Decision Problems for Finite Automata over Infinite Algebraic Structures

Bakhadyr Khoussainov(✉) and Jiamou Liu

Department of Computer Science, University of Auckland, Auckland, New Zealand
bmk@cs.auckland.ac.nz, jiamou.liu@auckland.ac.nz

Abstract. We introduce the concept of finite automata over algebraic structures. We address the classical emptiness problem and its various refinements in our setting. In particular, we prove several decidability and undecidability results. We also explain the way our automata model connects with the existential first order theory of algebraic structures.

1 Introduction

Most computer programs rely on operations and queries on a priori defined data types. An example of a such data type is the integers with the usual operations of addition, multiplication and the comparison test. Another example is the graph data type that encapsulates the operations of adding or deleting vertices and edges as well as edge and subgraph queries. Algebraically, data types are structures of the form $(D; f_1, \ldots, f_m, P_1, \ldots, P_n, \overline{c_1}, \ldots, \overline{c_\ell})$ where $f_1, \ldots, f_m$ are atomic operations, $P_1, \ldots, P_n$ are atomic relations and $\overline{c_1}, \ldots, \overline{c_\ell}$ are constants on the domain D. A program can thus be viewed as a sequence of operations and queries over a certain algebraic structure.

This view of programs motivated the definitions of many models of computation over structures. An example is the Blum-Shub-Smale (BSS) machines, where the underlying structure is the ordered ring of the reals [2]. This model builds a theoretical foundation of numerical algorithms on reals. Another example is the work of O. Bournez, et al. [5] that generalises BSS machines to models over arbitrary structures. Among several results, they prove that the set of all recursive functions over arbitrary structure $\mathcal{S}$ is exactly the set of decision functions computed by BSS machines over $\mathcal{S}$. Other examples include various classes of counter automata that use counters in different ways [6,9,12,15,17].

In this work we introduce finite automata models over algebraic structures. The work fits into two lines of research: Firstly, one may view our models as finite automata analogues of BSS machines over arbitrary structures. These automata process finite sequences of elements from the domain of a given structure $\mathcal{S}$, and accept or reject these sequences. Such an automaton is a finite state machine equipped with a fixed number of registers, a read only head that always moves to the right in the tape and transitions between the states. For the structure $\mathcal{S}$, we use the term $\mathcal{S}$-*automata* or *extended* $\mathcal{S}$-*automata* that define two different interpretations of this computation model on $\mathcal{S}$. Secondly, one may view our

Y.-S. Han and K. Salomaa (Eds.): CIAA 2016, LNCS 9705, pp. 3–11, 2016.
DOI: 10.1007/978-3-319-40946-7_1

automata models as automata over an infinite alphabet when the underlying structure $\mathcal{S}$ is infinite. Automata over infinite alphabet attracted considerable interests due to connections to model checking and verification [1,3,4,10,21,23]. One goal here is to extend automata-theoretic techniques to words and trees over data values. For example, Kaminsky and Francez in [13] proposed register automata. These are finite state machines equipped with a fixed number of registers which may hold values from an infinite domain D. The operations allowed by the automata are equality comparisons between the input and the register values and the copy operation. Another example is pebble automata introduced by Neven, Schwentick and Vianu [20]. Their automata use a fixed set of pebbles with a stack to keep track of values in the input data words. Operations include equality comparisons of the current pebble values, and dropping and lifting a pebble. Other examples of such automata models include Bojanczyk's data automata [3] and Alur's extended data automata [1]. While all the above automata allow only equality tests between data values, there has also been automata model for linearly ordered data domains [22]. The existence of many such models of automata calls for a general yet simple framework to formally reason about such finite state automata. We suggest one such framework.

2 Two Automata Models Over Algebraic Structures

Let $\mathcal{S} = (D; f_1, \ldots, f_m, P_1, \ldots, P_n, \overline{c_1}, \ldots, \overline{c_\ell})$ be an algebraic structure. We assume that $m = n$ and that all functions and predicate are binary. These are not restrictions. For instance, when $n < m$ we expand the structure by adding the relation P_n to its signature to ensure that in the expanded structure the number of atomic predicates and atomic operations are equal.

A *D-word* of length t is a sequence $d_1 \ldots d_t$ of elements of D. An automaton over $\mathcal{S}$ has k *updatable registers* $R_1, \ldots, R_k$; each register R_i contains an n-tuple $\overline{r_i} = (r_{i,1}, \ldots, r_{i,n})$ from D^n, and the content of the register might change. Furthermore, the automaton has ℓ *constant registers* containing the constants $\overline{c_1}, \ldots, \overline{c_\ell}$; these values never change. We represent these values of registers as matrices $\mathbf{R}$ and $\mathbf{C}$.

$$\mathbf{R} = \begin{pmatrix} r_{1,1} & \ldots & r_{1,n} \\ r_{2,1} & \ldots & r_{2,n} \\ \ldots & \ldots & \ldots \\ r_{k,1} & \ldots & r_{k,n} \end{pmatrix} \quad \text{and} \quad \mathbf{C} = \begin{pmatrix} c_{1,1} & \ldots & c_{1,n} \\ c_{2,1} & \ldots & c_{2,n} \\ \ldots & \ldots & \ldots \\ c_{\ell,1} & \ldots & c_{\ell,n} \end{pmatrix}$$

Let Op be the set of all atomic operations of $\mathcal{S}$. The automaton is a finite state machine where transitions are of the form (q, T_1, T_2, F, q') where q, q' are states, T_1, T_2 are a pair of $\{0,1\}$-valued matrices of sizes $k \times n$ and $\ell \times n$ respectively, and $F = (f_{i,j}) \in Op^{k \times n}$ is a $k \times n$ matrix of atomic operation of $\mathcal{S}$. Inputs to the automaton are D-words written on a one-way read-only tape. When the automaton is in state q and reads the next element x of an input D-word, the automaton proceeds with two steps:

1. (**Testing**) The automaton produces two $k \times n$ and $\ell \times n$ test matrices $Test(\mathbf{R}, x)$ and $Test(\mathbf{C}, x)$ with entries 1 (true) or 0 (false), respectively:

$$\begin{pmatrix} P_1(r_{1,1}, x) & \dots & P_n(r_{1,n}, x) \\ P_1(r_{2,1}, x) & \dots & P_n(r_{2,n}, x) \\ \dots & \dots & \dots \\ P_1(r_{k,1}, x) & \dots & P_n(r_{k,n}, x) \end{pmatrix}, \quad \begin{pmatrix} P_1(c_{1,1}, x) & \dots & P_n(c_{1,n}, x) \\ P_1(c_{2,1}, x) & \dots & P_n(c_{2,n}, x) \\ \dots & \dots & \dots \\ P_1(c_{\ell,1}, x) & \dots & P_n(c_{\ell,n}, x) \end{pmatrix}$$

   The automaton then makes a transition (q, T_1, T_2, F, q') where q is the current state, $T_1 = Test(\mathbf{R}, x)$ and $T_2 = Test(\mathbf{C}, x)$.
2. (**Updating**) When making the transition (q, T_1, T_2, F, q'), the automaton updates the values of registers using operations in F, transforming the matrix $\mathbf{R}$ to the matrix $F(\mathbf{R}, x)$ as presented below:

$$\mathbf{R} = \begin{pmatrix} r_{1,1} & \dots & r_{1,n} \\ r_{2,1} & \dots & r_{2,n} \\ \dots & \dots & \dots \\ r_{k,1} & \dots & r_{k,n} \end{pmatrix} \quad \Longrightarrow \quad F(\mathbf{R}, x) = \begin{pmatrix} f_{1,1}(r_{1,1}, x) & \dots & f_{1,n}(r_{1,n}, x) \\ f_{2,1}(r_{2,1}, x) & \dots & f_{2,n}(r_{2,n}, x) \\ \dots & \dots & \dots \\ f_{k,1}(r_{k,1}, x) & \dots & f_{k,n}(r_{k,n}, x) \end{pmatrix}$$

   where $f_{i,j}$ the (i, j)-entry of F for all $1 \leq i \leq k, 1 \leq j \leq n$.

After all elements on the input tape have been read, the $\mathcal{S}$-automaton stops and decides whether to accept the input depending on the last state.

In the following we put constraints on the register values $\mathbf{R}$ and the operation matrix F and introduce two finite automata models over the structure $\mathcal{S}$, namely, the $\mathcal{S}$-automata and extended $\mathcal{S}$-automata.

- **$\mathcal{S}$-automata:** We require the matrix F in each transition to be the same; furthermore, each row in F is the tuple $(f_1, \dots, f_n) \in Op^k$ of all atomic operation of $\mathcal{S}$. Hence any transition will transform the ith register $R_i = (r_{i,1}, \dots, r_{i,n})$ to $(f_1(r_{i,1}, x), \dots, f_n(r_{i,n}, x))$. Thus, a transition of an $\mathcal{S}$-automaton can simply be represented as (q, T_1, T_2, q').
- **Extended $\mathcal{S}$-automata:** We require the columns in both the register matrix $\mathbf{R}$ and the operation matrix F for each transition to be the same, that is:

$$\mathbf{R} = \begin{pmatrix} r_1 & \dots & r_1 \\ r_2 & \dots & r_2 \\ \dots & \dots & \dots \\ r_k & \dots & r_k \end{pmatrix} \text{ and } F = \begin{pmatrix} f_{i_1} & \dots & f_{i_1} \\ f_{i_2} & \dots & f_{i_2} \\ \dots & \dots & \dots \\ f_{i_k} & \dots & f_{i_k} \end{pmatrix}$$

  Thus we can simply write $\mathbf{R}$ as a tuple of elements $(r_1, r_2, \dots, r_k) \in D^k$ and F as a tuple $(f_{i_1}, \dots, f_{i_k}) \in Op^k$.

Definition 1. *We define two types of automata:*

1. *An* $(\mathcal{S}, k)$-automaton *is a tuple* $\mathcal{A} = (Q, \mathbf{R}, \Delta, I, F)$ *where* Q *is a finite set of states,* $\mathbf{R} = (\bar{r}_1, \dots, \bar{r}_k)$ *is the initial values of the registers with each* $\bar{r}_i \in D^n$, $I \subseteq Q$ *is the initial states set,* $F \subseteq Q$ *is the set of accepting states and* $\Delta \subseteq Q \times \{0,1\}^{n \cdot (k+\ell)} \times Q$ *is the transition relation of* $\mathcal{A}$. *The automaton is* deterministic *if* Δ *determines the function* $\Delta : Q \times \{0,1\}^{n \cdot (k+\ell)} \to Q$. *An* $\mathcal{S}$-automaton *is an* $(\mathcal{S}, k)$*-automaton for some* $k \in \omega$.

2. *An* extended $(\mathcal{S}, k)$-automaton *is defined in the same way as an* $(\mathcal{S}, k)$-*automaton, with the following exceptions: the register* $\mathbf{R}$ *is* $(r_1, \ldots, r_k) \in D^k$, *and* $\Delta \subseteq Q \times \{0,1\}^{(k+\ell)\cdot n} \times Op^k \times Q$ *is the transition relation. The automaton is* deterministic *if* Δ *is a function* $\Delta : Q \times \{0,1\}^{(k+\ell)\cdot n} \to Op^k \times Q$. *An* extended $\mathcal{S}$-automaton *is an extended* $(\mathcal{S}, k)$-*automaton for some* $k \in \omega$.

Let $\mathcal{A} = (Q, \mathbf{R}, \Delta, I, F)$ be an (extended) $(\mathcal{S}, k)$-automaton. We define the runs of the automaton on D-words as follows. A *configuration* of the automaton is a pair $(\mathbf{R}, q)$, where q is a state of the automaton and $\mathbf{R}$ is the matrix of register values. A *run* of $\mathcal{A}$ on a D-word $d_0 \ldots d_t$ is a sequence of configurations

$$(\mathbf{R}_0, s_0), (\mathbf{R}_1, s_1), \ldots, (\mathbf{R}_{t+1}, s_{t+1})$$

such that where $\mathbf{R}_0 = \mathbf{R}$, $s_0 \in I$, the transition from s_i to s_{i+1} is labeled with the test matrices $Test(\mathbf{R}_i, d_i)$ and $Test(\mathbf{C}, d_i)$, and $\mathbf{R}_{i+1} = F(\mathbf{R}_i, d_i)$ for all i. The run is *accepting* if $s_{t+1} \in F$. We say that $\mathcal{A}$ *accepts* the D-word $d_0 \ldots d_t$ if $\mathcal{A}$ has an accepting run on $d_0 \ldots d_t$.

Definition 2. *The* language $L(\mathcal{A})$ *of the (extended)* $\mathcal{S}$*-automaton* $\mathcal{A}$ *is the set of all* D*-words accepted by* $\mathcal{A}$. *We call such languages* (extended) $\mathcal{S}$-regular.

3 Decision Problems on $\mathcal{S}$-Automata

Simple Properties of $\mathcal{S}$-regular Languages. The class of $\mathcal{S}$-regular languages is a natural generalisation from regular languages in the following sense. Firstly, there is a natural connection between $\mathcal{S}$-regular languages and regular languages. In particular, when the structure $\mathcal{S}$ is finite, then any $\mathcal{S}$-regular language is regular. Secondly, the class of $\mathcal{S}$-regular languages is closed under the Boolean operations. Thirdly, every $\mathcal{S}$-regular language can be recognised by a deterministic $\mathcal{S}$-automaton. Furthermore, the class of $(\mathcal{S}, k)$-recognisable languages can be a proper subclass of the class of $(\mathcal{S}, k+1)$-recognisable languages. This is true for the infinite structure $\mathcal{S} = (D; =, \mathrm{pr}_1)$ where pr_1 is the projection: $\mathrm{pr}_1(x, y) = x$.

The Emptiness Problem for $\mathcal{S}$-automata. The emptiness problem asks for an algorithm that given an $\mathcal{S}$-automaton, detects if the language of the automaton is non-empty. This problem has a positive solution for regular languages and thus is decidable when $\mathcal{S}$ is finite. It turns out that for certain large class of structures $\mathcal{S}$, the emptiness problem is decidable.

Definition 3. *An equivalence relation* $\equiv_k$ *on the set* $M_{\mathcal{S}}(n, k)$ *of matrices is called* smooth *if the relation satisfies the following conditions:*

1. *The relation* $\equiv_k$ *is of finite index.*
2. *For all* $\mathbf{R}, \mathbf{R}' \in M_{\mathcal{S}}(n, k)$, *matrices* X *and* Y *with* 0, 1 *entries, if* $\mathbf{R} \equiv_k \mathbf{R}'$ *then we have* $\{z \mid Test(\mathbf{R}, z) = X \ \&\ Test(\mathbf{C}, z) = Y\}$ *is the empty set if and only if* $\{z \mid Test(\mathbf{R}', z) = X \ \&\ Test(\mathbf{C}, z) = Y\}$ *is the empty set.*

As a simple example, assume $\equiv$ is an equivalence relation of finite index on the domain D such that all atomic predicates and operations are compatible with $\equiv$. This relation naturally defines the relation $\equiv_k$ on the matrices $M_{\mathcal{S}}(n,k)$: Two matrices are $\equiv_k$-equivalent if the entries at the same positions of the matrices are $\equiv$-equivalent. Then the equivalence relation $\equiv_k$ is smooth.

Here is another example. Let $\mathcal{S}$ be a structure $(D; f_1, \ldots, f_n, =)$. On the set $M_{\mathcal{S}}(n,k)$ consider the following relation $\equiv_k$: Two matrices $\mathbf{R}$ and $\mathbf{R}$' are $\equiv_k$-equivalent if for all two positions (i,j) and (s,t) of the matrices we have $r_{i,j} = r_{s,t}$ if and only if $r'_{i,j} = r'_{s,t}$. Then the relation $\equiv_k$ is smooth.

Let $\{\equiv_k\}_{k>0}$ be a family of smooth equivalence relations on $\mathcal{S}$. Assume that for each k we can effectively represent the $\equiv_k$-classes by some finite objects. For instance, when $k = 2$ and $n = 2$, the relation $\equiv_2$ considered in the paragraph above has the following representatives:

$$\begin{pmatrix} a & a \\ a & a \end{pmatrix}, \begin{pmatrix} a & b \\ a & a \end{pmatrix}, \begin{pmatrix} a & b \\ c & a \end{pmatrix}, \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

where a, b, c, d are all pairwise distinct and fixed integers. We call these representatives *types* of the equivalence classes. With this set-up, we have:

Definition 4. *The structure $\mathcal{S}$ is* nice *if it satisfies the following two properties:*

1. *There is an algorithm that given a type of a matrix $\mathbf{R} \in M_{\mathcal{S}}(n,k)$, and given two $\{0,1\}$-valued matrices X, Y decides if the set $\{z \mid Test(\mathbf{R}, z) = X \;\&\; Test(\mathbf{C}, z) = Y\}$ is empty or not.*
2. *There is an algorithm that given a type of a matrix $\mathbf{R} \in M_{\mathcal{S}}(n,k)$, computes the types of all matrices $F(\mathbf{R}, x)$ where x satisfies the equations $Test(\mathbf{R}, z) = X$ and $Test(\mathbf{R}, z) = Y$ for given X, Y.*

In particular, let $\mathcal{S}$ be a structure $(D; f_1, \ldots, f_n, =)$. Assume that for each f_i there is a finite set $F_i \subset D$ such that

1. For every $d \notin F_i$ the function $f_{i,d}(x) = f_i(d, x)$ is injection on D.
2. For each $d \in F_i$, the function $f_{i,d}(x) = f_i(d, x)$ is a constant function, that is, there is an $a \in D$ such that $f_{i,d}(x) = a$ for all $x \in D$.

Then the smooth equivalence relation $\equiv_k$ makes the structure $\mathcal{S}$ nice. For instance, the structure $(\mathbb{Z}; +, \times, =)$ satisfies the properties above.

Theorem 5. *The emptiness problem over any nice structure is decidable. More precisely, for any nice structure $\mathcal{S}$ over domain D, there is an algorithm that, given an $\mathcal{S}$-automaton $\mathcal{A} = (Q, \mathbf{R}, \Delta, I, F)$ and the type of $\mathbf{R}$, detects if the automaton accepts at least one D-word.*

From the theorem above we immediately get the following corollary.

Corollary 6. *The emptiness problem is decidable over the arithmetic $(\mathbb{Z}; +, \times)$, the fields of reals $(\mathbb{R}; +, \times)$ and rational numbers $(\mathbb{Q}; +, \times)$.* □

4 Decision Problems on Extended $\mathcal{S}$-Automata

Simple Properties of Extended $\mathcal{S}$-regular Languages. Any $\mathcal{S}$-regular language is clearly extended $\mathcal{S}$-regular. On the other hand, extended $\mathcal{S}$-automata recognise larger class of languages. The limitation of $\mathcal{S}$-automata is that, when processing a D-word, the sequence of updates to the registers are the same regardless of which path the automaton take. In an extended $\mathcal{S}$-automaton, however, the operations performed on registers depends on the outcomes of the tests. This leads to a lack of some crucial properties enjoyed by $\mathcal{S}$-regular languages, such as determination. Furthermore, there exists extended $\mathcal{S}$-regular languages whose complements are not recognisable by any extended $\mathcal{S}$-automata. On the other hand, the class of languages recognised by deterministic extended $\mathcal{S}$-automata is closed under the Boolean operations.

Validation Problem for Extended $\mathcal{S}$-automata. We refine the emptiness problem for finite automata as follows. Ddesign an algorithm that, given an $\mathcal{S}$-automaton over the structure $\mathcal{S}$, and a path from an initial state to an accepting state in the automaton, builds an input sequence from the structure $\mathcal{S}$ that validates the path. We call this *the validation problem* for $\mathcal{S}$-automata. We will investigate the validation problem for extended $\mathcal{S}$-automata and connect the problem with the first order existential theory of the structure $\mathcal{S}$.

We postulate that $\mathcal{S}$ is a computable structure, i.e., its domain D and all of its atomic predicates P_1, ..., P_n and operations f_1, ..., f_n are computable. The validation problem for extended $\mathcal{S}$-automata turns out to be equivalent to deciding the existential theory (with parameters) $\mathsf{Th}_\exists(\mathcal{S})$ of the structure. For the next theorem, we use $\mathcal{S}[\mathrm{pr}_1, \mathrm{pr}_2]$ to denote the structure obtained from $\mathcal{S}$ upon expansion with two projection operations pr_1 and pr_2.

Theorem 7. *Suppose $\mathcal{S}$ is a computable structure. The validation problem for extended $\mathcal{S}[\mathrm{pr}_1, \mathrm{pr}_2]$-automata is decidable if and only if $\mathsf{Th}_\exists(\mathcal{S})$ is decidable.*

As a corollary, we see that the validation problem over computable structures with undecidable existential theory, such as the arithmetic $(\omega; +, \times, \leq,)$, is undecidable. Also, the validation problem over computable structures with decidable first order theory, such as the Presburger arithmetic, is decidable.

The Emptiness Problem for Extended $\mathcal{S}$-automata. On computable structures, the decidability of the emptiness problem implies decidability of the validation problem. The converse is not true. We discuss the emptiness problem on extended $\mathcal{S}$-automata for two special cases: the first case assumes that the transition graphs of the extended $\mathcal{S}$-automata are acyclic. The second concerns with fragments of the arithmetic $(\omega; +, \times, \leq, 0)$.

1. A state s is a *sink* if all outgoing transitions loop into s. All accepting (non-accepting) sink states can be collapsed into one (non-accepting) accepting sink state. Therefore we can always assume that every $\mathcal{S}$-automaton has at most 2 sink states. An extended $\mathcal{S}$-automaton *acyclic* if its state space without the sink states is an acyclic graph. If $\mathcal{S}$ is a computable structure, then the

emptiness problem of acyclic extended $\mathcal{S}[\mathrm{pr}_1, \mathrm{pr}_2]$-automata is equivalent to the corresponding validation problem. Hence, by Theorem 7, the emptiness problem is decidable for acyclic extended automata over such structures as $(\omega; +, \leq)$, $(\omega; \times, \leq)$, $(\mathbb{Q}; +, \leq)$ and finitely generated Abelian groups.

It is easy to find structures $\mathcal{S}$ with undecidable existential theory such that the emptiness problem for acyclic extended $(\mathcal{S}, k)$-automata is undecidable for every $k \geq 1$.

Let $\mathcal{S}_{\mathbb{Z}} = (\mathbb{Z}; +, \times, \mathrm{pr}_1, \mathrm{pr}_2, 0)$. One constructs, for any polynomial $p(\bar{x})$ in $\omega[x_1, \ldots, x_k]$, an acyclic extended $(\mathcal{S}_{\mathbb{Z}}, k+2)$-automaton $\mathcal{A}_p$ that evaluates p over a sequence $(a_1, \ldots, a_k) \in \mathbb{Z}^k$ of input values. This reduces Hilbert's tenth problem to the emptiness problem of acyclic extended $\mathcal{S}_{\mathbb{Z}}$-automata. Since Hilbert's tenth problem is undecidable for polynomials with bounded number of variables (the currently known bound that guarantees undecidability is 9 [16]), we obtain that the emptiness problem for acyclic $(\mathcal{S}_{\mathbb{Z}}, 11)$-automata is undecidable.

2. Let $\mathcal{S}$ be the following structure $(\omega; +1, \mathrm{pr}_1, 0)$ where $+1(x, y) = x + 1$. One constructs, given a 2-counter machine $\mathcal{M}$, an extended $\mathcal{S}$-automaton $\mathcal{M}'$ with 4 registers such that $\mathcal{M}$ accepts some word iff $\mathcal{M}'$ accepts some ω-word. This reduces the emptiness problem for 2-counter machines, known to be undecidable [17], to the emptiness problem for extended $(\mathcal{S}, 4)$-automata. Thus the emptiness problem for extended $(\mathcal{S}, 4)$-automata is undecidable.

The above shows for many structures $\mathcal{S}$, the emptiness problem for $\mathcal{S}$-automata is undecidable. We next present structures on which the emptiness problem is decidable. For this we use a tool similar to the notion of nice structures introduced for $\mathcal{S}$-automata; we recast Definition 4 in this setting:

Definition 8. *An equivalence relation $\equiv_k$ of finite index on the set D^k is* smooth *if for all $\mathbf{R}, \mathbf{R}' \in D^k$ and all $\{0,1\}$-valued matrices X, Y, the condition $\mathbf{R} \equiv_k \mathbf{R}'$ implies that the set $\{z \mid Test(\mathbf{R}, z) = X \ \&\ Test(\mathbf{C}, z) = Y\}$ is empty iff the set $\{z \mid Test(\mathbf{R}', z) = X \ \&\ Test(\mathbf{C}, z) = Y\}$ is empty.*

Definition 9. *The structure $\mathcal{S}$ is* k-nice *if we have:*

(a) There is an algorithm that given a type of a tuple $\mathbf{R} \in D^k$, and given two $\{0,1\}$-valued matrices X, Y decides if the set

$$\{z \mid Test(\mathbf{R}, z) = X \ \&\ Test(\mathbf{C}, z) = Y\}$$

is empty or not.

(b) There is an algorithm that given a type of a tuple $\mathbf{R} \in D^k$, matrices X, Y, and a tuple $F \subseteq Op^k$, computes the types of all tuples $F(\mathbf{R}, x)$ where x satisfies the equation $Test(\mathbf{R}, z) = X$ and $Test(\mathbf{R}, z) = Y$.

Theorem 10. *The emptiness problem for extended $(\mathcal{S}, k)$-automata over any k-nice structure $\mathcal{S}$ is decidable.*

Corollary 11. *The emptiness problem for extended $(\mathcal{S}, 1)$-automata is decidable for the structure $\mathcal{S} = (\omega; +, \times, \mathrm{pr}_1, \leq, c_1, \ldots, c_\ell)$.*

5 Conclusion

Our models of automata over algebraic structure provide a general framework for finite-state computation. Observe that: (1) we can vary the underlying structures $\mathcal{S}$ thus connecting algebraic properties of $\mathcal{S}$ with finite state machines, (2) in certain precise sense our machines can simulate Turing machines, (3) many known automata models (e.g., pushdown automata, Petri nets) can be simulated by our models of automata, and (4) the class of languages recognised by a $\mathcal{S}$-automata is closed under the Boolean set-theoretic operations. This extends the finite automata and tree automata models of computations. However, we note that it remains to be seen whether our model of automata leads to a general framework to decidability results for various models of automata (e.g., pushdown automata, vector addition systems).

Apart from the mentioned references, we note that the current paper refines and extends the approach taken in [11]. We also mention the papers [18,19] that, motivated by the approach in [11], develop the theory of automata over the fields of reals and complex numbers. We note that the current paper also addresses some topics discussed in [14]. It could be interesting to address simulation issues for our models of automata as for finite automata, as in [7,8].

References

1. Alur, R., Černý, P., Weinstein, S.: Algorithmic analysis of array-accessing programs. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 86–101. Springer, Heidelberg (2009)
2. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bull. Am. Math. Soc. **21**(1), 1–46 (1989)
3. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: Proceedings of LICS 2006, pp. 7–16 (2006)
4. Bojanczyk, M., David, C., Muscholl, M., Schwentick, T., Segoufin, L.: Two-variablelogic on data trees and XML reasoning. In: Proceedings of PODS 2006, pp. 10–19 (2006)
5. Bournez, O., Cucker, F., de Naurois, P.J., Marion, J.-Y.: Computability over an arbitrary structure. Sequential and parallel polynomial time. In: Gordon, A.D. (ed.) FOSSACS 2003. LNCS, vol. 2620, pp. 185–199. Springer, Heidelberg (2003)
6. Bozga, M., Iosif, R., Lakhnech, Y.: Flat parametric counter automata. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 577–588. Springer, Heidelberg (2006)
7. Calude, C., Calude, E., Khoussainov, B.: Deterministic automata: simulation and minimality. Ann. Pure Appl. Logic **90**(1–3), 263–276 (1997)
8. Calude, C., Calude, E., Khoussainov, B.: Finite nondeterministic automata: simulation and minimality. Theor. Comput. Sci. **242**(1–2), 219–235 (2000)
9. Comon, S., Jurski, Y.: Multiple counters automata, safety analysis and Presburger arithmetic. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998)
10. Figueira, D.: Reasoning on words and trees with data. Ph.D. thesis, ENS Cachan, France (2010)

11. Gandhi, A., Khoussainov, B., Liu, J.: Finite automata over structures (Extended Abstract). In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 373–384. Springer, Heidelberg (2012)
12. Ibarra, O.: Reversal-bounded multicounter machines and their decision problems. J. ACM **25**(1), 116–133 (1978)
13. Kaminsky, M., Francez, N.: Finite memory automata. Theor. Comput. Sci. **134**(2), 329–363 (1994)
14. Khoussainov, B., Nerode, A.: Open questions in the theory of automatic structures. Bull. Eur. Assoc. Theor. Comput. Sci. (EATCS) (94):181–204 (2008)
15. Leroux, J., Sutre, G.: Flat counter automata almost everywhere!. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 489–503. Springer, Heidelberg (2005)
16. Matiyasevich, Y.: Hilbert's Tenth Problem. MIT Press, Massachusetts (1993)
17. Minsky, M.: Recursive unsolvability of post's problem of "Tag" and other topics in theory of turing machines. Ann. Math. **74**(3), 437–455 (1961)
18. Meer, K., Naif, A.: Generalised finite automata over real and complex numbers. Theor. Comput. Sci. **591**(C), 86–98 (2015)
19. Meer, K., Naif, A.: Periodic generalized automata over the reals. In: Dediu, A.-H., et al. (eds.) LATA 2016. LNCS, vol. 9618, pp. 168–180. Springer, Heidelberg (2016). doi:10.1007/978-3-319-30000-9_13
20. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Logic **15**(3), 403–435 (2004)
21. Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 41–57. Springer, Heidelberg (2006)
22. Segoufin, L., Torunczyk, S.: Automata based verification over linearly ordered data domains. In: Proceedings of STACS, pp. 81–92. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)
23. Tan, T.: Graph reachability and pebble automata over infinite alphabets. In: Proceedings of LICS, pp. 157–166. IEEE Computer Society (2009)