm. 5)	?
deterministic RA $_{\mathbb{Q}}$	EXPTIME (Thm. 2)	EXPTIME (Cor. 1)	EXPTIME (Cor. 1)	UNDEC (Thm. 7)
deterministic CL-RA $_{\mathbb{Q}}$	PSPACE-c (Thm. 4)	PSPACE-c (Cor. 1)	PSPACE-c (Cor. 1)	?
NSTG-CL-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 $\vec{u} \geq \vec{v}$ when $\vec{u}(i) \geq \vec{v}(i)$ 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 $\text{aff}(V)$ 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 $\text{aff}(V) = \mathbb{A}$.

An *affine transformation* $T : \mathbb{Q}^k \rightarrow \mathbb{Q}^l$ 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 $\vec{x} \mapsto T_2 T_1 \vec{x}$, where $\vec{x} \mapsto T_2 T_1 \vec{x}$ denotes a function that maps \vec{x} to $T_2 T_1 \vec{x}$.

The following two lemmas will be useful.

Lemma 1. *Let $\mathbb{A} \subseteq \mathbb{Q}^k$ be an affine space and $T : \mathbb{Q}^{k+1} \rightarrow \mathbb{Q}^k$ 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 $m \geq \dim(\mathbb{H}) + 2$. Then, there is a set of indices $J = \{j_1, \dots, j_n\}$ such that $1 \leq j_1 < j_2 < \dots < j_n \leq m$, $|J| \leq \dim(\mathbb{H}) + 1$, 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 $T : \mathbb{Q}^n \rightarrow \mathbb{Q}^n$ 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 $\text{aff}(V) = \mathbb{A}$. 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 $\vec{v} \in \text{aff}(V_s)$. 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 $T\vec{v} \notin \text{aff}(V_{s_2})$. It holds that there is a path from (s_0, \vec{u}) to (s', \vec{v}) , for some $\vec{v} \notin \mathbb{H}$, iff $V_{s'} \not\subseteq \mathbb{H}$. 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 ($\text{RA}_{\mathbb{Q}}$)

In the following, we fix $X = \{x_1, \dots, x_k\}$ and $Y = \{y_1, \dots, y_l\}$, 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 $\text{cur} \notin X \cup Y$ 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 $X \cup \{\text{cur}\}$ is a Boolean combination of atomic formulas of the form $z \leq z'$, where $z, z' \in X \cup \{\text{cur}\}$. We write $\mathcal{C}(X, \text{cur})$ to denote the set of all linear constraints over $X \cup \{\text{cur}\}$. For convenience, we use $z < z'$ as an abbreviation of $\neg(z' \leq z)$. 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 $\{x_1, \dots, x_k\}$ to $\{x_1, \dots, x_k, \text{cur}\}$.

Definition 1. *A register automaton over the rationals ($\text{RA}_{\mathbb{Q}}$) 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

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

where $p, q \in Q$ are states, $\varphi(\vec{x}, \text{cur})$ is a linear constraint from $\mathcal{C}(\vec{x}, \text{cur})$, 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 $\varphi(\vec{x}, \text{cur})$ 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 $X \cup \{\text{cur}\}$, and (ii) a reassignment of each data variable in Y with a linear combination of variables in $X \cup Y \cup \{\text{cur}\}$ and constants. We will therefore sometimes write the matrices A, B , and the vector \vec{b} as reassignments to variables of the form $\{x_1 := r_1; \dots; x_k := r_k; y_1 := s_1; \dots; y_l := s_l\}$ or $\{\vec{x} := f(\vec{x}, \text{cur}); \vec{y} := g(\vec{x}, \vec{y}, \text{cur})\}$. 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 $\text{RA}_{\mathbb{Q}}$, 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 $t = (p, \varphi(\vec{x}, \text{cur})) \rightarrow (q, A, B, \vec{b})$ 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 $P = t_1 \dots t_n$, we write $(q_0, \vec{u}_0) \vdash_P (q_n, \vec{u}_n)$ if there is a word $d_1 \dots d_n$ such that $(q_0, \vec{u}_0) \vdash_{t_1, d_1} (q_1, \vec{u}_1) \vdash_{t_2, d_2} \dots \vdash_{t_n, d_n} (q_n, \vec{u}_n)$. In this case, we say that $d_1 \dots d_n$ is *compatible* with $t_1 \dots t_n$. 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 $t_1, \dots, t_n \in \delta$. 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 $\mathcal{A}(w) = \emptyset$).

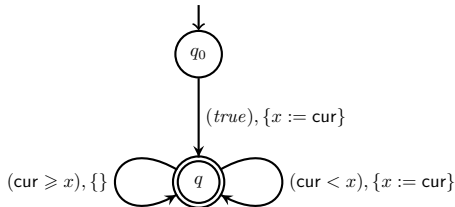
We say that \mathcal{A} is *deterministic* if, for any state p and a pair of transitions $(p, \varphi(\vec{x}, \text{cur})) \rightarrow (q, A, B, \vec{b})$ and $(p, \varphi'(\vec{x}, \text{cur})) \rightarrow (q', A', B', \vec{b}')$ starting in p , the formula $\varphi(\vec{x}, \text{cur}) \wedge \varphi'(\vec{x}, \text{cur})$ 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 , $|\mathcal{A}(w)| \leq 1$. Note that different input words may yield different outputs. Evidently, every deterministic $\text{RA}_{\mathbb{Q}}$ is single-valued.

We call \mathcal{A} *copyless* if the reassignment of its data variables is of the form $\vec{y} := A\vec{y} + f(\vec{x}, \text{cur})$, 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 $\text{RA}_{\mathbb{Q}}$ 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 $\text{RA}_{\mathbb{Q}}$. Also note that affine programs in the sense of Karr [16], [22] are also a special case of $\text{RA}_{\mathbb{Q}}$ in which control variables and input words are ignored.

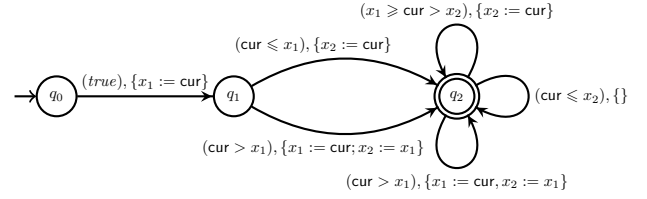
We present some typical aggregate functions that can be computed with $\text{RA}_{\mathbb{Q}}$.

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



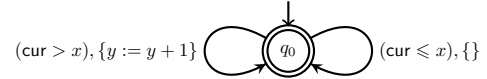
Intuitively, the $\text{RA}_{\mathbb{Q}}$ starts by storing the first value in x . Every subsequent value cur is then compared with x and if $\text{cur} < x$, it is stored in x via the reassignment $x := \text{cur}$.

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



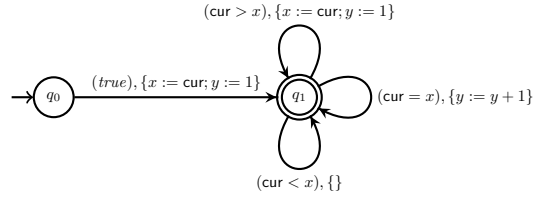
Intuitively, the $\text{RA}_{\mathbb{Q}}$ stores the first two values in x_1 and x_2 in a decreasing order. Each subsequent value cur is compared with x_1 and x_2 , which are updated if necessary.

Computing the number of elements larger than M: The $\text{RA}_{\mathbb{Q}}$ 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 $\text{cur} > x$, the $\text{RA}_{\mathbb{Q}}$ increments y by 1 via the reassignment $y := y + 1$.

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



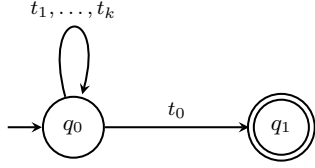
Intuitively, it stores the first value in x and reassigns $y := 1$. Every subsequent value cur is then compared with x . If it is the new largest element, it will be stored in x and the contents of y is reset to 1.

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 $\text{RA}_{\mathbb{Q}}$) *Let \mathcal{A} be an $\text{RA}_{\mathbb{Q}}$ 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*

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

The following example shows that $\text{RA}_{\mathbb{Q}}$ 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 $\text{RA}_{\mathbb{Q}}$ as illustrated below.



\mathcal{A} is over $X = \{x_1, \dots, x_k\}$ 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 $(b_1, \dots, b_k, 1)$, where $b_k \dots b_1$ is the binary representation of n , i.e., $n = \sum_{i=1}^k b_i 2^{i-1}$.

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

$$(q_0, x_i = 1 \wedge \bigwedge_{j=1}^{i-1} x_j = 0) \\ \rightarrow (q_0, \{x_1 := 1; \dots; x_{i-1} := 1; x_i := 0; y := p \cdot y\}).$$

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}, \dots, x_k stay the same. We define the output function $\zeta(q_1)$ to output y .

Intuitively, the contents of variables $\vec{x} = (x_1, \dots, x_k)$ 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, \dots, 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 $\text{RA}_{\mathbb{Q}}$ that always outputs $p_1^{n_1} \dots p_k^{n_k}$ by constructing one $\text{RA}_{\mathbb{Q}}$ for each $p_i^{n_i}$ and composing them sequentially. The final $\text{RA}_{\mathbb{Q}}$ has at most $\sum_{i=1}^k (\lceil \log(n_i) \rceil + 1)$ transitions.

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

Theorem 1. *For every arithmetic circuit C (division-free and without indeterminates), there is an $\text{RA}_{\mathbb{Q}}$ \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 $\text{RA}_{\mathbb{Q}}$

In this section, we will, in the same spirit as Karr [16], consider the *invariant* problem for $\text{RA}_{\mathbb{Q}}$. For an $\text{RA}_{\mathbb{Q}}$ $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ and a state $q \in Q$, define $\text{vec}(\mathcal{A}, q) = \{\vec{v} \mid (q_0, \vec{u}_0) \vdash^* (q, \vec{v})\}$, i.e., $\text{vec}(\mathcal{A}, q)$ 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 $\text{RA}_{\mathbb{Q}}$ \mathcal{A} , a state q of \mathcal{A} , and an affine space \mathbb{H} , decide whether $\text{vec}(\mathcal{A}, q) \not\subseteq \mathbb{H}$.* 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 $\text{aff}(V) = \mathbb{A}$. The invariant problem is tightly related to the *must-constancy* problem for programs [27], 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 $\text{RA}_{\mathbb{Q}}$ \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 $\text{RA}_{\mathbb{Q}}$ \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, \dots, 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 $\text{RA}_{\mathbb{Q}}$ obtained by adding a new state q_f into \mathcal{A} , and for every q_i adding the following transition: $(q_i, \text{true}) \rightarrow (q_f, \{y_1 := \zeta(q_i)\})$. \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 $\text{vec}(\mathcal{A}, q_f) \not\subseteq \mathbb{H}$, 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 $\{\vec{v}_1, \dots, \vec{v}_m\}$ 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 $\text{RA}_{\mathbb{Q}}$ obtained by adding the following into \mathcal{A} :

- $m + 1$ new states q_1, \dots, q_m and p ,
- $m + 1$ new data variables y_1, \dots, y_m and z ,
- for each q_i , the pair of transitions $(q_i, \text{true}) \rightarrow (q_i, \{y_i := (\frac{\vec{x}}{\vec{y}} - \vec{a}) \cdot \vec{v}_i\})$ and $(q_i, \text{true}) \rightarrow (p, \{z := y_i\})$.

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 $i = 1, \dots, m$, thus, $\text{vec}(\mathcal{A}, q) \not\subseteq \mathbb{H}$ 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 $\text{RA}_{\mathbb{Q}}$. Let $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ be the input $\text{RA}_{\mathbb{Q}}$ over (X, Y) , where $X = \{x_1, \dots, x_k\}$ and $Y = \{y_1, \dots, y_l\}$. 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., $\phi = z_1 \otimes_1 z_2 \otimes_2 \dots \otimes_{k-1} z_k$, where (z_1, \dots, z_k) is a permutation of (x_1, \dots, x_k) and each \otimes_i is either $<$ or $=$. An ordering ϕ is *consistent* with a transition $(p, \varphi(\vec{x}, \text{cur})) \rightarrow (q, A, B, \vec{b})$, if $\varphi(\vec{x}, \text{cur}) \wedge \phi$ is satisfiable.

We say that an ordering ϕ holds in a configuration (q, \vec{u}) if it holds when we substitute (x_1, \dots, 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}_A is based on the following lemma.

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

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

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

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

such that $\vec{u}_1 = \vec{v}_1$, $\vec{v}_{m+1} \notin \mathbb{H}$, and for every $i = 1, \dots, m+1$ 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}_A and apply Karr's algorithm on \mathcal{P}_A to decide the non-zero problem. Essentially, the set of states in \mathcal{P}_A is the Cartesian product of Q and the set of orderings of $X \cup \{\text{cur}\}$. The number of variables in \mathcal{P} is $k + l$. There are altogether $2^k(k+1)!$ orderings of $X \cup \{\text{cur}\}$, so the algorithm runs in an exponential time, as stated in Theorem 2 below.

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

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

Theorem 3. (A small model property for $\text{RA}_{\mathbb{Q}}$) *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 $|w'| \leq |Q|(k+l+1)2^k(k+1)!$.*

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 the appendix 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 $\text{RA}_{\mathbb{Q}}$ 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 $\text{RA}_{\mathbb{Q}}$

In the following, let \mathcal{A} be a copyless $\text{RA}_{\mathbb{Q}}$ with k control variables and l data variables. W.l.o.g., we assume that every transition of \mathcal{A} is of the form $(p, \varphi(\vec{x}, \text{cur})) \rightarrow (q, A, B, 0)$, i.e., $\vec{b} = 0$ (every $\text{RA}_{\mathbb{Q}}$ can be transformed to this form by adding new control variables to store the non-zero constants in \vec{b}). Recall that copyless $\text{RA}_{\mathbb{Q}}$ 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*

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

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

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

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

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 $\text{RA}_{\mathbb{Q}}$ (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 $\alpha_1, \dots, \alpha_m, \beta$,

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

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

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

We can assume that the values d'_1, \dots, 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 $\text{RA}_{\mathbb{Q}}$. 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 $\text{RA}_{\mathbb{Q}}$.

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 $f(\vec{x}, \text{cur})$, 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 $\alpha'_1, \dots, \alpha'_k$ instead. Recall that d'_1, \dots, 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 $\text{RA}_{\mathbb{Q}}$ 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 $\text{RA}_{\mathbb{Q}}$ has two control variables x_1 and x_2 storing 0.01 and 0.1 (in binary), respectively, and its transitions have the guard $x_1 < \text{cur} < x_2$ with the reassignment $\{x_1 := \text{cur}\}$. 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) $\text{RA}_{\mathbb{Q}}$. The restriction to copyless $\text{RA}_{\mathbb{Q}}$ allows us to guarantee that the space used by the algorithm is polynomial. If we applied the algorithm to general $\text{RA}_{\mathbb{Q}}$, 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 $\text{RA}_{\mathbb{Q}}$ 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 $\text{RA}_{\mathbb{Q}}$ 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 $\text{RA}_{\mathbb{Q}}$ without data variables is a special case of a copyless $\text{RA}_{\mathbb{Q}}$ but still a generalization of a standard RA.

It is also important that $\text{RA}_{\mathbb{Q}}$ 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 $\text{RA}_{\mathbb{Q}}$ to access only one bit of information from data variables, say, the least significant bit of the integer part of a rational number, $\text{RA}_{\mathbb{Q}}$ 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 $\sqcup^{\omega} 001 \sqcup^{\omega}$ (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 $w' \in \text{perm}(w)$, it holds that $\mathcal{A}(w) = \mathcal{A}(w')$, where $\text{perm}(w)$ 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 $\{\vec{v}_1, \dots, \vec{v}_m\}$ 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 $\pi_1(d_1d_2 \dots d_n) = d_2d_1 \dots d_n$ (swap the first two symbols) and $\pi_2(d_1d_2 \dots d_n) = d_2 \dots d_n d_1$ (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 [28].

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 [27], 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 [29]. On the other hand, we show that for copyless $\text{RA}_{\mathbb{Q}}$ with non-strict guards, the reachability problem is decidable.

Let X be a set of control variables. A transition guard $\varphi(\vec{x}, \text{cur})$ is *non-strict* if it does not contain negations, i.e., it is a positive Boolean combination of inequalities $z \leq z'$ for $z, z' \in X \cup \{\text{cur}\}$. An $\text{RA}_{\mathbb{Q}}$ \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 $\text{RA}_{\mathbb{Q}}$ with non-strict transition guards is in NEXP-TIME.*

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 $\text{RA}_{\mathbb{Q}}$ with non-strict transition guards over (X, Y) , where $X = \{x_1, \dots, x_k\}$ and $Y = \{y_1, \dots, y_l\}$. 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 $w = d_1 \dots d_n$ that leads to a zero output. Let $(q_0, \vec{u}_0) \vdash_{t_1, d_1} (q_1, \vec{u}_1) \vdash_{t_2, d_2} \dots \vdash_{t_n, d_n} (q_n, \vec{u}_n)$ be the run of \mathcal{A} on w . By Proposition 1, there are M and \vec{b} such that

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

The values d_1, \dots, d_n satisfy a set of inequalities imposed by the transitions t_1, \dots, t_n . Let $\Phi(\vec{z})$ denote the conjunction of those inequalities, where $\vec{z} = (z_1, \dots, z_n)^t$ are variables representing the data values d_1, \dots, d_n . For simplicity, we assume that the guards in t_1, \dots, 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:

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

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

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

which is equivalent to:

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

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 ≤ 0 ,

[F2] the supremum of $f(\vec{z})$ w.r.t. $\Phi(\vec{z})$ is ≥ 0 .

From the Simplex algorithm for linear programming [30], 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 $f(\vec{z}_1) \leq 0 \leq f(\vec{z}_2)$. 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 show 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

$$x_1 \leq x_2 \leq \dots \leq x_k.$$

- The reassignments of data variables are of the form $y_j := y_j + f(\vec{x}, \text{cur})$ for each $y_j \in Y$.

The construction can be generalized to arbitrary copyless $\text{RA}_{\mathbb{Q}}$ 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 $x_1 \leq x_2 \leq \dots \leq x_k$, so that the guard in each transition is of the form: $\text{cur} = x_i$, $\text{cur} \leq x_1$, $x_i \leq \text{cur} \leq x_{i+1}$, or $x_k \leq \text{cur}$.

Let $\vec{u}_0(X) = (c_1, \dots, c_k)^t$. Then $\mathcal{N} = \{c_1, \dots, c_k\}$ and $c_1 < \dots < c_k$. Let $\mathcal{N}_{\infty} = \{-\infty, +\infty\} \cup \mathcal{N}$. A *specification* is a mapping η from X to \mathcal{N}_{∞} that respects the ordering of \mathcal{N}_{∞} , i.e., for $i \leq j$, $\eta(x_i) \leq \eta(x_j)$. 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 $\vec{y}_1 = (y_{1,1}, \dots, y_{1,l})$ and $\vec{y}_2 = (y_{2,1}, \dots, y_{2,l})$ 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 (η_1, \vec{y}_1) and (η_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 $\text{RA}_{\mathbb{Q}}$ so far). The details of the transition relation Δ can be found in the appendix.

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.

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

or

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

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 $\text{RA}_{\mathbb{Q}}$. One of the most general models with

registers and arithmetics are *counter automata* [31] (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 [31], 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) [32], 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 [33] (but decidable [34]) and equivalence is undecidable [35].

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* [36], 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 [37] to allow transition guards to test the order relation between data values (denoted by RA_{\leq}), in which case they are able to simulate *timed automata* [38] by encoding timed words with data words. The model of $RA_{\mathbb{Q}}$ can be seen as an extension of RA_{\leq} 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 [39].

As mentioned in the introduction, the model of $RA_{\mathbb{Q}}$ is inspired by the model of *streaming data string transducers* (SDST), proposed by Alur and Černý in [14]. SDST are an extension of deterministic RA_{\leq} 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) [40] 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 $RA_{\mathbb{Q}}$ 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 $RA_{\mathbb{Q}}$ 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) [41] is a model that combines register automata with *weighted automata* [42]. Using the framework of this paper, the model of WRA can be seen as a variant of $RA_{\mathbb{Q}}$ 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 [41] focused on the expressibility issues and did not investigate the decision problems.

Finally, let us mention *symbolic automata* and *symbolic transducers* [43]–[45]. 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 [45], [46]. It is an interesting open problem to find a way of combining symbolic automata with $RA_{\mathbb{Q}}$.

VIII. CONCLUDING REMARKS

In this paper, we defined $RA_{\mathbb{Q}}$ 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/non-zero problem, which is a generalization of the standard non-emptiness problem, as well as the equivalence, commutativity, and reachability problems. $RA_{\mathbb{Q}}$ 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 $RA_{\mathbb{Q}}$. Both of them subsume the corresponding problems for \mathbb{Z} - and \mathbb{Q} -VASS, since such VASS can be viewed as $RA_{\mathbb{Q}}$ where data variables represent the counters in the VASS. From Theorem 7, we can already deduce that they are undecidable for general $RA_{\mathbb{Q}}$. We leave the corresponding problems for copyless $RA_{\mathbb{Q}}$ 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. McDermid, 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] M. Müller-Olm and O. Rüdthig, “On the complexity of constant propagation,” in *Proc. of ESOP*, 2000.
- [28] T. W. Hungerford, *Algebra*, 8th ed. Springer, 2003.
- [29] E. L. Post, “A variant of a recursively unsolvable problem,” *Bull. of the Amer. Math. Soc.*, vol. 52, pp. 264–268, 1946.
- [30] V. Chvátal, *Linear programming*. New York (NY): Freeman, 1983.
- [31] 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.
- [32] C. Haase and S. Halfon, “Integer vector addition systems with states,” in *Proc. of RP*, 2014.
- [33] R. J. Lipton, “The reachability problem requires exponential space,” Yale University, Tech. Rep. 62, 1976.
- [34] E. W. Mayr, “An algorithm for the general Petri net reachability problem,” *SIAM J. on Comput.*, vol. 13, no. 3, pp. 441–460, 1984.
- [35] M. Hack, “The equality problem for vector addition systems is undecidable,” *Theor. Comp. Sci.*, vol. 2, no. 1, pp. 77–95, 1976.
- [36] J. Leroux and G. Sutre, “Flat counter automata almost everywhere,” in *Proc. of ATVA*, 2005.
- [37] 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.
- [38] R. Alur and D. L. Dill, “A theory of timed automata,” *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.
- [39] B. Brütsch, P. Landwehr, and W. Thomas, “ \mathbb{N} -memory automata over the alphabet \mathbb{N} ,” in *Proc. of LATA*, 2017.
- [40] R. Alur, L. D’Antoni, J. Deshmukh, M. Raghothaman, and Y. Yuan, “Regular functions and cost register automata,” in *Proc. of LICS*, 2013.
- [41] P. Babari, M. Droste, and V. Perevoshchikov, “Weighted register automata and weighted logic on data words,” in *Proc. of ICTAC*, 2016.
- [42] M. P. Schützenberger, “On the definition of a family of automata,” *Inf. and Cont.*, vol. 4, no. 2–3, pp. 245–270, 1961.
- [43] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjørner, “Symbolic finite state transducers: algorithms and applications,” in *Proc. of POPL*, 2012.
- [44] M. Veanes, “Applications of symbolic finite automata,” in *Proc. of CIAA*, 2013.
- [45] L. D’Antoni and M. Veanes, “Extended symbolic finite automata and transducers,” *FMSD*, vol. 47, no. 1, pp. 93–119, 2015.
- [46] C. Czyba, C. Spinrath, and W. Thomas, “Finite automata over infinite alphabets: Two models with transitions for local change,” in *Proc. of DLT*, 2015.
- [47] K. N. Verma, H. Seidl, and T. Schwentick, “On the complexity of equational Horn clauses,” in *Proc. of CADE*, 2005.

APPENDIX

A. Proof of Lemma 1

Let $\mathbb{A} = \vec{u} + \mathbb{V}$, where \mathbb{V} is a vector space. Let $T\vec{x} = M\vec{x} + \vec{a}$. Furthermore, let $M = [M_0 \mid \vec{b}]$, i.e., \vec{b} is the last column of matrix M . Then, $T \begin{bmatrix} \vec{v} \\ d_i \end{bmatrix} = M_0\vec{v} + d_i\vec{b} + \vec{a} \in \mathbb{A}$, and hence, $M_0\vec{v} + d_i\vec{b} + \vec{a} - \vec{u} \in \mathbb{V}$, for every $i = 1, 2$. Subtracting one from the other, we get $(d_1 - d_2)\vec{b} \in \mathbb{V}$.

Since $\alpha(d_1 - d_2)\vec{b} \in \mathbb{V}$, for every $\alpha \in \mathbb{Q}$, and $T \begin{bmatrix} \vec{v} \\ d_i \end{bmatrix} \in \mathbb{A}$,

$$T \begin{bmatrix} \vec{v} \\ d_i \end{bmatrix} + \alpha(d_1 - d_2)\vec{b} \in \mathbb{A}, \quad \text{for every } \alpha \in \mathbb{Q}.$$

For every $d \in \mathbb{Q}$, if we take $\alpha = (d - d_1)/(d_1 - d_2)$, we have $T \begin{bmatrix} \vec{v} \\ d \end{bmatrix} = T \begin{bmatrix} \vec{v} \\ d_1 \end{bmatrix} + \alpha(d_1 - d_2)\vec{b}$, which by the equation above, is in \mathbb{A} . This completes our proof of Lemma 1.

B. Proof of Lemma 2

It suffices to show that for $m \geq \dim(\mathbb{H}) + 2$, there is such a set J such that $|J| \leq m - 1$. The proof is by induction on $\dim(\mathbb{H})$. The base case is $\dim(\mathbb{H}) = 0$, i.e., \mathbb{H} consists of only one point, say \vec{c} . We pick the smallest index $j \in \{1, \dots, m+1\}$ such that $\vec{u}_j \neq \vec{c}$. If $j = 1$, we set $J = \emptyset$, and the claim holds trivially. If $j \neq 1$, then $\vec{u}_1 = \dots = \vec{u}_{j-1} = \vec{c}$, we can set $J = \{j-1\}$, where $T_{j-1}\vec{u}_1 = T_{j-1}\vec{c} \notin \mathbb{H}$.

For the induction step, let $m \geq \dim(\mathbb{H}) + 2$ and $\vec{u}_1, \dots, \vec{u}_{m+1}$ be vectors such that $\vec{u}_{i+1} = T_i\vec{u}_i$. If $\vec{u}_m \notin \mathbb{H}$, we can take $J = \{1, \dots, m-1\}$. So, we assume that $\vec{u}_m \in \mathbb{H}$.

Let \mathbb{K} be the pre-image of \mathbb{H} under T_m , i.e., $\mathbb{K} = \{\vec{u} \mid T_m\vec{u} \in \mathbb{H}\}$. Since $T_m(\vec{u}_m) \notin \mathbb{H}$, we have $\vec{u}_m \notin \mathbb{K}$. Thus, $\mathbb{H} \cap \mathbb{K} \subsetneq \mathbb{H}$, and therefore, $\dim(\mathbb{H} \cap \mathbb{K}) \leq \dim(\mathbb{H}) - 1$. Applying the induction hypothesis, we have $J_0 = \{j_1, \dots, j_n\}$ such that $|J_0| \leq m - 2$ and $T_{j_n} \dots T_{j_1}\vec{u}_1 \notin \mathbb{H} \cap \mathbb{K}$.

Let $\vec{z} = T_{j_n} \dots T_{j_1}\vec{u}_1$. If $\vec{z} \notin \mathbb{H}$, the set $J = J_0$ is as desired. Assume that $\vec{z} \in \mathbb{H}$. Since $\vec{z} \notin \mathbb{H} \cap \mathbb{K}$, it should be that $T_m\vec{z} \notin \mathbb{H}$. So, the set $J = J_0 \cup \{m\}$ is as desired. This completes our proof of Lemma 2.

C. Proof for Remark 1

Let $\mathcal{P} = (S, s_0, \mu)$ be an AP. Let the path from (s_0, \vec{u}) to (s', \vec{v}) , for some $\vec{v} \notin \mathbb{H}$, as follows.

$$(s_0, T_1, s_1), (s_1, T_2, s_2), \dots, (s_{\ell-1}, T_\ell, s_\ell)$$

If $\ell > (n+1)|S|$, there is a state that appears at least $n+2$ times in the path. Let us denote by p such state.

Let T'_0 be the composition of the transformations from s_0 to the first appearance of p , and T'_1 the composition of the transformations from the first appearance of p to the second, and T'_2 the composition of the transformations from the second appearance of p to the third, and so on. Let $m \geq n+2$ be the number of appearances of p in the path. So, we have:

$$RT'_{m-1} \dots T'_1 T'_0 \vec{u} = \vec{v} \notin \mathbb{H}$$

where R is the composition of the transformations from the last appearance of p to s' .

Let \mathbb{K} be the pre-image of \mathbb{H} under R . So,

$$T'_{m-1} \dots T'_1 T'_0 \vec{u} \notin \mathbb{K}$$

Now, $m-1 \geq n+1$. On the other hand, $\dim(\mathbb{K}) \leq n-1$, as otherwise, \mathbb{K} is the whole space \mathbb{Q}^n . So, $m-1 \geq \dim(\mathbb{K}) + 2$. By Lemma 2, there is $J = \{j_1, \dots, j_n\} \subseteq \{1, \dots, m-1\}$ such that $|J| \leq \dim(\mathbb{K}) + 1$ and

$$T'_{j_n} \dots T'_{j_1} T'_0 \vec{u} \notin \mathbb{K}$$

and thus, $RT'_{j_n} \dots T'_{j_1} T'_0 \vec{u} \notin \mathbb{H}$. The path for such composition of transformations contains the state p only for at most $\dim(\mathbb{K}) + 2$ number of times. So, we have the bound that the path contains the state p at most $n+1$ times.

Note that if all the transformations in \mathcal{P} are one-to-one, R is also one-to-one, and thus, $\dim(\mathbb{K}) = \dim(\mathbb{H})$. So, the path contains p at most $\dim(\mathbb{H}) + 2$ times.

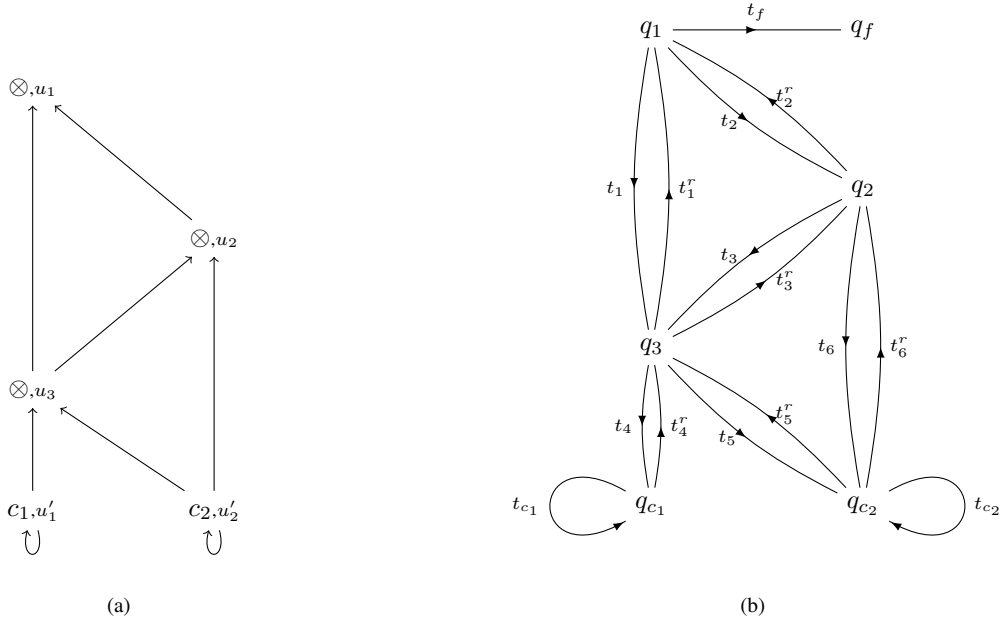


Fig. 1. An AC on the left and the constructed $RA_{\mathbb{Q}}$ on the right. The loops on the constant nodes in AC are imaginary loops whose sole purpose is to make the nodes “uniform” in that all of them have incoming edges, thus, avoid cumbersome case analysis in our construction.

D. A brief review of arithmetic circuits and proof of Theorem 1

Briefly, a (division-free) arithmetic circuit (AC) is a directed acyclic graph with nodes labeled with constants from \mathbb{Z} , or with some indeterminates X_1, \dots, X_m , or with the operators $+, -, \times$. The nodes labeled with constants are called constant nodes, while those labeled with indeterminates are called input nodes. Both constant and input nodes don’t have incoming edges. Internal nodes are those labeled with $+, -, \times$. Output node is one without out-going edges.

We assume that all the operators $+, -, \times$ are binary, so all the internal nodes have in-degree 2. We also assume that there is only one output node. Each node u in a circuit represents a multivariate polynomial $\mathbb{Z}[X_1, \dots, X_m]$, denoted by val_u . Here we are only interested in arithmetic circuits without input nodes, thus, val_u is an integer.

In the following, \oplus -nodes and \otimes -nodes refer to nodes with label $+$ and \times , respectively. Since $-$ can be rewritten with $+$ and multiplication with -1 , we assume our AC contains only internal \oplus - and \otimes -nodes. A circuit is called *multiplicative*, if all the internal nodes are \otimes -nodes. Likewise, it is additive, if all the internal nodes are \oplus -nodes.

The rest of this section is devoted to the proof of Theorem 1. We will first describe the proof for multiplicative/additive circuits. The general case will follow after that.

We need some terminologies. For an edge $e = (u, v)$, we say that u is the *source node* of e , while v is the *target node* of e . A node u is *below* v in a circuit C , if either $u = v$ or there is a path from u to v in C . An edge e is below v , if the target node of e is below v . Intuitively, if u is below v , the value of v depends on the value of u . Likewise, if an edge e is below v , the value of v depends on the value of both the source and the target nodes of v .

To avoid cumbersome case-by-case analysis, we assume that there is an “imaginary” loop on each constant node in the AC, as it will help reduce the complication in our proof. With such loops, all nodes in AC have incoming edges.

Multiplicative/additive circuits: We will only describe our proof for multiplicative circuits. The additive case can be treated in a similar manner.

Before we present the formal detail, we would like to explain our proof via the example in Figure 1. Figure 1(a) shows an AC, where \otimes, u_i indicate the node u_i with label \times . Figure 1(b) shows the topology of the constructed $RA_{\mathbb{Q}}$ with initial state q_1 corresponding to the output node u_1 , final state q_f and four states q_2, q_3 and q_{c_1}, q_{c_2} corresponding to the internal nodes u_2, u_3 and the constant nodes c_1, c_2 in AC, respectively.

Intuitively, the $RA_{\mathbb{Q}}$ has one data variable y with initial content 1. The output function in the final state q_f will output the value in data variable y . To reach q_f , the $RA_{\mathbb{Q}}$ will be “forced” to traverse along the following path:

$$q_1, q_3, q_{c_1}, q_3, q_{c_2}, q_3, q_1, q_2, q_3, q_{c_1}, q_3, q_{c_2}, q_3, q_2, q_{c_2}, q_2, q_1, q_f$$

When it reaches q_{c_1} , it will take transition t_{c_1} in which there is a reassignment $y := c_1 \cdot y$, before it can leave q_{c_1} . Likewise, when it reaches q_{c_2} , it will take transition t_{c_2} with reassignment $y := c_2 \cdot y$. In the transitions t_1, \dots, t_6, t_f the reassignment

is $y := y$ (i.e. the value of y is unchanged). Note that in the path above, the $\text{RA}_{\mathbb{Q}}$ enters q_3 once from q_1 and once from q_2 . Each time it enters q_3 , it has to enter both q_{c_1} and q_{c_2} to ensure y is multiplied with c_1 and c_2 .

To make sure that the $\text{RA}_{\mathbb{Q}}$ traverses through all its transitions correctly, we will employ control variables x_e and z_e for the forward and reverse direction of each edge e , respectively, that stores 0-1 value to remember which edge yet to traverse and which edge to backtrack to. The 0 value of a variable x_e or z_e means the corresponding transition can be taken and 1 otherwise. The formal construction is as follows. Let $C = (V, E)$ be an AC where all the internal nodes are \otimes -nodes, and E is the set of edges which includes the loops on the constant nodes.

Let $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ be the following $\text{RA}_{\mathbb{Q}}$.

- There is one data variable y with initial content 1.
- For each edge e in E , there are two control variables x_e and z_e with initial content 0.
- The set Q of states is $\{q_f\} \cup \{q_u \mid u \in V\}$, where q_u is a state corresponding to the node u in V .
- The initial state is q_u , where u is the output node in C , and there is only final state q_f .
- The output function in q_f outputs the value y .

The transitions in δ are defined as follows.

In the following we consider a loop e on a node u as both its incoming and outgoing edge. For each internal node u , we fix one of its incoming edge as its *left-edge* and the other one as its *right-edge*.

[T1] Let u be the output node in C and e_1, e_2 be its left- and right-edges, respectively. Then, δ contains the transition:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1) \rightarrow (q_f, \{\})$$

Intuitively, the transition means that when \mathcal{A} is in state q_u and it has traversed both the edges e_1 and e_2 (indicated by them containing value 1), then it can go to the final state q_f .

[T2] For each internal node u in C , the transitions from the state q_u are defined as follows.

Let e_1, e_2 be left- and right-edges of u , respectively, and let v_1, v_2 be their source nodes.

The following transitions are in δ :

$$\begin{aligned} (q_u, x_{e_1} = 0) &\rightarrow (q_{v_1}, \{z_{e_1} := 1\} \cup \{x_e := 0 \mid e \text{ is an incoming edge of } v_1\}) \\ (q_u, x_{e_1} = 1 \wedge x_{e_2} = 0) &\rightarrow (q_{v_2}, \{z_{e_2} := 1\} \cup \{x_e := 0 \mid e \text{ is an incoming edge of } v_2\}) \end{aligned}$$

Note that if v_i is a constant node, then there is only one incoming edge e to v_i , which is the loop on v_i , and x_e is assigned 0, as well.

Let e'_1, \dots, e'_m be the outgoing edges of u , and let v'_1, \dots, v'_m be their target nodes, respectively.

For each $i = 1, 2, \dots, m$, the following transition is in δ :

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1\})$$

[T3] For each constant node u in C , the transition from state q_u is defined as follows.

Let p be the integer label of u .

Let e be the loop on u . The following transition is in δ :

$$(q_u, x_e = 0) \rightarrow (q_u, \{x_e := 1; y := p \cdot y\})$$

Let e'_1, \dots, e'_m be the outgoing edges of u , and let v'_1, \dots, v'_m be their target nodes, respectively.

For each $i = 1, 2, \dots, m$, the following transition is in δ :

$$(q_u, x_e = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1\})$$

The following claim immediately implies the correctness of our $\text{RA}_{\mathbb{Q}}$. Intuitively, it states that for each node u in the circuit C , if \mathcal{A} “enters” q_u with the initial content of y being, say N , then \mathcal{A} “exits” q_u with the content of y being $\text{val}_u \cdot N$.

Claim 1. Let u be a node in the circuit C . Let \vec{v}_1 be the contents of the variables such that $\vec{v}_1(x_e) = 0$, for every incoming edge e of u . Then,

$$(q_u, \vec{v}_1) \vdash^* (q_u, \vec{v}_2)$$

for some \vec{v}_2 such that

- $\vec{v}_2(y) = \text{val}_u \times \vec{v}_1(y)$;
- $\vec{v}_2(x_e) = \begin{cases} 1, & \text{for every edge } e \text{ below } u \\ \vec{v}_1(x_e), & \text{for other } e. \end{cases}$

We omit the proof which is by straightforward induction on the distance between u and its furthest constant node.

Proof of Theorem 1: Let C be an AC. The construction of \mathcal{A} follows similar idea as above by evaluating val_u , for each node u in C . When u is \otimes -node, the evaluation is similar to the above. When u is a \oplus -node, the evaluation is done via the distributive law: $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$.

Let $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ be the following $\text{RA}_{\mathbb{Q}}$.

- \mathcal{A} contains the following data variables.
 - For each \otimes -node u , there is one data variable y_u with initial content 1.
 - For each \oplus -node u , there are three data variables $y_u, y_u^{\text{left}}, y_u^{\text{right}}$ with initial content 1.
 - For each constant node u , there is one data variable y_u with initial content 1.

In fact, as we will see later, the initial contents of data variables are not important, except the one associated with the output node.

Intuitively, the meaning of the data variable for \oplus -node is as follows. Let u be a \oplus -node and e be an arc from u to v , when \mathcal{A} goes from a state q_v to q_u in some transition, y_u will be assigned with the value of y_v , moreover, the value of y_u will keep unchanged before returning to the state q_v . In addition, let v_1, v_2 be the source nodes of the left- and right-edge of u respectively, then after returning to the state q_u from the state q_{v_1} for the first time, $y_u^{\text{left}} = y_u \cdot \text{val}_{v_1}$, similarly, after returning to the state q_u from the state q_{v_2} for the first time, $y_u^{\text{right}} = y_u \cdot \text{val}_{v_2}$.

- For each edge e in E , there are two control variables x_e and z_e with initial content 0.
- The set Q of states is $\{q_f\} \cup \{q_u \mid u \in V\}$, where q_u is a state corresponds to the node u in V .
- The initial state is q_u , where u is the output node in C , and there is only final state q_f .
- If the output node u is \otimes -node, the function $\zeta(q_f)$ outputs y_u .
- If the output node u is \oplus -node, the function $\zeta(q_f)$ outputs $y_u^{\text{left}} + y_u^{\text{right}}$.

The transitions in δ are defined as follows.

[T1] Let u be the output node in C and e_1, e_2 be its left- and right-edges, respectively. Then, δ contains the transition:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1) \rightarrow (q_f, \{\})$$

[T2] For each internal \otimes -node u in C , the transitions from state q_u are defined as follows.

Let e_1, e_2 be left- and right-edges of u , respectively, and let v_1, v_2 be their source nodes.

The following transitions are in δ :

$$\begin{aligned} (q_u, x_{e_1} = 0) &\rightarrow (q_{v_1}, \{z_{e_1} := 1; y_{v_1} := y_u\}) \cup \{x_e := 0 \mid e \text{ is an incoming edge of } v_1\}) \\ (q_u, x_{e_1} = 1 \wedge x_{e_2} = 0) &\rightarrow (q_{v_2}, \{z_{e_2} := 1; y_{v_2} := y_u\}) \cup \{x_e := 0 \mid e \text{ is an incoming edge of } v_2\}) \end{aligned}$$

Let e'_1, \dots, e'_m be the outgoing edges of u , and let v'_1, \dots, v'_m be their target nodes, respectively.

For each $i = 1, 2, \dots, m$, the following transition is in δ :

- If v'_i is \otimes -node:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1; y_{v'_i} := y_u\})$$

- If v'_i is \oplus -node and e'_i is its left-edge:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1; y_{v'_i}^{\text{left}} := y_u\})$$

- If v'_i is \oplus -node and e'_i is its right-edge:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1; y_{v'_i}^{\text{right}} := y_u\})$$

[T3] For each internal \oplus -node u in C , the transitions from state q_u are defined as follows.

Let e_1, e_2 be left- and right-edges of u , respectively, and let v_1, v_2 be their source nodes.

The following transitions are in δ :

$$\begin{aligned} (q_u, x_{e_1} = 0) &\rightarrow (q_{v_1}, \{z_{e_1} := 1; y_{v_1} := y_u\}) \cup \{x_e := 0 \mid e \text{ is an incoming edge of } v_1\}) \\ (q_u, x_{e_1} = 1 \wedge x_{e_2} = 0) &\rightarrow (q_{v_2}, \{z_{e_2} := 1; y_{v_2} := y_u\}) \cup \{x_e := 0 \mid e \text{ is an incoming edge of } v_2\}) \end{aligned}$$

Let e'_1, \dots, e'_m be the outgoing edges of u , and let v'_1, \dots, v'_m be their target nodes, respectively.

For each $i = 1, 2, \dots, m$, the following transition is in δ :

- If v'_i is \otimes -node:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1; y_{v'_i} := y_u^{\text{left}} + y_u^{\text{right}}\})$$

- If v'_i is \oplus -node and e'_i is its left-edge:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1; y_{v'_i}^{\text{left}} := y_u^{\text{left}} + y_u^{\text{right}}\})$$

- If v'_i is \oplus -node and e'_i is its right-edge:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1; y_{v'_i}^{\text{right}} := y_u^{\text{left}} + y_u^{\text{right}}\})$$

[T4] For each constant node u in C , the transition from state q_u is defined as follows.

Let p be the integer label of u and e be the loop on u . The following transition is in δ :

$$(q_u, x_e = 0) \rightarrow (q_u, \{x_e := 1; y_u := p \cdot y_u\})$$

Let e'_1, \dots, e'_m be the outgoing edges of u , and let v'_1, \dots, v'_m be their target nodes, respectively.

For each $i = 1, 2, \dots, m$, the following transition is in δ :

- If v'_i is \otimes -node:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1; y_{v'_i} := y_u\})$$

- If v'_i is \oplus -node and e'_i is its left-edge:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1; y_{v'_i}^{\text{left}} := y_u\})$$

- If v'_i is \oplus -node and e'_i is its right-edge:

$$(q_u, x_{e_1} = 1 \wedge x_{e_2} = 1 \wedge z_{e'_i} = 1) \rightarrow (q_{v'_i}, \{z_{e'_i} := 0; x_{e'_i} := 1; y_{v'_i}^{\text{right}} := y_u\})$$

The following claim immediately implies the correctness of our construction. The proof is by straightforward induction on the distance between u and its furthest constant node.

Claim 2. *Let u be a node in the circuit C . Let \vec{v}_1 be the content of the variables of \mathcal{A} such that $\vec{v}_1(x_e) = 0$, for every incoming edge e of u . Then,*

$$(q_u, \vec{v}_1) \vdash^* (q_u, \vec{v}_2)$$

for some \vec{v}_2 such that the following holds.

- u is \otimes -node:
 - $\vec{v}_2(y_u) = \mathbf{val}_u \cdot \vec{v}_1(y_u)$;
 - $\vec{v}_2(x_e) = \begin{cases} 1, & \text{for every edge } e \text{ below } u \\ \vec{v}_1(x_e), & \text{for other } e. \end{cases}$
- u is \oplus -node:
 - $\vec{v}_2(y_u^{\text{left}}) + \vec{v}_2(y_u^{\text{right}}) = \mathbf{val}_u \cdot \vec{v}_1(y)$;
 - $\vec{v}_2(x_e) = \begin{cases} 1, & \text{for every edge } e \text{ below } u \\ \vec{v}_1(x_e), & \text{for other } e. \end{cases}$

E. Proof of Lemma 3

We will need the following terminologies. We say that cur is *bound* in a constraint/ordering ϕ , if ϕ implies $\text{cur} = x_i$, for some x_i , in which case, we say that cur is bound to x_i . Otherwise, we say that cur is *free*. For example, in $(x_2 < \text{cur} = x_1)$ and $(\text{cur} = x_2 < x_1)$, cur is bound to x_1 and x_2 , respectively, while in $(\text{cur} < x_2 < x_1)$ and $(x_1 < \text{cur} < x_2)$, cur is free.

The proof of Lemma 3 is by induction on m . The base case $m = 0$ is trivial. For the induction step, suppose the following.

$$(q_1, \vec{u}_1) \vdash_{t_1, d_1} \dots \vdash_{t_m, d_m} (q_{m+1}, \vec{u}_{m+1}) \quad \text{and} \quad \vec{u}_{m+1} \notin \mathbb{H}.$$

Let t_m be $(p, \varphi(\vec{x}, \text{cur})) \rightarrow (q, A, B, \vec{b})$. It induces a mapping, denoted by the same symbol t_m , that maps an affine space \mathbb{A} to another as follows.

- If cur is bound in φ to x_i in \vec{x} , then

$$t(\mathbb{A}) = \left\{ \vec{v} \mid \vec{v}(X) = A \begin{bmatrix} \vec{u}(X) \\ \vec{u}(i) \end{bmatrix}, \vec{v}(Y) = B \begin{bmatrix} \vec{u} \\ \vec{u}(i) \end{bmatrix} + \vec{b} \text{ where } \vec{u} \in \mathbb{A} \right\}$$

- If cur is free in φ , then

$$t(\mathbb{A}) = \left\{ \vec{v} \mid \vec{v}(X) = A \begin{bmatrix} \vec{u}(X) \\ \alpha \end{bmatrix}, \vec{v}(Y) = B \begin{bmatrix} \vec{u} \\ \alpha \end{bmatrix} + \vec{b} \text{ where } \vec{u} \in \mathbb{A} \text{ and } \alpha \in \mathbb{Q} \right\}$$

Let \mathbb{K} be the pre-image of \mathbb{H} under t_m . Thus, $\vec{u}_m \notin \mathbb{K}$. For the the induction hypothesis, we assume that there is $c_1 \cdots c_{m-1}$ such that

$$(q_1, \vec{v}_1) \vdash_{t_1, c_1} \cdots \vdash_{t_m, c_{m-1}} (q_m, \vec{v}_m) \quad \text{and} \quad \vec{v}_m \notin \mathbb{K},$$

where all the conditions (a)–(e) holds for each $i = 1, \dots, m$. In particular, (q_m, \vec{u}_m) and (q_m, \vec{v}_m) have the same ordering.

We will show that there is c_m such that $(q_m, \vec{v}_m) \vdash_{t_m, c_m} (q_{m+1}, \vec{v}_{m+1})$ such that all the conditions (a)–(e) holds. There are a few cases.

- Case 1: $d_m = \vec{u}_m(X)(j)$ for some $1 \leq j \leq k$.

We fix $c_m = \vec{v}_m(X)(j)$. Since \vec{u}_m and \vec{v}_m have the same ordering, $\varphi(\vec{v}_m, c_m)$ holds too. Thus, let \vec{v}_{m+1} be such that $(q_m, \vec{v}_m) \vdash_{t_m, c_m} (q_{m+1}, \vec{v}_{m+1})$. Since they are obtained by taking the same transition from configurations with the same ordering, (q_{m+1}, \vec{v}_{m+1}) and (q_{m+1}, \vec{u}_{m+1}) have the same ordering, as well.

- Case 2: $d_m < \vec{u}_m(X)(j)$, where $\vec{u}_m(X)(j) = \min(\vec{u}_m(X))$, where $\min(\vec{u}_m(X))$ is the minimum component of $\vec{u}_m(X)$. Since (q_m, \vec{u}_m) and (q_m, \vec{v}_m) have the same ordering, $\varphi(\vec{v}_m, c)$ holds for either $c = \vec{v}_m(X)(j) - 1$ or $c = \vec{v}_m(X)(j) - 2$. Now, $\vec{v}_m \notin \mathbb{K}$. Hence, $t_m(\{\vec{v}_m\}) \not\subseteq \mathbb{H}$. By Lemma 1, for at least one of $c = \vec{v}_m(j) - 1$ or $c = \vec{v}_m(j) - 2$,

$$\vec{v}_{m+1} \notin \mathbb{H},$$

where $\vec{v}_{m+1}(X) = A \begin{bmatrix} \vec{u}(X) \\ c \end{bmatrix}$ and $\vec{v}_{m+1}(Y) = B \begin{bmatrix} \vec{u}(Y) \\ c \end{bmatrix} + \vec{b}$. Let c_{m+1} be such c . Hence, $(q_m, \vec{v}_m) \vdash_{t_m, c_m} (q_{m+1}, \vec{v}_{m+1})$, which also implies (q_{m+1}, \vec{v}_{m+1}) and (q_{m+1}, \vec{u}_{m+1}) having the same ordering.

- The other two cases can be proved in a similar manner as in Case 2.

This completes our proof of Lemma 3.

F. Construction of the affine program and application of Karr's algorithm for Theorem 2

Define the affine program $\mathcal{P} = (S, s_0, \Delta)$ as follows. The variables in \mathcal{P} are $X \cup Y$. The set S consists of states of the form (q, ϕ) , where $q \in Q$ and ϕ is an ordering of $X \cup \{\text{cur}\}$. Let ϕ_0 be the ordering of \vec{u}_0 and $s_0 = (q_0, \phi_0)$. For each transition $t = (p, \varphi(\vec{x}, \text{cur})) \rightarrow (q, A, B, \vec{b})$ in δ , for each ordering ϕ of $X \cup \{\text{cur}\}$ that is consistent with t , we add the following transitions to Δ . Note that since $A \in \mathbb{P}^{k \times (k+1)}$ simply selects k elements from $X \cup \{\text{cur}\}$, we can infer a set of orderings that are consistent with the result of the application of A on ϕ . In the following let ϕ' be an ordering that is consistent with the application of A on ϕ .

- If cur is bound to x_i in ϕ , we add $((p, \phi), T, (q, \phi'))$, where T represents the reassignment:

$$\vec{x} := A \begin{bmatrix} \vec{x} \\ x_i \end{bmatrix} \quad \vec{y} := B \begin{bmatrix} \vec{y} \\ x_i \end{bmatrix}$$

- If $x_i < \text{cur} < x_j$ appears in ϕ , we add $((p, \phi), T_1, (q, \phi'))$ and $((p, \phi), T_2, (q, \phi'))$, where T_1 represents the reassignment:

$$\vec{x} := A \begin{bmatrix} \vec{x} \\ \frac{1}{3}x_i + \frac{2}{3}x_j \end{bmatrix} \quad \vec{y} := B \begin{bmatrix} \vec{y} \\ \frac{1}{3}x_i + \frac{2}{3}x_j \end{bmatrix} + \vec{b}$$

and T_2 represents the reassignment:

$$\vec{x} := A \begin{bmatrix} \vec{x} \\ \frac{2}{3}x_i + \frac{1}{3}x_j \end{bmatrix} \quad \vec{y} := B \begin{bmatrix} \vec{y} \\ \frac{2}{3}x_i + \frac{1}{3}x_j \end{bmatrix} + \vec{b}$$

- If cur is minimal with $\text{cur} < x_i$ appearing in ϕ , we add $((p, \phi), T_1, (q, \phi'))$ and $((p, \phi), T_2, (q, \phi'))$, where T_1 represents the reassignment:

$$\vec{x} := A \begin{bmatrix} \vec{x} \\ x_i - 1 \end{bmatrix} \quad \vec{y} := B \begin{bmatrix} \vec{y} \\ x_i - 1 \end{bmatrix} + \vec{b}$$

and T_2 represents the reassignment:

$$\vec{x} := A \begin{bmatrix} \vec{x} \\ x_i - 2 \end{bmatrix} \quad \vec{y} := B \begin{bmatrix} \vec{y} \\ x_i - 2 \end{bmatrix} + \vec{b}$$

- If cur is maximal with $x_i < \text{cur}$ appearing in ϕ , we add $((p, \phi), T_1, (q, \phi'))$ and $((p, \phi), T_2, (q, \phi'))$, where T_1 represents the reassignment:

$$\vec{x} := A \begin{bmatrix} \vec{x} \\ x_i + 1 \end{bmatrix} \quad \vec{y} := B \begin{bmatrix} \vec{y} \\ x_i + 1 \end{bmatrix} + \vec{b}$$

and T_2 represents the reassignment:

$$\vec{x} := A \begin{bmatrix} \vec{x} \\ x_i + 2 \end{bmatrix} \quad \vec{y} := B \begin{bmatrix} \vec{x} \\ \vec{y} \\ x_i + 2 \end{bmatrix} + \vec{b}$$

We can apply Karr's algorithm starting with $V_{(q_0, \phi_0)} = \{\vec{u}_0\}$ and $V_{(q, \phi)} = \emptyset$, for the other (q, ϕ) . By Lemma 3, there is w such that $\mathcal{A}(w) \neq 0$ if and only if there is a final state q_f and an ordering ϕ such that $\text{aff}(V_{(q_f, \phi)}) \not\subseteq \mathbb{H}$, where \mathbb{H} is the space of the solutions of $\zeta(q_f)(\vec{x}, \vec{y}) = 0$.

Since there are altogether $2^k(k+1)!$ ordering of $X \cup \{\text{cur}\}$, \mathcal{P} has $|Q|2^k(k+1)!$ states with $k+l$ variables. So, overall our algorithm runs in exponential time.

G. Polynomial space algorithm for non-zero problem with constant space assumption for rational numbers in $[-1, 1]$

Let $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ be the input $\text{RA}_{\mathbb{Q}}$ over (X, Y) , where $X = \{x_1, \dots, x_k\}$ and $Y = \{y_1, \dots, y_l\}$. Without loss of generality, we can normalize \mathcal{A} so that it has the following properties.

- There is only one final state q_F whose output function is $\zeta(q_F) = y_1$. It implies that the output of \mathcal{A} on a word w is simply the content of the variable y_1 when it enters the final state q_F .
This can be achieved by adding new transitions that reach q_F , where y_1 is assigned the original output function, and all the variables $x_1, \dots, x_k, y_2, \dots, y_l$ are assigned zero.
- Every transition in δ is of the form $(p, \varphi(\vec{x}, \text{cur})) \rightarrow (q, A, B, 0)$, i.e., $\vec{b} = 0$.
This can be achieved by adding new control variables to store the non-zero constants in \vec{b} .
- We add two new control variables containing 1 and 2.
In every transition the content of these two new variables never change, except in the transition entering the final state q_F , where every variable, except y_1 , is assigned 0.

The algorithm works non-deterministically as follows.

- (1) Guess a positive integer $N \leq |Q|(k+l+1)2^k(k+1)!$.
- (2) For each integer $i \leq N$, compute d_i and t_i , (q_i, \vec{u}_i) such that $(q_{i-1}, \vec{u}_{i-1}) \vdash_{t_i, d_i} (q_i, \vec{u}_i)$ as follows.
Let $\phi = x_{j_1} \otimes_1 \dots \otimes_{k-1} x_{j_k}$ be the ordering of $\vec{u}_{i-1}(X)$, where each \otimes_j is $<$ or $=$.
Guess a transition $t_i = (q_{i-1}, \varphi(\vec{x}, \text{cur})) \rightarrow (q_i, A, B, 0)$ that is consistent with the ordering ϕ of \vec{u}_{i-1} .
Guess d_i according to one of the following cases.
 - (a) If cur is bound to some x_j in $\varphi(\vec{x}, \text{cur})$, then $d_i = \vec{u}_{i-1}(j)$.
 - (b) If $\text{cur} < x_{j_1}$ is consistent with $\varphi(\vec{x}, \text{cur})$, then either $d_i = \vec{u}_{i-1}(j_1) - 1$ or $d_i = \vec{u}_{i-1}(j_1) - 2$.
 - (c) If $\text{cur} > x_{j_k}$ is consistent with $\varphi(\vec{x}, \text{cur})$, then either $d_i = \vec{u}_{i-1}(j_k) + 1$ or $d_i = \vec{u}_{i-1}(j_k) + 2$.
 - (d) If $x_j < \text{cur} < x_{j'}$ is consistent with $\varphi(\vec{x}, \text{cur})$, then either $d_i = \frac{1}{3}\vec{u}_{i-1}(j) + \frac{2}{3}\vec{u}_{i-1}(j')$ or $d_i = \frac{2}{3}\vec{u}_{i-1}(j) + \frac{1}{3}\vec{u}_{i-1}(j')$ for d_i .
Let \vec{u}_i be such that $\vec{u}_i(X) = A \begin{bmatrix} \vec{u}_{i-1}(X) \\ d_i \end{bmatrix}$ and $\vec{u}_i(Y) = B \begin{bmatrix} \vec{u}_{i-1}(Y) \\ d_i \end{bmatrix}$.
Divide \vec{u}_i by $\max_j |\vec{u}_i(j)|$, if the absolute values of some components in \vec{u}_i are bigger than 1.
- (3) Verify that $q_N \in F$ and $\zeta(q_N)(\vec{u}_N) \neq 0$.

For the complexity analysis, we assume each rational number between -1 and 1 occupies constant number of bits. The number N occupies polynomial space. Moreover, on each $i = 1, \dots, N$, the Turing machine only needs to remember the last two configurations (q_{i-1}, \vec{u}_{i-1}) and (q_i, \vec{u}_i) . Hence, the algorithm runs in polynomial space.

The proof of the correctness of our algorithm is as follows. Dividing each \vec{u}_i by $\max_j |\vec{u}_i(j)|$ does not effect the correctness of our algorithm, since $A(\alpha \vec{u}) = \alpha A\vec{u}$, for every non-zero α . By Theorem 3, if there is $w \in L(\mathcal{A})$ such that $\mathcal{A}(w) \neq 0$, we can assume that $|w| \leq |Q|(k+l+1)2^k(k+1)!$. The correctness immediately follows from Lemma 3.

H. Proof of Theorem 3 and the tightness of the bound

Proof of Theorem 3: Note that there can be only $2^k(k+1)!$ different orderings of the content of the control variables. Let $w = d_1 \dots d_N$, where $N > |Q|(k+l+1)2^k(k+1)!$ with its accepting run as follows:

$$(q_0, \vec{u}_0) \vdash_{t_1, d_1} \dots \vdash_{t_N, d_N} (q_N, \vec{u}_N) \quad \text{and} \quad q_N \in F,$$

where $\zeta(q_N)(\vec{u}_N) \neq 0$. Since $N > |Q|(k+l+1)2^k(k+1)!$, there are at least $k+l+2$ configurations in the run with the same ordering, say ϕ . Let $(p, \vec{v}_1), \dots, (p, \vec{v}_{m+1})$ be such configurations with the ordering ϕ , where $m \geq k+l+1$. Let us also denote by T_0, \dots, T_{m+1} the sequences of transitions such that:

$$(q_0, \vec{u}_0) \vdash_{T_0} (p, \vec{v}_1) \vdash_{T_1} \dots \vdash_{T_m} (p, \vec{v}_{m+1}) \vdash_{T_{m+1}} (q_N, \vec{u}_N)$$

Let \mathbb{A} be the affine space $\{\vec{z} \mid \zeta(q_N)(\vec{z}) = 0\}$, and let \mathbb{H} be the pre-image of \mathbb{A} under T_{m+1} . Since $\vec{u}_N \notin \mathbb{A}$, we have $\vec{v}_{m+1} \notin \mathbb{H}$. From this, we deduce that $\dim(\mathbb{H}) \leq k + l - 1$, so $m \geq \dim(\mathbb{H}) + 2$. By Lemma 2 there is a set $J = \{j_1, \dots, j_n\}$ such that $|J| \leq k + l$ and $\vec{z}_1, \dots, \vec{z}_{n+1}$ such that $\vec{z}_1 = \vec{v}_1$,

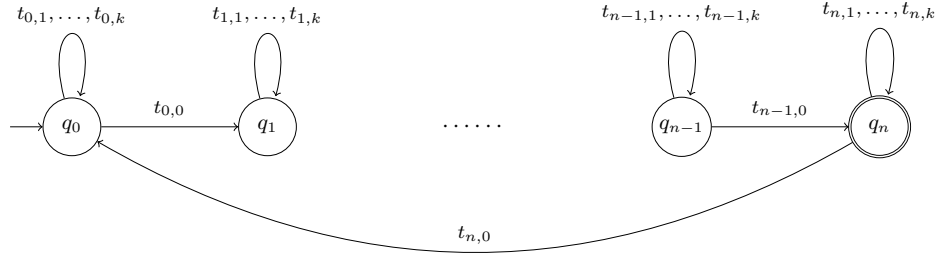
$$(q_0, \vec{u}_0) \vdash_{T_0} (p, \vec{z}_1) \vdash_{T_{j_1}} \dots \vdash_{T_{j_n}} (p, \vec{z}_{n+1}),$$

and $\vec{z}_{n+1} \notin \mathbb{H}$. Furthermore, because each of T_1, \dots, T_m transforms each configuration (p, \vec{u}) of the ordering ϕ to another configuration (p, \vec{v}) with the same ordering, $\vec{v}_1, \dots, \vec{v}_{m+1}, \vec{z}_1, \dots, \vec{z}_{n+1}$ have the same orderings. Since \mathbb{H} is the pre-image of $\mathbb{A} = \{\vec{z} \mid \zeta(q_N)(\vec{z}) = 0\}$ under T_{m+1} , we have:

$$(q_0, \vec{u}_0) \vdash_{T_0} (p, \vec{z}_1) \vdash_{T_{j_1}} \dots \vdash_{T_{j_n}} (p, \vec{z}_{n+1}) \vdash_{T_{m+1}} (q_N, \vec{a})$$

for some $\vec{a} \notin \mathbb{A}$, and therefore, $\zeta(q_N)(\vec{a}) \neq 0$. This completes our proof.

Tightness of the exponential bound: The exponential bound in Theorem 3 above is tight. The idea is almost the same as the $\text{RA}_{\mathbb{Q}}$ that represents the number p^n in Section III. Consider the following $\text{RA}_{\mathbb{Q}}$ $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ over (X, Y) , where $X = \{x_1, \dots, x_k, x_{k+1}, x_{k+2}\}$, $Y = \{y_1, \dots, y_l\}$, $Q = \{q_0, \dots, q_n\}$, $F = \{q_n\}$ and the transitions $t_{i,j}$, where $0 \leq i \leq n$ and $0 \leq j \leq k$, as illustrated below.



During the computation, the contents of the variables x_{k+1} and x_{k+2} never change, and they contain 0 and 1, respectively.

- The initial content of the control variables $(x_1, \dots, x_k, x_{k+1}, x_{k+2}) = (0, \dots, 0, 0, 1)$, i.e., all x_i 's are initially 0, except x_{k+2} which is initially 1.
- The initial content of the data variables $(y_1, y_2, \dots, y_l) = (1, 0, \dots, 0)$, i.e., y_1 is initially 1, and the rest are initially 0.
- The function $\zeta(q_n) = y_l$, i.e., the output function associated with q_n is the function $g(\vec{x}, \vec{y}) = y_l$.
- For each $0 \leq i \leq n$, the transition $t_{i,0}$ is

$$(q_i, \varphi_k) \rightarrow (q_{(i+1) \bmod (n+1)}, A_{i,0}, B, 0),$$

where

- the guard φ_k is $\bigwedge_{h=1}^k x_h = 1$;
- the matrix $A_{i,0}$ represents the assignment $\{x_1 := 0; \dots; x_k := 0\}$;
- the matrix B represents the “shift right” of the content of the data variables, i.e., $\{y_1 := y_l; y_2 := y_1; \dots; y_l := y_{l-1}\}$.

Here the equality and assignment of the variables with 0 and 1 can be done via the variables x_{k+1} and x_{k+2} , whose contents are always 0 and 1, respectively.

- For each $0 \leq i \leq n$ and $1 \leq j \leq k$, the transition $t_{i,j}$ is

$$(q_i, \varphi_j) \rightarrow (q_{(i+1) \bmod (n+1)}, A_{i,j}, B, 0),$$

where

- the guard φ_j is $x_j = 0 \wedge \bigwedge_{h=1}^{j-1} x_h = 1$;
- the matrix $A_{i,j}$ represents the assignment $\{x_1 := 0; \dots; x_{j-1} := 0; x_j := 1\}$;
- the matrix B represents the assignment $\{y_1 := y_1, \dots, y_l := y_l\}$, i.e., it does not change the content of the data variables y_1, \dots, y_l .

Again, the equality and assignment of the variables with 0 and 1 can be done via the variables x_{k+1} and x_{k+2} , whose contents are always 0 and 1, respectively.

If $(n + 1)$ and l are coprime, then the length of any path from the initial configuration (q_0, \vec{u}_0) to an accepting configuration that outputs non-zero is a multiple of $l(n + 1)2^k$.

I. Proof of Lemma 4

The proof is similar to the proof of Lemma 3 in Appendix E.

The idea is that if $(q_j, \vec{u}_j) \vdash_{t_j, d_j} (q_{j+1}, \vec{u}_{j+1})$, where $\vec{u}_{j+1} \notin \mathbb{H}$ and d_j does not appear in $\vec{u}_j(x)$, then there are infinitely many other values c such that $(q_j, \vec{u}_j) \vdash_{t_j, c} (q_{j+1}, \vec{v})$, where $\vec{v} \notin \mathbb{H}$. Indeed, if d_j does not appear in $\vec{u}_j(x)$, by the density of rational numbers, there are infinitely many values c for cur such that the guard of t_j is satisfied. By Lemma 1, for all, but one, such c , we have $(q_j, \vec{u}_j) \vdash_{t_j, c} (q_{j+1}, \vec{v})$, where $\vec{v} \notin \mathbb{H}$.

J. The PSPACE-hardness reductions

In this appendix, we will establish all the PSPACE-hardness mentioned in this paper, i.e., for the following problems.

- The non-zero, equivalence, and commutativity problems for copyless $\text{RA}_{\mathbb{Q}}$.
- The equivalence and commutativity problems for deterministic RA.

It has been shown that the non-emptiness problem for deterministic standard RA is PSPACE-complete [9]. All of our reductions either trivially follow their reduction, or are simply slight modifications of theirs.

For clarity and completeness, we will repeat their reduction here, and then present our reductions. It is from the following PSPACE-complete problem. Let $c > 0$ be a constant. *On input $[\mathcal{M}]$ (a Turing machine description) and a word $w \in \{0, 1\}^*$, decide whether \mathcal{M} accepts w using at most $c|w|$ space.*

Reduction 1: To the non-emptiness of deterministic standard RA [9]: Construct an RA \mathcal{A} with $c|w| + 3$ registers. The first three registers contain 0, 1 and \sqcup , respectively, where \sqcup represents the blank symbol in Turing machine. The next $|w|$ registers contain the bits of w . Define the states of \mathcal{A} as pairs (p, j) , where p is a state of \mathcal{M} and $1 \leq j \leq c|w|$ indicating the position of the head of \mathcal{M} . On state (p, j) , \mathcal{A} can use transitions to check whether the content of register $j + 3$ is one of 0, 1 or \sqcup by comparing its content with the first three registers and simulate the transitions of \mathcal{M} . The final states are those (p, j) , where p is the accepting state of \mathcal{M} . Thus, \mathcal{M} accepts w using at most $c|w|$ space if and only if \mathcal{A} can reach one of its final states.

Reduction 2: To the equivalence problem for deterministic standard RA: Note that an RA accepts some word iff it is not equivalent to a trivial RA that accepts nothing. Since PSPACE is closed under complement, the hardness follows.

Reduction 3: To the commutativity problem for deterministic standard RA: Indeed, we can modify the constructed RA \mathcal{A} in reduction 1 such that it enforces the first two values are 0 and the subsequent values are all different from the 0. Obviously, if the machine \mathcal{M} accepts w using at most $c|w|$ space, then \mathcal{A} accepts some word, and \mathcal{A} is not commutative, since it requires the first two values are 0 and the rest are non-zero. On the other hand, if \mathcal{M} does not accept w using at most $c|w|$ space, \mathcal{A} accepts nothing, which implies that \mathcal{A} is commutative. This completes the reduction.

Reduction 4: To the non-zero problem for copyless $\text{RA}_{\mathbb{Q}}$: Standard RA are copyless $\text{RA}_{\mathbb{Q}}$ that outputs constant 1 in the final states. So, it follows from reduction 1.

Reduction 5: To the equivalence problem for copyless $\text{RA}_{\mathbb{Q}}$: Like in reduction 2, this follows from the fact that PSPACE is closed under complement. An $\text{RA}_{\mathbb{Q}}$ \mathcal{A} outputs a non-zero value for some input word iff it is not equivalent to a trivial $\text{RA}_{\mathbb{Q}}$ that always outputs 0.

Reduction 6: To the commutativity problem for copyless $\text{RA}_{\mathbb{Q}}$: Standard RA are copyless $\text{RA}_{\mathbb{Q}}$ that outputs constant 1 in the final states. So, it follows from reduction 3.

K. Proof of Theorem 5

First, both the commutativity and equivalence problems of non-deterministic RA can be reduced to the corresponding problems of single-valued $\text{RA}_{\mathbb{Q}}$, because non-deterministic RA is a special case of single-valued $\text{RA}_{\mathbb{Q}}$.

Second, the equivalence/universality problem of non-deterministic register automata has been proved to be undecidable in Theorem 5.1 of [15], by constructing an RA that accepts the set of words that encode all non-solutions to a PCP instance. Thus, if the PCP has a solution, the RA is not commutative, because the permutations of the solution are simply arbitrary string. Likewise, if the RA is not commutative, hence, not universal, the PCP has a solution. Therefore, the PCP instance has a solution if and only if the constructed RA is not commutative.

L. Proof of Theorem 6

From invariant to non-zero

Note that the Karp reductions between the invariant problem and the non-zero problem from Section IV cannot be used, because they construct nondeterministic $\text{RA}_{\mathbb{Q}}$. Therefore, we modify the said reductions into the following Cook reductions that preserve determinism.

Given an $\text{RA}_{\mathbb{Q}}$ \mathcal{A} , a state q , and $\mathbb{H} = \vec{a} + \mathbb{V}$, similarly to the reduction from Section IV, we compute a basis $\{\vec{v}_1, \dots, \vec{v}_m\}$ of the orthogonal complement of \mathbb{V} and transform \mathcal{A} into m new $\text{RA}_{\mathbb{Q}}$ $\mathcal{A}_1, \dots, \mathcal{A}_m$, all having a single final state q , such that the output function of \mathcal{A}_i on q is $\begin{pmatrix} \vec{x} \\ \vec{y} \end{pmatrix} - \vec{a} \cdot \vec{v}_i$. Each of these automata is then tested for the non-zero problem.

From non-zero to invariant

For an $\text{RA}_{\mathbb{Q}}$ \mathcal{A} with final states $\{q_1, \dots, q_n\}$ and output function ζ , we test for every q_i whether $\text{vec}(\mathcal{A}, q_i) \subseteq \mathbb{H}_i$, where \mathbb{H}_i is the space of solutions of the system $\zeta(q_i)(\vec{x}, \vec{y}) = 0$.

From equivalence to non-zero

Let $\mathcal{A}_1 = \langle Q_1, q_{0,1}, F_1, \vec{u}_{0,1}, \delta_1, \zeta_1 \rangle$ over (X_1, Y_1) and $\mathcal{A}_2 = \langle Q_2, q_{0,2}, F_2, \vec{u}_{0,2}, \delta_2, \zeta_2 \rangle$ over (X_2, Y_2) be deterministic $\text{RA}_{\mathbb{Q}}$. Here we use a function instead of a vector to represent the initial content of variables to ease presentation. Assume w.l.o.g. that the sets of variables in \mathcal{A}_1 and \mathcal{A}_2 are disjoint.

First, we make each \mathcal{A}_i complete. This can be done in a linear time by adding a new sink state without affecting the semantics of \mathcal{A}_i . Next, we construct a new $\text{RA}_{\mathbb{Q}}$ \mathcal{A} over $(X_1 \cup X_2, Y_1 \cup Y_2)$ such that \mathcal{A}_1 and \mathcal{A}_2 are equivalent iff $\mathcal{A}(w) = 0$ for all w . The construction is the standard product construction. Note that \mathcal{A} is deterministic.

- The set of states is $Q_1 \times Q_2$.
- The initial state is $(q_{0,1}, q_{0,2})$.
- The set of final states is $Q_1 \times F_2 \cup F_1 \times Q_2$.
- The initial content of the variables is the function $\{z \mapsto \vec{u}_{0,1}(z) \mid z \in X_1 \cup Y_1\} \cup \{z \mapsto \vec{u}_{0,2}(z) \mid z \in X_2 \cup Y_2\}$.
- The set of transitions δ consists of the following.

For each pair of transitions t_1, t_2 , where $t_1 = (p_1, g_1) \rightarrow (q_1, M_1) \in \delta_1$ and $t_2 = (p_2, g_2) \rightarrow (q_2, M_2) \in \delta_2$, and M_1, M_2 represent the reassignments, we add a new transition $((p_1, p_2), g_1 \wedge g_2) \rightarrow ((q_1, q_2), M_1 \cup M_2)$ in δ .

- The output function on state $(q_1, q_2) \in F_1 \times F_2$ is $\zeta_1(q_1) - \zeta_2(q_2)$. Otherwise, for pairs (q_1, q_2) , where one of them is non-final, the output is the constant 1.

From non-zero to commutativity

Let \mathcal{A} be an $\text{RA}_{\mathbb{Q}}$. We first apply the following changes to \mathcal{A} :

- we make \mathcal{A} complete (see the previous reduction) and
- we change all non-final states in \mathcal{A} into final states that output the constant 0.

Both changes can be done in a linear time and will not affect the result of the non-zero problem on \mathcal{A} . Now we have $|\mathcal{A}(w)| = 1$ for all w , i.e., the output of \mathcal{A} is “defined” on all inputs w .

The idea is as follows. We construct an $\text{RA}_{\mathbb{Q}}$ \mathcal{A}' such that \mathcal{A}' outputs the following.

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

We construct \mathcal{A} in a linear time by adding two new states that check the first two input values and a new final state that outputs the constant 0 for words failed the check, i.e., those that do not begin with 12.

Below we show that there is a word w such that $\mathcal{A}(w) \neq 0$ if and only if \mathcal{A}' is not commutative.

- Assume there is w such that $\mathcal{A}(w) \neq 0$: Then $\mathcal{A}'(12w) = \mathcal{A}(w) \neq 0$. By definition, $\mathcal{A}'(21w) = 0$ and $21w \in \text{perm}(12w)$. It follows that \mathcal{A}' is not commutative.
- Assume that $\mathcal{A}(w) = 0$ for all w :
 - For any word v of the form $12w$, it holds that $\mathcal{A}'(v) = \mathcal{A}'(12w) = \mathcal{A}(w) = 0$.
 - For any word v not in form of $12w$, it holds that $\mathcal{A}'(v) = 0$ (by definition).

It follows that \mathcal{A}' outputs 0 on all inputs and, hence, \mathcal{A}' is commutative.

From commutativity to equivalence

Let $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ be an $\text{RA}_{\mathbb{Q}}$ over $X \cup Y$. (Again we use a function instead of a vector to represent the initial contents of variables.) W.l.o.g we assume that $\mathcal{A}(w) = \emptyset$ for all w such that $|w| \leq 1$ (commutativity of words of lengths zero or one is trivial). We show the construction of deterministic $\text{RA}_{\mathbb{Q}}$ \mathcal{A}_1 and \mathcal{A}_2 such that, for every word w , $\mathcal{A}_1(w) = \mathcal{A}(\pi_2(w))$ and $\mathcal{A}_2(w) = \mathcal{A}(\pi_*(w))$.

The construction of \mathcal{A}_1 is straightforward, thus omitted. We focus on the construction of \mathcal{A}_2 . First, we make a copy X' of X , and Y' of Y . That is, for every $x \in X$, we have its corresponding primed version $x' \in X'$. Likewise for every $y \in Y$.

Assuming that the states in Q are of the form q_i , define a set $F' = \{q_{(i \rightarrow j)} \mid q_i \in Q, q_j \in F\}$. Intuitively, the set F' has a copy of each state q_i for each final state q_j .

We construct an $\text{RA}_{\mathbb{Q}}$ $\mathcal{A}_2 = \langle Q_2, q_{0,2}, F_2, \vec{u}_{0,2}, \delta_2, \zeta_2 \rangle$ over (X_2, Y_2) , where the components of \mathcal{A}_2 and (X_2, Y_2) are defined as follows:

- $X_2 = X \cup X' \cup \{x_t\}$, where x_t is a new control variable, and $Y_2 = Y \cup Y'$.
- $Q_2 = Q \cup F' \cup \{q_{0,2}\}$, where $q_{0,2}$ is a new state and F' is as defined above.
- The new state $q_{0,2}$ is the initial state.
- $F_2 = F'$ as defined above is the set of final states.

- $\vec{u}_{0,2} = \{z \mapsto \vec{u}_0(z) \mid z \in X \cup Y\} \cup \{z \mapsto 0 \mid z \in X' \cup Y' \cup \{x_t\}\}$.
- ζ_2 is defined as follows.

For all $q_{(i \rightarrow j)} \in F'$, the output function $\zeta_2(q_{(i \rightarrow j)})$ is the same as $\zeta(q_j)$ but substituting all the variables with their primed versions. For example, if $\zeta(q_j) = x_1 + 3y_3$ then $\zeta_2(q_{(i \rightarrow j)}) = x'_1 + 3y'_3$.

Before the construction of the transition rules δ_2 , let us first explain the intuition behind.

- For each state q_i in \mathcal{A} , there are $|F| + 1$ copies in \mathcal{A}_2 , that is, the state q_i , and the states $q_{i,j}$ with $q_j \in F$.
- For each state $q_{i,j}$ in \mathcal{A}_2 and every two transitions t_1, t_2 in \mathcal{A} from some state q_{i_0} to q_i and from q_i to q_j respectively, \mathcal{A}_2 includes a transition t' from q_{i_0} to $q_{i,j}$ which reassigns \vec{x}' and \vec{y}' with the new values of \vec{x} and \vec{y} after the two transitions (that is, t_1 and t_2), while still reassigns \vec{x} and \vec{y} with their values after t_1 only.
- The only purpose of the variables \vec{x}' and \vec{y}' is to be used in the output. (Recall that $\zeta(q_{i,j})$ is defined as $\zeta(q_j)$, with \vec{x} and \vec{y} substituted by \vec{x}' and \vec{y}' respectively).
- In addition, for each state q_{i_1} in \mathcal{A} and each copy of q_{i_1} in \mathcal{A}_2 , say $q_{i_1,j}$ for instance, each transition $t = (q_{i_1}, \varphi(\vec{x}, \text{cur})) \rightarrow (q_{i_2}, A, B, \vec{b})$ in \mathcal{A} is split into multiple transitions of \mathcal{A}_2 out of $q_{i_1,j}$, by splitting the guard $\varphi(\vec{x}, \text{cur})$ into mutually disjoint constraints, in order to make sure that \mathcal{A}_2 is still deterministic.

The transitions in δ_2 are defined as follows.

[T1] δ_2 contains the following transition to store the first cur in x_t

$$(q_{0,2}, \text{true}) \rightarrow (q_0, \{x_t := \text{cur}\})$$

[T2] For each pair of transitions $t_1 = (q_{i_1}, \varphi(\vec{x}, \text{cur})) \rightarrow (q_{i_2}, A_1, B_1, \vec{b}_1), t_2 = (q_{i_2}, \varphi'(\vec{x}, \text{cur})) \rightarrow (q_{i_3}, A_2, B_2, \vec{b}_2) \in \delta$ such that $q_{i_3} \in F$, δ_2 contains the following transitions to summarize the effect of t_1 and t_2 :

$$\begin{aligned} (q_{i_1}, (\varphi(\vec{x}, \text{cur}) \wedge \varphi'(A_1 \begin{bmatrix} \vec{x} \\ \text{cur} \end{bmatrix}, x_t))) &\rightarrow (q_{(i_2 \rightarrow i_3)}, M) \\ \forall q_j \in F, (q_{(i_1 \rightarrow j)}, (\varphi(\vec{x}, \text{cur}) \wedge \varphi'(A_1 \begin{bmatrix} \vec{x} \\ \text{cur} \end{bmatrix}, x_t))) &\rightarrow (q_{(i_2 \rightarrow i_3)}, M), \end{aligned}$$

where M encodes $\vec{x} := A_1 \begin{bmatrix} \vec{x} \\ \text{cur} \end{bmatrix}, \vec{y} := B_1 \begin{bmatrix} \vec{y} \\ \text{cur} \end{bmatrix} + \vec{b}_1, \vec{x}' := A_2 \begin{bmatrix} \vec{x} \\ \text{cur} \\ x_t \end{bmatrix}, \vec{y}' := B_2 \begin{bmatrix} A_1 \begin{bmatrix} \vec{x} \\ \text{cur} \end{bmatrix} \\ B_1 \begin{bmatrix} \vec{y} \\ \text{cur} \end{bmatrix} + \vec{b}_1 \\ x_t \end{bmatrix} + \vec{b}_2$, and $x_t := x_t$.

Intuitively, M updates variables in $X \cup Y$ in the same way as t_1 does, updates variables in $X' \cup Y'$ to summarize the effect of t_1 and t_2 using cur as the first and x_t as the second input symbol. The summarization is achievable by Proposition 1.

[T3] We define $T_{q,F} \subseteq \delta$ as the set of transitions starting from q and ending at a final state F . For each transition $t_1 = (q_{i_1}, \varphi(\vec{x}, \text{cur})) \rightarrow (q_{i_2}, A, B, \vec{b})$, δ_2 contains the following transitions

$$\begin{aligned} (q_{i_1}, \varphi''(\vec{x}, x_t, \text{cur})) &\rightarrow (q_{i_2}, M) \\ \forall q_j \in F, (q_{(i_1 \rightarrow j)}, \varphi''(\vec{x}, x_t, \text{cur})) &\rightarrow (q_{i_2}, M), \end{aligned}$$

where $\varphi''(\vec{x}, x_t, \text{cur})$ is a predicate defines as

$$\varphi(\vec{x}, \text{cur}) \wedge \bigwedge \{ \neg \varphi'(A \begin{bmatrix} \vec{x} \\ \text{cur} \end{bmatrix}, x_t) \mid \varphi'(\vec{x}, \text{cur}) \text{ is the guard of a transition in } T_{q_{i_2}, F} \},$$

and M encodes $\vec{x} := A \begin{bmatrix} \vec{x} \\ \text{cur} \end{bmatrix}, \vec{y} := B \begin{bmatrix} \vec{y} \\ \text{cur} \end{bmatrix} + \vec{b}, \vec{x}' := \vec{x}', \vec{y}' := \vec{y}'$, and $x_t := x_t$.

Intuitively, the guard enforces that \mathcal{A}_2 can take the **[T3]**-type transition from a configuration (q_{i_1}, \vec{u}) only when all **[T2]**-type transitions from q_{i_1} cannot be taken. The variables in $X \cup Y$ are updated in the same way as t_1 does and the values of other variables remain unchanged.

Observe that if we reset the initial state of \mathcal{A}_2 to q_0 , reset the final states of \mathcal{A}_2 to F , and use ζ as the output function, then \mathcal{A}_2 and \mathcal{A} are equivalent.

M. Proof of Theorem 7

We show undecidability of the reachability problem by a reduction from the Post correspondence problem. Let $\mathcal{P} = \{(u_i, v_i)\}_{1 \leq i \leq n}$ be a set of pairs of sequences over the alphabet $\Sigma = \{1, \dots, b-1\}$ for some base b , i.e., for all $1 \leq j \leq n$ it holds that $u_i, v_i \in \Sigma^*$. The *Post correspondence problem* (PCP) asks whether there exists a sequence $i_1 \dots i_k \in \{1, \dots, n\}^+$ of indices such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$. The PCP is well known to be undecidable [29].

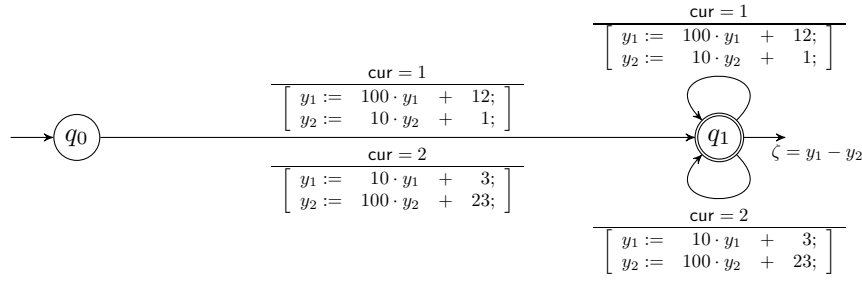


Fig. 2. An example of our encoding of the PCP instance $\{(1.2, 1), (3, 2.3)\}$ over the alphabet $\{1, \dots, 9\}$ into an SNT (all numbers are given in base-4).

Given \mathcal{P} , we build a deterministic $\text{RA}_{\mathbb{Q}}$ $\mathcal{A}_{\mathcal{P}} = \langle \{q_0, q_1\}, q_0, \{q_1\}, \vec{u}_0, \delta, \zeta \rangle$ over $X = \emptyset$ and $Y = \{y_1, y_2\}$, where $\vec{u}_0 = [0, 0]$ (i.e., the initial configuration of registers is $y_1 = 0$ and $y_2 = 0$), the output function is $\zeta(q_1) = y_1 - y_2$, and δ is constructed as described later. The reduction is based on treating strings over Σ as natural numbers in base- b encoding and representing concatenation of strings using linear expressions, e.g., if we assume $b = 10$, the concatenation of strings $1.2.3 \in \Sigma^*$ and $4.5.6 \in \Sigma^*$ can be represented using natural numbers as $123 \cdot 10^3 + 456 = 123456$. In particular, for every $1 \leq i \leq n$, we translate the pair $(u_i, v_i) \in \mathcal{P}$ into the pair of transitions

$$\begin{aligned} (q_0, (\text{cur} = i)) &\rightarrow (q_1, [\omega_1; \omega_2;]) \in \delta \\ (q_1, (\text{cur} = i)) &\rightarrow (q_1, [\omega_1; \omega_2;]) \in \delta, \end{aligned}$$

such that, if $u_i = g_1 \cdots g_r$ and $v_i = h_1 \cdots h_s$, the term ω_1 is the assignment $y_1 := b^r \cdot y_1 + e(g_1 \cdots g_r)$ and ω_2 is the assignment $y_2 := b^s \cdot y_2 + e(h_1 \cdots h_s)$ where we use $e(g_1 \cdots g_r)$ to represent the number $g_1 b^{r-1} + \dots + g_r b^0$ (similarly for $e(h_1 \cdots h_s)$); for instance, for $b = 5$, we have $e(4.2) = 42_5$ (note that $e : \Sigma^* \rightarrow \mathbb{N}$). In Figure 2, we show an example of an encoding of a PCP instance into an SNT. The transitions from q_0 to q_1 are necessary in order to “bootstrap” the computation (a solution to a PCP cannot be an empty sequence of indices). The crucial property of $\mathcal{A}_{\mathcal{P}}$ is that for a sequence $i_1 \cdots i_k \in \{1, \dots, n\}^+$, it holds that $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$ (i.e., $i_1 \cdots i_k$ is a solution of \mathcal{P}) iff $\mathcal{A}_{\mathcal{P}}(i_1 \cdots i_k) = 0$, which we prove in the following.

First, we want to show that after reading $i_1 \cdots i_l$, for $l \geq 1$ and $\forall 1 \leq k \leq l : 1 \leq i_k \leq n$, \mathcal{A} is in the state q_1 and the content of the variable y_1 is $e(u_{i_1} \cdots u_{i_l})$ and the content of the variable y_2 is $e(v_{i_1} \cdots v_{i_l})$. We show the previous by induction on the length of $i_1 \cdots i_l$. For $l = 1$, after reading i_1 , $\mathcal{A}_{\mathcal{P}}$ will move from the state q_0 to the state q_1 , taking the transition $(q_0, (\text{cur} = i_1)) \rightarrow (q_1, [\omega_1; \omega_2;]) \in \delta$, which sets the new content of the variable y_1 to $0 + e(u_{i_1})$ and the content of the variable y_2 to $0 + e(v_{i_1})$. For the inductive step, assume that $\mathcal{A}_{\mathcal{P}}$ is after reading $i_1 \cdots i_l$ in the state q_1 , the content of the registers y_1 and y_2 being $e(u_{i_1} \cdots u_{i_l})$ and $e(v_{i_1} \cdots v_{i_l})$ respectively. If $\mathcal{A}_{\mathcal{P}}$ now reads the symbol i_{l+1} , it takes the transition $(q_1, (\text{cur} = i_{l+1})) \rightarrow (q_1, [\omega_1; \omega_2;]) \in \delta$ where ω_1 is of the form $y_1 := y_1 b^r + e(u_{i_{l+1}} = g_1 \cdots g_r)$ and ω_2 is of the form $y_2 := y_2 b^s + e(v_{i_{l+1}} = h_1 \cdots h_s)$. The new value of the variable y_1 will therefore be $e(u_{i_1} \cdots u_{i_l}) \cdot b^r + e(g_1 \cdots g_r) = e(u_{i_1} \cdots u_{i_l} \cdot u_{i_{l+1}})$, the new value of the variable y_2 will be $e(v_{i_1} \cdots v_{i_l}) \cdot b^s + e(h_1 \cdots h_s) = e(v_{i_1} \cdots v_{i_l} \cdot v_{i_{l+1}})$, and the next state will be q_1 , which concludes this part of the proof.

Based on the previous paragraph and the fact that the only output of $\mathcal{A}_{\mathcal{P}}$ is in the state q_1 and has the value computed as $y_1 - y_2$, we infer that

$$\mathcal{A}_{\mathcal{P}}(i_1 \cdots i_k) = e(u_{i_1} \cdots u_{i_k}) - e(v_{i_1} \cdots v_{i_k}).$$

What remains to show is the following:

$$u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k} \quad \text{iff} \quad e(u_{i_1} \cdots u_{i_k}) = e(v_{i_1} \cdots v_{i_k}),$$

which holds because e is a bijection.

A more succinct proof can also be obtained by noticing that the reachability problem of $\text{RA}_{\mathbb{Q}}$ s can encode instances of the scalar reachability problem of matrices, which is undecidable [?].

N. Proof of Theorem 8: The decision procedure

In the following, we present a proof of Theorem 8.

Let $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ be a copyless $\text{RA}_{\mathbb{Q}}$ with non-strict transition guards over (X, Y) , where $X = \{x_1, \dots, x_k\}$ and $Y = \{y_1, \dots, y_l\}$. Let \mathcal{N} be the set of constants found in $\vec{u}_0(X)$. W.l.o.g., we assume that $0 \in \mathcal{N}$, that is, the constant 0 is in the initial contents control variables. In addition, let c_{\min} and c_{\max} be the minimum and maximum constant in \mathcal{N} and $\mathcal{N}_{\infty} = \{-\infty, +\infty\} \cup \mathcal{N}$.

To simplify the presentation, we first apply a normalization procedure to \mathcal{A} . By abuse the notation, we still use \mathcal{A} to denote the normalized $\text{RA}_{\mathbb{Q}}$. The normalized $\text{RA}_{\mathbb{Q}}$ \mathcal{A} satisfies the following properties.

- 1) The set of control variables X is partitioned into X_1, X_2 such that X_1 is the set of read-only control variables and X_2 holds all the constants in \mathcal{N} , more specifically, there is a bijection cst between X_2 and \mathcal{N} (intuitively, each variable $x \in X_2$ stores the constant $\text{cst}(x)$).
- 2) For each state $q \in Q$, a total preorder over X , denoted by \preceq_q , is associated with q , which is consistent with the rational order relation on \mathcal{N} , that is, for every $x, x' \in X_1$, $x \preceq_q x'$ iff $\text{cst}(x) \leq \text{cst}(x')$. We will use \simeq_q to denote the equivalence relation induced by \preceq_q , that is, $x \simeq_q x'$ iff $x \preceq_q x'$ and $x' \preceq_q x$. In addition, let $\prec_q = \preceq_q \setminus \simeq_q$. Note that the total preorders \preceq_q for $q \in Q$ can be seen as a reformulation of the orderings used in Section IV.
- 3) For each transition $(p, \varphi(\vec{x}, \text{cur})) \rightarrow (q, A, B, \vec{b})$, let $[z'_1]_p, \dots, [z'_{k'}]_p$ be an enumeration of the equivalence classes of \simeq_p such that $z'_1 \prec_p \dots \prec_p z'_{k'}$, then the guard $\varphi(\vec{x}, \text{cur})$ is of the form $\text{cur} = z'_i$ with $i \in [k']$, or $z'_i \leq \text{cur} \leq z'_{i+1}$ with $i \in [k' - 1]$, or $\text{cur} \leq z'_1$, or $z'_{k'} \leq \text{cur}$.

For an $\text{RA}_{\mathbb{Q}}$ \mathcal{A} of n states and k control variables, k additional read-only control variables may be introduced to store the constants, then the $\text{RA}_{\mathbb{Q}}$ after normalization has at most $O(n2^{2k-1}(2k)!)$ states, where $2^{2k-1}(2k)!$ is the number of orderings for $2k$ control variables (cf. Section IV), which is an upper bound on the number of total preorders for $2k$ control variables.

From now on, we assume that \mathcal{A} is normalized.

Suppose there is a word $w = d_1 \cdots d_n$ that leads to 0. Let the run be $(q_0, \vec{u}_0) \vdash_{t_1, d_1} (q_1, \vec{u}_1) \vdash_{t_2, d_2} \cdots \vdash_{t_n, d_n} (q_n, \vec{u}_n)$. By Proposition 1, there are M and \vec{b} such that

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

Now, these values d_1, \dots, d_n satisfies a set of inequalities imposed by the transitions t_1, \dots, t_n . Let $\Phi(d_1, \dots, d_n)$ denote the conjunction of those inequalities. Note that due to the fact that \mathcal{A} is normalized, the guards in t_1, \dots, 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 be $\vec{a} \cdot \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} + a'$. Define the following function:

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

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

$$f((d_1, \dots, d_n)^t) = 0 \quad \wedge \quad \Phi((d_1, \dots, d_n)^t) = \text{true}.$$

It follows that

$$\exists \vec{z}_1, \vec{z}_2 \in \mathbb{Q}^n : f(\vec{z}_1) \leq 0 \leq f(\vec{z}_2) \wedge \Phi(\vec{z}_1) \wedge \Phi(\vec{z}_2). \quad (3)$$

Observe that (3) holds iff the following two constraints hold simultaneously:

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

By the Simplex algorithm for linear programming [30], 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 $\Phi(\vec{z})$ 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 represented by $\Phi(\vec{z})$ have components from \mathcal{N}_{∞} .

To establish F1 and F2, it is sufficient to find two corner points \vec{z}_1 and \vec{z}_2 such that:

$$f(\vec{z}_1) \leq 0 \leq f(\vec{z}_2)$$

To find these two points, we will construct a \mathbb{Q} -VASS \mathcal{B} , whose reachability is decidable in NP. (See Appendix O.)

Let $q \in Q$. A *specification* of X w.r.t. q is a mapping η from X to \mathcal{N}_{∞} that respects \preceq_q , i.e., 1) for each $x_i \in X_1$, $\eta(x_i) = \text{cst}(x_i)$, and 2) for each $x_i, x_j \in X$, if $x_i \preceq_q x_j$, then $\eta(x_i) \leq \eta(x_j)$. Intuitively, η encodes the value of x_i of a corner point, and $\eta(x_i) = c \in \mathcal{N}$ means that x_i is assigned with c ,

For $m, n \in \mathbb{N}$, we will use $[m]$ to denote $\{1, \dots, m\}$, and $[m, n]$ to denote $\{m, \dots, n\}$, provided that $m \leq n$.

We will construct a two-dimensional \mathbb{Q} -VASS $\mathcal{B} = (S, \Delta)$ with two rational variables C_1, C_2 . The set S of states comprises two special states q'_0, q'_f (whose purpose will become clear later) and the set of tuples $(q, q_f, \eta_1, \eta_2, \pi)$ such that

- $q \in Q, q_f \in F$,

- η_1, η_2 are the specifications of X w.r.t. \preceq_q ,
- π is a mapping from $[l]$ to $[l] \cup \{0\}$.

Intuitively,

- q represents the current state of \mathcal{A} ,
- q_f represents the final state of \mathcal{A} that we will reach eventually,
- η_1 and η_2 represent the two points \vec{z}_1 and \vec{z}_2 we are looking for,
- for each $i \in [l]$, if $\pi(i) \in [l]$, then the current value of y_i will be stored in $y_{\pi(i)}$ when reaching the state q_f , otherwise, the current value of y_i will be lost eventually,
- C_1, C_2 represent the values of $\zeta(q_f)(\vec{z}_1)$ and $\zeta(q_f)(\vec{z}_2)$ respectively.

Before defining Δ , we introduce some additional notations.

Suppose $t = (p, \varphi) \rightarrow (q, A, B, \vec{b})$ in δ is a transition of \mathcal{A} and $(p, q_f, \eta_1, \eta_2, \pi) \in S$.

The mapping storedIn_t: Let $B = [B_X \ B_Y \ B_{\text{cur}}]$ such that $B_X \in \mathbb{Q}^{l \times k}$, $B_Y \in \mathbb{Q}^{l \times l}$ and $B_{\text{cur}} \in \mathbb{Q}^{l \times 1}$. Intuitively, B_X, B_Y, B_{cur} are divided according to control variables, data variables, and cur respectively. In addition, for $j \in [l]$, we will use $\text{row}_j(B_Y)$ to denote the j -th row of B_Y . We also define a mapping $\text{storedIn}_t : [l] \rightarrow [l] \cup \{0\}$ to record where the original value of each data variable is stored after executing t , as follows: For each $i \in [l]$,

- if there is $i' \in [l]$ such that $(\text{row}_{i'}(B_Y))(i) = 1$ (intuitively, the original value of y_i is stored in $y_{i'}$ after executing t), then $\text{storedIn}_t(i) = i'$,
- otherwise, $\text{storedIn}_t(i) = 0$.

Note that because of the copyless constraint, for each $i \in [l]$, there is at most one $i' \in [l]$ such that $(\text{row}_{i'}(B_Y))(i) = 1$, thus the mapping storedIn_t is well-defined.

π is compatible with t : We say that π are *compatible* with t , if the following constraints are satisfied.

- For each $i \in [l]$ such that $\pi(i) \neq 0$, it holds that $\text{storedIn}_t(i) \neq 0$.
Intuitively, if the current value of y_i will be stored in some data variable eventually, then the current value of y_i should not be lost after executing t .
- For each $i_1, i_2 \in [l]$ such that $\text{storedIn}_t(i_1) = \text{storedIn}_t(i_2) \neq 0$, it holds that $\pi(i_1) = \pi(i_2)$.
Intuitively, if the current value of y_{i_1} and y_{i_2} will be stored in the same data variable after executing t , then eventually, either both of them will be lost, or otherwise they will be stored in the same data variable.

π' is consistent with the application of t on π : We say that a mapping $\pi' : [l] \rightarrow [l] \cup \{0\}$ is *consistent with the application of t on π* if for each $i \in [l]$ such that $\text{storedIn}_t(i') = i$ for some $i' \in [l]$, we have $\pi'(i) = \pi(i')$ (intuitively, if the original value of $y_{i'}$ is stored in y_i after executing t and the original value of $y_{i'}$ will be stored in $y_{\pi(i')}$ eventually, then the value of y_i after executing t should be stored in $y_{\pi(i')}$ as well, thus $\pi'(i)$ takes the value of $\pi(i')$),

For each transition $t = (p, \varphi) \rightarrow (q, A, B, \vec{b})$ in δ and each $(p, q_f, \eta_1, \eta_2, \pi) \in S$ such that π is *compatible with t* , we will define a set of transitions in Δ .

In the following, let $\zeta(q_f) = \vec{a} \cdot \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} + a'$, and the reassignments of the data variables in t be $y_j := (\text{row}_j(B_Y))^t \cdot \vec{y} + f_j(\vec{x}, \text{cur})$.

In addition, let $\vec{f}(\vec{x}, \text{cur}) = (f_1(\vec{x}, \text{cur}), \dots, f_l(\vec{x}, \text{cur}))$.

Let $[z'_1], \dots, [z'_{k'}]$ be an enumeration of the equivalence classes of \simeq_p on X such that $z'_1 \prec_p \dots \prec_p z'_{k'}$. Then $\varphi(\vec{x}, \text{cur})$ is of one of the following forms, $\text{cur} = z'_i$ for $i \in [k']$, $\text{cur} \leq z'_1$, $z'_i \leq \text{cur} \leq z'_{i+1}$ for $i \in [k' - 1]$, and $z'_{k'} \leq \text{cur}$.

We make the following conventions below.

- Let η'_1 denote a specification that is consistent with the application of A on η_1 , where a specification η'_1 is *consistent with the application of A on η_1* if for each $j, j' \in [k]$ such that $A(j, j') = 1$, we have $\eta'_1(x_j) = \eta_1(x_{j'})$. Note that this definition does not take the assignments of cur to control variables into consideration.
- Similarly, let η'_2 denote a specification that is consistent with the application of A on η_2 .
- In addition, let $\pi' : [l] \rightarrow [l] \cup \{0\}$ denote a mapping that is consistent with the application of t on π .

We will define the transitions of Δ according to the different forms of φ .

[T1] If φ is of the form $\text{cur} = z'_i$ for $i \in [k']$, then

$$((p, q_f, \eta_1, \eta_2, \pi), (c'_1, c'_2), (q, q_f, \eta'_1, \eta'_2, \pi')) \in \Delta$$

where $c'_1 = \sum_{\pi'(j) \neq 0} \vec{a}(\pi'(j)) \cdot c_{1,j}$ and $c'_2 = \sum_{\pi'(j) \neq 0} \vec{a}(\pi'(j)) \cdot c_{2,j}$ such that for each $j = 1, \dots, l$:

$$\begin{aligned} c_{1,j} &= f_j(\eta_1(\vec{x}), \eta_1(z'_i)), \\ c_{2,j} &= f_j(\eta_2(\vec{x}), \eta_2(z'_i)). \end{aligned}$$

Intuitively, $c_{1,j}$ represents the value added to y_j by t , in addition, if $\pi'(j) \neq 0$, then this value will be stored in $y_{\pi'(j)}$ eventually, thus $\bar{a}(\pi'(j)) \cdot c_{1,j}$ will be added to the final output. Similarly for $c_{2,j}$.

[T2] If φ is of the form $\text{cur} \leq z'_1$, then the following transitions are in Δ :

- $((p, q_f, \eta_1, \eta_2, \pi), (c'_{1,low}, c'_{2,low}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = -\infty$ and $\eta'_2(x_j) = -\infty$,
- $((q, \eta_1, \eta_2), (c'_{1,low}, c'_{2,up}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = -\infty$ and $\eta'_2(x_j) = \eta_2(z'_1)$,
- $((q, \eta_1, \eta_2), (c'_{1,up}, c'_{2,low}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = \eta_1(z'_1)$ and $\eta'_2(x_j) = -\infty$,
- $((q, \eta_1, \eta_2), (c'_{1,up}, c'_{2,up}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = \eta_1(z'_1)$ and $\eta'_2(x_j) = \eta_2(z'_1)$,

where

- $c'_{1,low} = \sum_{\pi'(j) \neq 0} \bar{a}(\pi'(j)) \cdot \bar{c}_{1,low}(j)$, and $c'_{2,low} = \sum_{\pi'(j) \neq 0} \bar{a}(\pi'(j)) \cdot \bar{c}_{2,low}(j)$,
- $c'_{1,up} = \sum_{\pi'(j) \neq 0} \bar{a}(\pi'(j)) \cdot \bar{c}_{1,up}(j)$, and $c'_{2,up} = \sum_{\pi'(j) \neq 0} \bar{a}(\pi'(j)) \cdot \bar{c}_{2,up}(j)$,
- $\bar{c}_{1,low}$ and $\bar{c}_{2,low}$ denote the vectors $\vec{f}(\vec{x}, \text{cur})$ with (\vec{x}, cur) being substituted with $(\eta_1(\vec{x}), -\infty)$ and $(\eta_2(\vec{x}), -\infty)$ respectively,
- $\bar{c}_{1,up}$ and $\bar{c}_{2,up}$ denote the vectors $\vec{f}(\vec{x}, \text{cur})$ with (\vec{x}, cur) being substituted with $(\eta_1(\vec{x}), \eta_1(z'_1))$ and $(\eta_2(\vec{x}), \eta_2(z'_1))$ respectively.

Above, when we substitute cur with $-\infty$ inside $\vec{f}(\vec{x}, \text{cur})$, we do not evaluate the $\pm\infty$ and take them as two special variables. For example, we may have expression $3x_1 - 2\text{cur}$. Under the substitution $(x_1, \text{cur}) \mapsto (-\infty, -\infty)$, we obtain a $3(-\infty) - 2(-\infty) = (3-2)(-\infty) = -\infty$. Intuitively, $+\infty$ and $-\infty$ are to be interpreted as arbitrary rational numbers bigger and smaller than c_{\max} and c_{\min} respectively. In Appendix O, we will show that the reachability problem for \mathbb{Q} -VASS that contain special symbols $\pm\infty$ can also be reduced to the satisfiability of existential Presburger formula, similar to the normal \mathbb{Q} -VASS (without $\pm\infty$).

[T3] If φ is of the form $z'_i \leq \text{cur} \leq z'_{i+1}$ for some $i \in [k' - 1]$, then the following transitions are in Δ :

- $((p, q_f, \eta_1, \eta_2, \pi), (c'_{1,low}, c'_{2,low}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = \eta_1(z'_i)$ and $\eta'_2(x_j) = \eta_2(z'_i)$,
- $((q, \eta_1, \eta_2), (c'_{1,low}, c'_{2,up}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = \eta_1(z'_i)$ and $\eta'_2(x_j) = \eta_2(z'_{i+1})$,
- $((q, \eta_1, \eta_2), (c'_{1,up}, c'_{2,low}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = \eta_1(z'_{i+1})$ and $\eta'_2(x_j) = \eta_2(z'_i)$,
- $((q, \eta_1, \eta_2), (c'_{1,up}, c'_{2,up}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = \eta_1(z'_{i+1})$ and $\eta'_2(x_j) = \eta_2(z'_{i+1})$,

where

- $c'_{1,low} = \sum_{\pi'(j) \neq 0} \bar{a}(\pi'(j)) \cdot \bar{c}_{1,low}(j)$, $c'_{2,low} = \sum_{\pi'(j) \neq 0} \bar{a}(\pi'(j)) \cdot \bar{c}_{2,low}(j)$,
- and $c'_{1,up} = \sum_{\pi'(j) \neq 0} \bar{a}(\pi'(j)) \cdot \bar{c}_{1,up}(j)$, and $c'_{2,up} = \sum_{\pi'(j) \neq 0} \bar{a}(\pi'(j)) \cdot \bar{c}_{2,up}(j)$,
- $\bar{c}_{1,low}$ and $\bar{c}_{2,low}$ denote the vectors $\vec{f}(\vec{x}, \text{cur})$ with (\vec{x}, cur) being substituted with $(\eta_1(\vec{x}), \eta_1(z'_i))$ and $(\eta_2(\vec{x}), \eta_2(z'_i))$ respectively,
- $\bar{c}_{1,up}$ and $\bar{c}_{2,up}$ denote the vectors $\vec{f}(\vec{x}, \text{cur})$ with (\vec{x}, cur) being substituted with $(\eta_1(\vec{x}), \eta_1(z'_{i+1}))$ and $(\eta_2(\vec{x}), \eta_2(z'_{i+1}))$ respectively.

[T4] If φ is of the form $z'_{k'} \leq \text{cur}$, then the following transitions are in Δ :

- $((p, q_f, \eta_1, \eta_2, \pi), (c'_{1,low}, c'_{2,low}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = \eta_1(z'_{k'})$ and $\eta'_2(x_j) = \eta_2(z'_{k'})$,
- $((q, \eta_1, \eta_2), (c'_{1,low}, c'_{2,up}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = \eta_1(z'_{k'})$ and $\eta'_2(x_j) = +\infty$,
- $((q, \eta_1, \eta_2), (c'_{1,up}, c'_{2,low}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = +\infty$ and $\eta'_2(x_j) = \eta_1(z'_{k'})$,
- $((q, \eta_1, \eta_2), (c'_{1,up}, c'_{2,up}), (q, q_f, \eta'_1, \eta'_2, \pi'))$, such that for each $j \in [k]$ with $A(j, k+1) = 1$, we have $\eta'_1(x_j) = +\infty$ and $\eta'_2(x_j) = +\infty$,

where

- $c'_{1,low} = \sum_{\pi'(j) \neq 0} \vec{a}(\pi'(j)) \cdot \vec{c}_{1,low}(j)$, and $c'_{2,low} = \sum_{\pi'(j) \neq 0} \vec{a}(\pi'(j)) \cdot \vec{c}_{2,low}(j)$,
- $c'_{1,up} = \sum_{\pi'(j) \neq 0} \vec{a}(\pi'(j)) \cdot \vec{c}_{1,up}(j)$, and $c'_{2,up} = \sum_{\pi'(j) \neq 0} \vec{a}(\pi'(j)) \cdot \vec{c}_{2,up}(j)$,
- $\vec{c}_{1,low}$ and $\vec{c}_{2,low}$ denote the vectors $\vec{f}(\vec{x}, \text{cur})$ with (\vec{x}, cur) being substituted with $(\eta_1(\vec{x}), \eta_1(z'_{k'}))$ and $(\eta_2(\vec{x}), \eta_1(z'_{k'}))$ respectively,
- $\vec{c}_{1,up}$ and $\vec{c}_{2,up}$ denote the vectors $\vec{f}(\vec{x}, \text{cur})$ with (\vec{x}, cur) being substituted with $(\eta_1(\vec{x}), +\infty)$ and $(\eta_2(\vec{x}), +\infty)$ respectively.

[T5] Finally, Δ includes the following transitions involving q'_0 and q'_f .

- For each $q_f \in F$ and $\pi : [l] \rightarrow [l] \cup \{0\}$ such that $\zeta(q_f) = \vec{a} \cdot \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} + a'$, the transition $(q'_0, (c_1, c_2), (q_0, q_f, \eta, \eta, \pi)) \in \Delta$, where $c_1 = \sum_{\pi(j) \neq 0} \vec{a}(\pi(j)) \cdot \vec{u}_0(Y)(j)$, $c_2 = \sum_{\pi(j) \neq 0} \vec{a}(\pi(j)) \cdot \vec{u}_0(Y)(j)$, and $\eta(x_i) = \vec{u}_0(X)(i)$ for each $x_i \in X$.
- For each state $(q_f, q_f, \eta_1, \eta_2, \pi) \in S$ such that $\pi(j) = j$ for each $j \in [l]$, suppose $\zeta(q_f) = \vec{a} \cdot \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} + a'$, then the transition $((q_f, q_f, \eta_1, \eta_2, \pi), (c_1, c_2), q'_f) \in \Delta$, where $c_1 = a' + \vec{a}(X) \cdot \eta_1(\vec{x})$ and $c_2 = a' + \vec{a}(X) \cdot \eta_2(\vec{x})$. The requirement that $\pi(j) = j$ for each $j \in [l]$ is consistent with the intuition of π : When reaching the final state q_f , for each $j \in [l]$, the current value of y_j is stored in $y_{j=\pi(j)}$. Note that the contents of $y_j \in Y$ have been added to the final output, this is why c_1, c_2 only need take the contents of \vec{x} into consideration.

Then there is w such that $0 \in \mathcal{A}(w)$ iff there is a configuration (q'_f, c'_1, c'_2) reachable from $(q'_0, 0, 0)$ in \mathcal{B} such that either $c'_1 \leq 0 \leq c'_2$ or $c'_2 \leq 0 \leq c'_1$.

Therefore, the reachability problem for the $\text{RA}_{\mathbb{Q}}$ \mathcal{A} is reduced to the configuration coverability problem for \mathbb{Q} -VASS \mathcal{B} (via a Karp reduction), which in turn, can be reduced to satisfiability problem for existential Presburger formula. See Appendix O for the details.

Complexity analysis: Suppose a non-normalized $\text{RA}_{\mathbb{Q}}$ \mathcal{A} is given as the input, with n states, k control variables, and l data variables respectively. Then as mentioned before, the number of states of the normalized $\text{RA}_{\mathbb{Q}}$ \mathcal{A}' is at most $n2^{2k-1}(2k)!$. The number of specifications η is at most k^{2k} (at most $2k$ control variables and at most k constants in \mathcal{A}'), while the number of mappings π is at most $(l+1)^l$. Since each state from S is of the form $(q, q_f, \eta_1, \eta_2, \pi)$ (except two special states), it follows that the cardinality of S is at most $(n2^{2k-1}(2k)!)^2(k^{2k})^2(l+1)^l$, which is exponential over the size of \mathcal{A} . Because the reachability problem for \mathbb{Q} -VASS is in NP (cf. Theorem 11 in Appendix O), we conclude that the reachability problem for copyless $\text{RA}_{\mathbb{Q}}$ is in NEXPTIME.

O. Rational VASS (\mathbb{Q} -VASS), with or without $\pm\infty$

We will use the following theorem.

Theorem 9. [47] *For every finite state automaton \mathcal{A} over the alphabet $\{\alpha_1, \dots, \alpha_k\}$, one can construct in polynomial time an existential Presburger formula $\Psi_{\mathcal{A}}(x_1, \dots, x_k)$ such that for every $(a_1, \dots, a_k) \in \mathbb{N}^k$, $\Psi_{\mathcal{A}}(a_1, \dots, a_k)$ holds if and only if there is a word $w \in L(\mathcal{A})$ with Parikh image (a_1, \dots, a_k) .*

In fact, Theorem 9 holds also for context-free grammar, but for our purpose, finite state automata is sufficient. One can easily modify it so that instead of Parikh image, it talks about the number of transitions in an accepting run, as stated below.

Theorem 10. [47] *For every finite state automaton \mathcal{A} over transitions $\{t_1, \dots, t_m\}$, one can construct in polynomial time an existential Presburger formula $\Psi_{\mathcal{A}}(x_1, \dots, x_m)$ such that for every $(a_1, \dots, a_m) \in \mathbb{N}^m$, $\Psi_{\mathcal{A}}(a_1, \dots, a_m)$ holds if and only if there is an accepting run of \mathcal{A} in which transition t_i appears a_i times.*

Indeed, Theorem 10 is a straightforward consequence of Theorem 9 by labelling the transitions with new labels, so that all of them have different labels.

Rational VASS (\mathbb{Q} -VASS): A k -dimensional \mathbb{Q} -VASS is a pair (S, Δ) , where S is a finite set of states and Δ is a finite subset of $S \times \mathbb{Q}^k \times S$. A configuration is a pair $(s, \vec{u}) \in S \times \mathbb{Q}^k$. A configuration (r, \vec{v}) is reachable from (s, \vec{u}) , if there is a sequence of transitions $(s_0, \vec{v}_0, s_1), (s_1, \vec{v}_1, s_2), \dots, (s_n, \vec{v}_n, s_{n+1})$ of Δ such that $s_0 = s$, $s_{n+1} = r$ and $\vec{v} = \vec{u} + \sum_{i=0}^n \vec{v}_i$. We say that (r, \vec{v}) is coverable by (s, \vec{u}) , if $\vec{v} \leq \vec{u} + \sum_{i=0}^n \vec{v}_i$.

Two popular problems for \mathbb{Q} -VASS are:

- **\mathbb{Q} -VASS configuration reachability:**

On input \mathbb{Q} -VASS (S, Δ) and two configurations (s, \vec{v}) and (r, \vec{u}) , decided whether (s, \vec{v}) is reachable from (r, \vec{u}) .

- **\mathbb{Q} -VASS configuration coverability:**

On input \mathbb{Q} -VASS (S, Δ) and two configurations (s, \vec{v}) and (r, \vec{u}) , decided whether (s, \vec{v}) is coverable from (r, \vec{u}) .

The two problems above are special cases of the following problem:

- **\mathbb{Q} -VASS strong configuration reachability:**

On input k -dimensional \mathbb{Q} -VASS (S, Δ) , a configuration (r, \vec{u}) , a state s and a Boolean combination of atomic Presburger formula $\varphi(x_1, \dots, x_k)$, decide whether there is a configuration (s, \vec{v}) such that $\varphi(\vec{v})$ holds and (s, \vec{v}) is reachable from (r, \vec{u}) .

Note that reachability and coverability are special cases of strong reachability, where $\varphi(z_1, \dots, z_m)$ is $(z_1, \dots, z_m) = \vec{v}$ and $(z_1, \dots, z_m) \geq \vec{v}$, respectively.

Theorem 11. \mathbb{Q} -VASS strong configuration reachability is in NP.

Proof. Let the input be (S, Δ) , (r, \vec{u}) , s and $\varphi(x_1, \dots, x_k)$ as above. Let $\Delta = \{(s_1, \vec{v}_1, t_1), \dots, (s_m, \vec{v}_m, t_m)\}$. Deciding strong reachability is equivalent to deciding the satisfiability of the following formula:

$$\Phi := \exists x_1 \cdots \exists x_m \Psi_{\mathcal{A}}(x_1, \dots, x_m) \wedge \varphi\left(\vec{u} + \sum_{i=1}^m x_i \vec{v}_i\right)$$

where $\Psi_{\mathcal{A}}(x_1, \dots, x_m)$ is the formula constructed via Theorem 10, by viewing (S, Δ) as a finite state automaton, r the initial state and s the final state. Note that each \vec{v}_i may contain rational numbers. However, we can get rid of the denominators easily by scalar multiplication with integers, which only increases the size of the formula polynomially (since the constants are encoded in binary). Then the satisfiability for Φ is reduced to integer linear programming problem, which is well-known to be NP-complete. \square

The conversion of \mathbb{Q} -VASS with $\pm\infty$ to existential Presburger formula: Let the input be (S, Δ) , (r, \vec{u}) , s and $\varphi(x_1, \dots, x_k)$ as above. Let $\Delta = \{(s_1, \vec{v}_1, t_1), \dots, (s_m, \vec{v}_m, t_m)\}$ such that $\vec{v}_1, \dots, \vec{v}_m$ may include the special symbols $\pm\infty$. In addition, suppose we know that $+\infty$ and $-\infty$ represent an arbitrary rational number bigger and smaller than c_{\max} and c_{\min} respectively. Let Φ be the formula constructed in the proof of Theorem 11, that is,

$$\Phi := \exists x_1 \cdots \exists x_m \Psi_{\mathcal{A}}(x_1, \dots, x_m) \wedge \varphi\left(\vec{u} + \sum_{i=1}^m x_i \vec{v}_i\right).$$

Then we can transform Φ into a formula Φ' without $\pm\infty$ as follows.

- For term $x_i(+\infty)$, we replace it with a fresh existential variable z and add a conjunct $z \geq c_{\max}x_i$.
- For term $x_i(-\infty)$, we replace it with a fresh new existential variable z and add a conjunct $z \leq c_{\min}x_i$.
- Likewise for terms such as $a(+\infty)$ and $a(-\infty)$.

P. Results on the coverability problem

Theorem 12. *The invariant problem of $\text{RA}_{\mathbb{Q}}$ can be reduced to the coverability problem of $\text{RA}_{\mathbb{Q}}$, which can be further reduced to the reachability problem of $\text{RA}_{\mathbb{Q}}$. Both reductions are in polynomial-time.*

Let \mathcal{A} be an $\text{RA}_{\mathbb{Q}}$. The first reduction is done by creating (1) an $\text{RA}_{\mathbb{Q}}$ \mathcal{A}' such that $\exists v \in \mathcal{A}(w) : v \geq 0$ for some w iff $\exists v \in \mathcal{A}'(w) : v > 0$ for some w by adding an arbitrary positive value to the output of \mathcal{A} , e.g., add a $\text{cur} > 0$ and (2) another $\text{RA}_{\mathbb{Q}}$ \mathcal{A}'' by negating all output expressions of \mathcal{A}' . The answer to the non-zero problem of \mathcal{A} is positive iff the answers to the coverability problems of \mathcal{A}' and \mathcal{A}'' are both positive. The second reduction is done by adding a transition $(q, \text{cur} \geq 0) \rightarrow (q', y_1 := \zeta(q) - \text{cur})$ from all final states q to a new state q' for some data variable y_1 and setting q' as the only new final state with the output expression y .

Corollary 3. *The coverability problem of copyless $\text{RA}_{\mathbb{Q}}$ s is in NEXPTIME.*

Q. Results on configuration reachability and coverability

For Petri-net or VASS, people are also interested in configuration reachability and configuration coverability problems. The corresponding problems in $\text{RA}_{\mathbb{Q}}$ are defined as follows. Given an $\text{RA}_{\mathbb{Q}}$ $\mathcal{A} = \langle Q, q_0, F, \vec{u}_0, \delta, \zeta \rangle$ and a configuration (q_n, \vec{u}_n) , the configuration reachability problem asks if $(q_0, \vec{u}_0) \vdash^* (q_n, \vec{u}_n)$ and the configuration coverability problem asks if $(q_0, \vec{u}_0) \vdash^* (q_n, \vec{u}'_n)$ and $\vec{u}'_n \geq \vec{u}_n$. We show that the two problems in $\text{RA}_{\mathbb{Q}}$ are inter-reducible in polynomial-time and they are not easier than the reachability problem, which is undecidable (Theorem 7).

Lemma 5. *The configuration reachability problem of $\text{RA}_{\mathbb{Q}}$ can be reduced to the configuration coverability problem of $\text{RA}_{\mathbb{Q}}$, and vice versa.*

Let \mathcal{A} be an $\text{RA}_{\mathbb{Q}}$ over (X, Y) and the two problem are targeting the configuration (q, \vec{v}) . The first reduction is done by creating a $\text{RA}_{\mathbb{Q}}$ \mathcal{A}' over $(X \cup X', Y \cup Y')$ such that X' and Y' compute the negation of X and Y , using a similar construction of the proof of Lemma 2 in [32]. The reverse direction is done by adding the transition $(q, \bigwedge_{x_i \in X} x = \vec{v}(X)[i]) \rightarrow (q', \{\})$ and for all $y \in Y$ the transition $(q', \text{cur} \geq 0) \rightarrow (q', \{y := y - \text{cur}\})$ and targeting the configuration (q', \vec{v}) instead.

Lemma 6. *The coverability and reachability problems of $RA_{\mathbb{Q}}$ can be reduced to the configuration coverability and reachability problems of $RA_{\mathbb{Q}}$, respectively.*

The reduction is done by adding a transition with the assignment $y := \zeta(q)$ from all final states q to a new state q' and use the configuration $(q', 0)$ in the corresponding configuration coverability and reachability problems.