Finite Automata over Structures
(Extended Abstract)

Aniruddh Gandhi[1], Bakhadyr Khoussainov[1], and Jiamou Liu[2]

[1] Department of Computer Science, University of Auckland, New Zealand
[2] School of Computing and Mathematical Sciences
Auckland University of Technology, New Zealand
agan014@aucklanduni.ac.nz, bmk@cs.auckland.ac.nz,
jiamou.liu@aut.ac.nz

Abstract. We introduce a finite automata model for performing computations over an arbitrary structure $\mathcal{S}$. The automaton processes sequences of elements in $\mathcal{S}$. While processing the sequence, the automaton tests atomic relations, performs atomic operations of the structure $\mathcal{S}$, and makes state transitions. In this setting, we study several problems such as closure properties, validation problem and emptiness problems. We investigate the dependence of deciding these problems on the underlying structures and the number of registers of our model of automata. Our investigation demonstrates that some of these properties are related to the existential first order fragments of the underlying structures.

1 Introduction

Most algorithms use methods, operations, and test predicates over an already defined underlying structure. For instance, algorithms that use integer variables assume that the underlying structure contains the set of integers $\mathbb{Z}$ and the usual operations of addition $+$, multiplication $\times$, and the predicate $\leq$. Similarly algorithms that work on graphs or trees assume that the underlying structure consists of graphs and trees with operations such as adding or deleting a vertex or an edge, merging trees or graphs, and test predicates such as the subtree predicate. Generally, an algorithm over an algebraic structure $\mathcal{S} = (D; f_0, \ldots, f_n, R_0, \ldots, R_k)$, where each f_i is a total operation on D and each R_i is a predicate on D, is a sequence of instructions that uses the operations and predicates of the structure. This simple observation has led to the introduction of various models of computations over arbitrary structures and their analysis. The first example here is the class of Blum-Shub-Smale (BSS) machines [2], where the underlying structure is the ordered ring of the reals. The model is essentially a multiple register machine that stores tuples of real numbers and that can evaluate polynomials at unit cost. The second example is the work of O. Bournez, et al. [5], where the authors introduce computations over arbitrary structures thus generalizing the work of L. Blum, M. Shub and S. Smale [2]. In particular, 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 $\mathcal{S}$.

The third example is various classes of counter automata that use counters in different ways [6,7,10,14,16].

In this paper, we introduce the notion of finite automata over algebraic structures which accept or reject finite sequences of elements from the domain of the underlying structure. Our main motivation here is that our model is the finite automata analogue of BSS machines over arbitrary structures. Namely, we define finite state automata over any given structure $\mathcal{S}$. Such an automaton is equipped with a finite number of states, a fixed number of registers, a read only head that always moves to the right in the tape and transitions between the states. The automaton processes finite sequences of elements of $\mathcal{S}$. During the computation, given an input from the sequence, the automaton tests the input against the values of the registers. Depending on the outcomes of the test, the automaton updates the register values by performing basic operations on the input and the register values and then makes a transition to a state. Given a structure $\mathcal{S}$, we use the term $\mathcal{S}$*-automata* to denote the instantiation of this computation model for $\mathcal{S}$.

Another motivation is that our model can also be viewed as finite automata over an infinite alphabet when the underlying structure $\mathcal{S}$ is infinite. We mention that there has recently been a lot of interest in the study of finite automata over infinite alphabets due to investigations in program verification and databases, [1,3,4,9,19,21]. One goal of these investigations is to extend automata-theoretic techniques to words and trees over data values. Several models of computations have been proposed towards this goal. Examples of such automata models include Kaminsky and Francez's register automata [11], Neven, Schwentick and Vianu's pebble automata [17], Bojanczyk's data automata [3] and Alur's extended data automata [1]. While all the above automata models allow only equality tests between data values, there has also been automata model proposed for linearly ordered data domains [20]. The existence of many such models of automata over either structures or infinite alphabets calls for a general yet simple framework to formally reason about such finite state automata. This paper addresses this issue and suggests one such framework.

One important property of our model is that the class of all languages recognized by deterministic $\mathcal{S}$-automata is closed under all the Boolean operations. Furthermore, every language recognized by an $\mathcal{S}$-automaton over a (computable) structure is decidable. Thus $\mathcal{S}$-automata do not generate undecidable languages. Another important implication of our definition is that one can recast many decision problems about standard finite automata in our setting. For instance, we address *the emptiness problem* for $\mathcal{S}$-automata and investigate the interplay between decidability and undecidability of the emptiness problem by varying the structure $\mathcal{S}$. In particular, we provide examples of fragments of arithmetic over which the emptiness problem becomes decidable or undecidable. The third implication is that our model recasts the emptiness problem for finite automata by refining the problem as follows. One would like to design 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 $\mathcal{S}$-automata and connect it with the first order existential fragment of the underlying structure $\mathcal{S}$. Roughly, the existential fragment of the structure $\mathcal{S}$ is equivalent to finding solutions to systems of equations and in-equations in the structure. We show that the validation problem for $\mathcal{S}$-automata is decidable if and only if the existential fragment of the structure $\mathcal{S}$ is decidable.

It is still a speculation that our model provides a general framework for all other known models of automata. However, generality of our model comes from the following observations: (1) we can vary the underlying structures and thus investigate models of finite automata over arbitrary structures, (2) in certain precise sense our machines can simulate Turing machines, (3) many known automata models (e.g pushdown automata, Petri nets, visibly pushdown automata) can easily be simulated by our model but whether decidability results for these models can be derived from decidability results of our model remains to be seen.

The rest of the paper is organized as follows. Section 2 provides basic definitions and introduces the notion of $\mathcal{S}$-automata over structures. Section 3 discusses some basic properties of $\mathcal{S}$-automata. Section 4 investigates the validation problem that we mentioned above. Section 5 investigates the emptiness problem and provides both negative and positive cases. The emphasis here is on the study of $\mathcal{S}$-automata when the structure $\mathcal{S}$ constitutes some natural fragments of arithmetic. For instance we show that the emptiness problem for automata with two registers over the structure $(\mathbb{N}; +1, -1, =, \mathrm{pr}_1, 0)$ is undecidable. In contrast to this we show that the emptiness problem for automata with one register over much richer structure $(\mathbb{N}; +, \times, =, \leq, \mathrm{pr}_1, \mathrm{pr}_2, c_1, \ldots, c_k)$ is decidable. Section 6 discusses potential future work.

2 The Automata Model

A structure $\mathcal{S}$ consists of a (possibly infinite) domain D and finitely many atomic operations $f_1, \ldots, f_m$, relations $R_1, \ldots, R_n$ and constants $c_1, \ldots, c_\ell$ on the set D. We denote this by

$$\mathcal{S} = (D; f_1, \ldots, f_m, R_1, \ldots, R_n, c_1, \ldots, c_\ell).$$

To simplify our notation, we consider structures whose operations and relations have arity 2. Generally speaking, the structures under consideration can be arbitrary structures. Therefore the operations and relations are not necessarily computable. However, we will always assume that given two elements x_1, x_2 in the domain, computing the value of $f_i(x_1, x_2)$ as well as checking $R_j(x_1, x_2)$ can be carried out effectively for all i and j. We denote the set of all atomic operations and the set of all atomic relations of $\mathcal{S}$ by $\mathsf{Op}(\mathcal{S})$ and $\mathsf{Rel}(\mathcal{S})$, respectively.

Definition 1. *A D-*word *of length t is a sequence $a_1 \ldots a_t$ of elements in the domain D. A D-*language *is a set of D-words.*

Given a structure $\mathcal{S}$ with domain D, we investigate a certain type of programs that process D-words. Informally, such a program reads a D-word as input while updating a fixed number of registers. Each register holds an element in D at any given time. Whenever the program reads an element from the input D-word, it first checks if some atomic relations hold on this input element and the current values of the registers, then applies some atomic operations to update the registers. The program stops when the last element in the D-word is read.

We model such programs using finite state machines and call our model $(\mathcal{S}, k)$-automata ($k \in \mathbb{N}$). An $(\mathcal{S}, k)$-automaton keeps k *changing registers* as well as ℓ *constant registers.* The ℓ constant registers store the constants $\overline{c} = c_1, \dots, c_\ell$ and their values are fixed. Each changing register stores an element of D at any time. We normally use $m_1, \dots, m_k$ to denote the current values of the changing registers. Inputs to the automaton are written on a one-way read-only tape. Every state q is associated with $k + \ell$ atomic relations $P_1, \dots, P_{k+\ell} \in \mathsf{Rel}(\mathcal{S})$. Whenever the state q is reached, the $(\mathcal{S}, k)$-automaton reads the next element x of the input D-word and tests the predicate $P_i(x, m_i)$ for each $i \in \{1, \dots, k\}$ and $P_{k+j}(x, c_j)$ for each $j \in \{1, \dots, \ell\}$. The $(\mathcal{S}, k)$-automaton then chooses a transition depending on the outcome of the tests and moves to the next state. Each transition is labelled with k operations, say $g_1, \dots, g_k \in \mathsf{Op}(\mathcal{S})$. The automaton changes the value of its ith register from m_i to $g_i(m_i, x)$. After all elements on the input tape have been read, the $(\mathcal{S}, k)$-automaton stops and decides whether to accept the input depending on the current state. Here is a formal definition.

Definition 2. *An* $(\mathcal{S}, k)$-automaton *is a tuple* $\mathcal{A} = (Q, \alpha, \overline{x}, \Delta, q_0, F)$ *where* Q *is a finite set of states, the mapping* α *is a function from* Q *to* $\mathsf{Rel}^{k+\ell}(\mathcal{S})$*,* $\overline{x} \in D^k$ *are the initial values of the registers,* $q_0 \in Q$ *is the initial state,* $F \subseteq Q$ *is the set of accepting states and* $\Delta \subseteq Q \times \{0,1\}^{k+\ell} \times Q \times \mathsf{Op}^k(\mathcal{S})$ *is the transition relation of* $\mathcal{A}$*. The* $(\mathcal{S}, k)$*-automaton is* deterministic *if for each* $q \in Q$*,* $\overline{b} \in \{0,1\}^{k+\ell}$*, there is exactly one* q' *and* $\overline{g} \in \mathsf{Op}^k(\mathcal{S})$ *such that* $(q, \overline{b}, q', \overline{g}) \in \Delta$*. A* (deterministic) $\mathcal{S}$-automaton *is a (deterministic)* $(\mathcal{S}, k)$*-automaton for some* k*.*

One can view each state q of an $\mathcal{S}$-automaton as a test state and an operational state; the state q is a test state because the predicates from $\alpha(q)$ are tested on tuples of the form (a, m) where a is the input and m is a value from the registers. The state q is an operational state because depending on the outcomes of the tests, an appropriate list of operations are applied to the tuples (m, a).

To define runs of $\mathcal{S}$-automata, we introduce the following notations. For any $k \in \mathbb{N}$, given a tuple $\overline{P} = (P_1, \dots, P_k) \in \mathsf{Rel}^k(\mathcal{S})$, $\overline{m} = (m_1, \dots, m_k) \in D^k$ and $a \in D$, we let $\chi(\overline{P}, \overline{m}, a) = (b_1, \dots, b_k) \in \{0,1\}^k$ such that $b_i = 1$ if $\mathcal{S} \models P_i(a, m_i)$ and $b_i = 0$ otherwise, where $1 \leq i \leq k$. Fix an $(\mathcal{S}, k)$-automaton $\mathcal{A}$. A *configuration* of $\mathcal{A}$ is a tuple $r = (q, \overline{m}) \in Q \times D^k$. Given two configurations $r_1 = (q, \overline{m})$, $r_2 = (q', \overline{m'})$ and $a \in D$, by $r_1 \hookrightarrow_a r_2$ we denote that $(q, \chi(\alpha(q), (\overline{m}, \overline{c}), a), q', g_1, \dots, g_k) \in \Delta$ and $m'_i = g_i(m_i, a)$ for all $i \in \{1, \dots, k\}$.

Definition 3. *A* run *of $\mathcal{A}$ on a D-word $a_1 \dots a_n$ is a sequence of configurations*

$$r_0, r_1, r_2, \dots, r_n$$

where $r_0 = (q_0, x_1, \dots, x_k)$ and $r_{i-1} \hookrightarrow_{a_i} r_i$ for all $i \in \{1, \dots, n\}$. The run is accepting *if the state in the last configuration r_n is accepting. The $(\mathcal{S}, k)$-automaton $\mathcal{A}$* accepts *the D-word $a_1 \dots a_n$ if $\mathcal{A}$ has an accepting run on $a_1 \dots a_n$. The* language *$L(\mathcal{A})$ of the automaton is the set of all D-words accepted by $\mathcal{A}$.*

We say a D-language L is *(deterministic) $\mathcal{S}$-automata recognizable* if $L = L(\mathcal{A})$ for some (deterministic) $\mathcal{S}$-automata $\mathcal{A}$. The next section presents several examples of $\mathcal{S}$-automata recognizable languages and discuss some simple properties of $\mathcal{S}$-automata. These examples and properties provide justification to investigate $\mathcal{S}$-automata as a general framework for finite state machines.

3 Simple Properties of $\mathcal{S}$-Automata

We present several examples to establish some simple properties of $\mathcal{S}$-automata and $\mathcal{S}$-automata recognizable D-languages. The first result shows that when $\mathcal{S}$ is a finite structure, $\mathcal{S}$-automata recognize regular languages. We use $\mathcal{S}[\overline{a}]$ to denote the structure obtained from $\mathcal{S}$ by adding constants $\overline{a}$ to the signature. Suppose that the structure $\mathcal{S}$ contains an atomic equivalence relation $\equiv$ of finite index. Let $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ be the set of all equivalence classes of $\equiv$. For every word $w = w_1 \dots w_n$ over the alphabet Σ, let $R(w)$ be the D-language $\{a_1 \dots a_n \mid a_i \in w_i \text{ for all } i = 1, \dots, n\}$. For every language $\mathcal{L}$ over Σ, let $R(\mathcal{L})$ be the D-language $\bigcup_{w \in \mathcal{L}} R(w)$.

Theorem 4. *Let a_i be an element from the $\equiv$-equivalence class σ_i, where $i = 1, \dots, k$. Then each of the following is true.*

- *For every regular language $\mathcal{L}$ over Σ, the D-language $R(\mathcal{L})$ is recognized by an $(\mathcal{S}[a_1, \dots, a_k], 0)$-automaton.*
- *Suppose $\mathcal{S}$ contains only one atomic relation $\equiv$ and the atomic operations of $\mathcal{S}$ are compatible with $\equiv$. For every $\mathcal{S}$-automata recognizable D-language W, there is a regular language $\mathcal{L}$ over Σ such that $W = R(\mathcal{L})$.*

Example 5. Using Theorem 4 one can present many example of $\mathcal{S}$-automata recognizable D-languages.

(a) Regular languages over $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ are recognized by $(\mathcal{S}, 0)$-automata where $\mathcal{S} = (\Sigma; =, \sigma_1, \dots, \sigma_k)$.
(b) Let $\mathcal{S}$ be a finite structure with domain D where equality is part of the signature. Any D-language acceptable by an $\mathcal{S}$-automaton is a regular language over the alphabet D.
(c) Let $\mathcal{S}$ be $(\mathbb{Z}; \equiv)$ where $\equiv = \{(i,j) \mid i = j = 0 \text{ or } ij > 0\}$. The following $\mathbb{Z}$-languages is recognized by $(\mathcal{S}[-1, 0, 1], 0)$-automata:

$$\{n_0 \dots n_k \mid k \in \mathbb{N}, n_j > 0 \text{ when } j \text{ is even and } n_j < 0 \text{ when } j \text{ is odd}\}.$$

We now single out two functions that will be used throughout the paper.

Definition 6. *For $i \in \{1,2\}$, define* projection *on the ith coordinate as an operation* $\mathrm{pr}_i : D^2 \to D$ *such that* $\mathrm{pr}_i(a_1, a_2) = a_i$ *for all* $a_1, a_2 \in D$.

The next example shows that for infinite structures of the form $\mathcal{S} = (D; =, \mathrm{pr}_1, \mathrm{pr}_2)$, the class of $(\mathcal{S}, k)$-automata recognizable D-languages properly contains the class of $(\mathcal{S}, k-1)$-automata recognizable D-languages.

Example 7 (Separation between $(\mathcal{S}, k)$-automata and $(\mathcal{S}, k+1)$-automata). Let $\mathcal{S} = (D; =, \mathrm{pr}_1, \mathrm{pr}_2)$ with D infinite. For $k > 0$, let D_k be the D-language

$$\{a_0 \dots a_k \mid \forall i, j \in \{0, \dots, k\} : i \neq j \Rightarrow a_i \neq a_j\}.$$

It is easy to see that an $(\mathcal{S}, k)$-automaton recognizes the D-language D_k but no $(\mathcal{S}, k-1)$-automaton recognizes D_k.

The next example shows that deterministic $\mathcal{S}$-automata form a proper subclass of $\mathcal{S}$-automata. Furthermore, the class of $\mathcal{S}$-automata recognizable D-languages is not closed under Boolean operations in the general case.

Example 8 (Separation between deterministic and nondeterminstic $\mathcal{S}$-automata). Let $\mathcal{S} = (\mathbb{N}; +, \mathrm{pr}_1, =, 1)$. Let L be the $\mathbb{N}$-language $\{1^n m \mid n, m \in \mathbb{N}, m \leq n\}$. It is clear that an $(\mathcal{S}, 1)$-automaton recognizes L. However, such an $\mathcal{S}$-automaton is necessarily nondeterministic. One may prove that no deterministic $\mathcal{S}$-automaton recognizes L. Now let L' be the $\mathbb{N}$-language $\{1^n m \mid n, m \in \mathbb{N}, m > n\}$. One may prove that no $\mathcal{S}$-automaton recognizes L'. Since $\mathbb{N}^* \setminus L = \{\varepsilon\} \cup \{1^n m w \mid n, m \in \mathbb{N}, m \neq 1, w \in \mathbb{N}^+\} \cup L'$, it is easy to see that the class of $\mathcal{S}$-automata recognizable $\mathbb{N}$-languages is not closed under the set operations.

The rest of the section focuses on deterministic $\mathcal{S}$-automata. The next theorem shows that the class of deterministic $\mathcal{S}$-automata recognizable D-languages forms a Boolean algebra.

Theorem 9 (Closure under Boolean operations). *Let $\mathcal{S}$ be a structure. The class of languages recognized by deterministic $\mathcal{S}$-automata is closed under union, intersection and complementation.*

Below we present two examples of deterministic $\mathcal{S}$-automata where the structure $\mathcal{S}$ has the set of natural numbers $\mathbb{N}$ as its domain.

Example 10. Let $\mathcal{S} = (\mathbb{N}; +, \mathrm{pr}_1, =)$. Let F contain all *Fibonacci sequences*, i.e. $\mathbb{N}$-words $a_1 a_2 \dots a_n$ $(n \in \mathbb{N})$ where $a_{i+2} = a_{i+1} + a_i$ for $i \in \{1, \dots, n-2\}$. Then a deterministic $(\mathcal{S}, 2)$-automaton accepting F is shown in Fig 1.

The next example shows how deterministic $\mathcal{S}$-automata may be used to accept execution sequences of algorithms.

Example 11. Let $\mathcal{S} = (\mathbb{N}; +, /, \mathrm{pr}_2, =, 0)$ where $/$ denotes the modulo operation on natural numbers, where $a/b = r$ means $r < b$ and $r + bq = a$ for some $q \in \mathbb{N}$. Euclid's algorithm computes the greatest common divisor of two given natural numbers $x, y \in \mathbb{N}$ by repeatedly computing the sequence $a_1, a_2, a_3, \dots$ such that

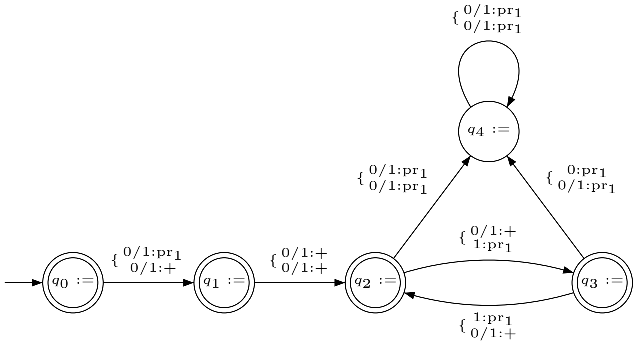

Fig. 1. An $(\mathcal{S}, 2)$-automaton accepting the Fibonacci sequences. The initial value is $(0, 0)$.

$a_1 = x, a_2 = y$, and $a_i = a_{i-2}/a_{i-1}$ for $i > 2$. The procedure terminates when $a_i = 0$ and declares that a_{i-1} is $\gcd(x, y)$. We call such a sequence $a_1, a_2, a_3, \ldots$ an *Euclidean path.* For example, the $\mathbb{N}$-word 384 270 114 42 30 12 6 0 is an Euclidean path. Note that if $a_1 \ldots a_n$ is an Euclidean path, then $a_{n-1} = \gcd(a_1, a_2)$. Hence an Euclidean path can be thought of as a computation of Euclid's algorithm. Then a deterministic $(\mathcal{S}, 2)$-automaton accepting the set of all Euclidean paths is shown in Fig 2.

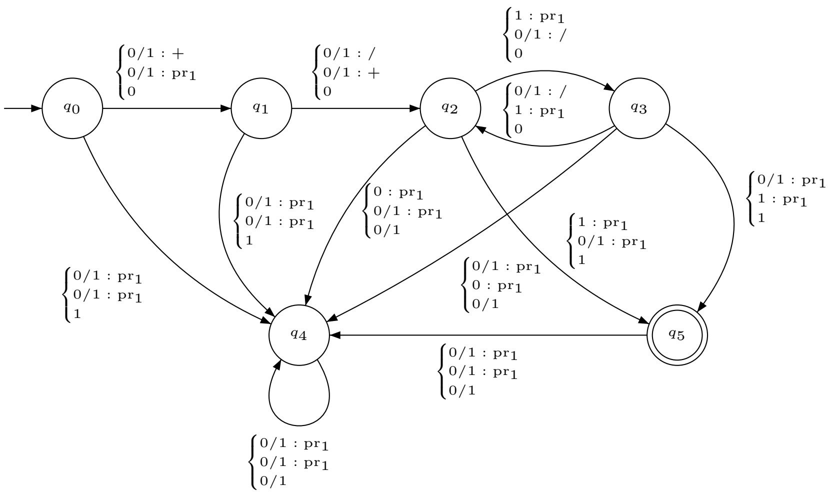

Fig. 2. An $((\mathbb{N}; +, /, \mathrm{pr}_1, =, 0), 3)$-automaton accepting the Euclidean paths. The initial value is $(0, 0, 0)$. The mapping α maps every state q to the tuple $(=, =, =)$.

4 The Validation Problem

This section discusses the validation problem for automata over a given structure $\mathcal{S}$. The main result is that deciding the validation problem on $\mathcal{S}$-automata is equivalent to deciding the existential theory of the structure $\mathcal{S}$ (see Theorem 13 below). The validation problem is formulated as follows.

Validation Problem. Design an algorithm that, given an $\mathcal{S}$-automaton $\mathcal{A}$ and a path p in $\mathcal{A}$ from the initial state to an accepting state, decides if there exists a D-word $\overline{a}$ such that a run of $\mathcal{A}$ over $\overline{a}$ proceeds along p.

Obviously the problem depends on the given structure $\mathcal{S}$. For instance, if $\mathcal{S}$ is a finite structure then, by Example 5(b), both the validation and the emptiness problem are decidable. The validation problem for $\mathcal{S}$-automata turns out to be equivalent to solving systems of equations and in-equations over the structure. More formally, we define the following:

Definition 12. *The* existential theory *of $\mathcal{S}$, denoted by $\mathsf{Th}_\exists(\mathcal{S})$ is the set of all existential sentences true in $\mathcal{S}$, that is,*

$$\mathsf{Th}_\exists(\mathcal{S}) = \{\varphi \mid \mathcal{S} \models \varphi \text{ and } \varphi \text{ is an existential sentence}\}.$$

The following is the main result of this section:

Theorem 13. *The validation problem for $\mathcal{S}[\mathrm{pr}_1, \mathrm{pr}_2, =]$-automata is decidable if and only if $\mathsf{Th}_\exists(\mathcal{S})$ is decidable.*

5 The Emptiness Problem

This section discusses the emptiness problem for $\mathcal{S}$-automata.

Emptiness Problem. Design an algorithm that, given a structure $\mathcal{S}$ and an $(\mathcal{S}, k)$-automaton $\mathcal{A}$, decides if $\mathcal{A}$ accepts at least one D-word.

A *sink state* in an $\mathcal{S}$-automaton is a state whose all outgoing transitions loop into the state itself. All accepting sink states can be collapsed into one accepting sink state, and all non-accepting sink states can be collapsed into one non-accepting sink state. Therefore we can always assume that every $\mathcal{S}$-automaton has at most 2 sink states.

Definition 14. *We call an $\mathcal{S}$-automaton* acyclic *if its state space without the sink states is an acyclic graph.*

Note that in any acyclic $\mathcal{S}$-automaton, there are only finitely many paths from the initial state to an accepting state. Hence the emptiness problem is computationally equivalent to the validation problem.

Theorem 15. *The emptiness problem of acyclic $\mathcal{S}[\mathrm{pr}_1, \mathrm{pr}_2, =]$-automata is decidable if and only if $\mathcal{S}$ has decidable existential theory.*

The above theorem immediately provides a wide range of structures $\mathcal{S}$ for which the emptiness problem of acyclic $\mathcal{S}[\mathrm{pr}_1, \mathrm{pr}_2, =]$-automata is decidable. Below we list several examples of such structures. The structures (a-c) are well-known to have decidable first-order theory, (d) has decidable theory by [18], and (e,f) have decidable theory since they are instances of automatic structures [12].

Example 16. The emptiness problem is decidable for acyclic $\mathcal{S}[\mathrm{pr}_1, \mathrm{pr}_2, =]$-automata where $\mathcal{S}$ is the following structures and $c_1, \ldots, c_k$ are constants in the respective domain:

(a) $(\mathbb{N}; +, <, \leq, c_1, \ldots, c_\ell)$.
(b) $(\mathbb{N}; \times, c_1, \ldots, c_\ell)$.
(c) Any finitely generated Abelian group.
(d) $(\mathbb{N}; +, \mathsf{pow}_2, c_1, \ldots, c_\ell)$ where the function $\mathsf{pow}_2 : \mathbb{N}^2 \to \mathbb{N}$ is the function $(x, y) \to 2^x$.
(e) $(\mathbb{Q}; +, \leq, c_1, \ldots, c_\ell)$ where $\mathbb{Q}$ is the set of rational numbers.
(f) The Boolean algebra of finite and co-finite subsets of $\mathbb{N}$.

Theorem 15 above poses the following question. Let $\mathcal{S}$ be a structure with undecidable existential theory, find k such that the emptiness problem for acyclic $(\mathcal{S}, k)$-automata is undecidable. Speculatively there might be a structure $\mathcal{S}$ with undecidable existential theory (and hence undecidable emptiness problem for acyclic $\mathcal{S}$-automata) such that for each k the emptiness problem for acyclic $(\mathcal{S}, k)$-automata is decidable, but we don't know any such example. Below we provide an example of a structure $\mathcal{S}$ such that the emptiness problem for acyclic $(\mathcal{S}, 1)$-automata is undecidable.

Let $G = (V, E)$ be a computable graph for which testing whether each node is isolated is undecidable (the reader is referred to [8] for the existence of such a graph). Then the following acyclic $(G[\mathrm{pr}_2, =], 1)$-automaton $\mathcal{A}$ has undecidable emptiness problem. The $(G, 1)$-automaton $\mathcal{A}$ has four states $q_0, q_1, q_\mathsf{f}, q_\mathsf{s}$ where $q_\mathsf{f}, q_\mathsf{s}$ are sink states and $F = \{q_\mathsf{f}\}$. The mapping α maps q_0 to $=$ and q_1 to E. The transitions on q_0 and q_1 are

$$\{(q_0, b, q_1, \mathrm{pr}_2) \mid b = 0, 1\} \cup \{(q_1, 0, q_\mathsf{s}, \mathrm{pr}_2)\} \cup \{(q_1, 1, q_\mathsf{f}, \mathrm{pr}_2)\}$$

We now give a more natural example of a structure $\mathcal{S}$ where emptiness problem is undecidable for acyclic $(\mathcal{S}, k)$-automata with small k. By a reduction from Hilbert's tenth problem[15], one obtain the following theorem.

Theorem 17. *Consider the following structures:*

$$\mathcal{S}_\mathbb{Z} = (\mathbb{Z}; +, \times, \mathrm{pr}_1, \mathrm{pr}_2, =, 0) \text{ and } \mathcal{S}_\mathbb{N} = (\mathbb{N}; +, \times, \mathrm{pr}_1, \mathrm{pr}_2, =, 0).$$

(a) The emptiness problem for deterministic acyclic $(\mathcal{S}_\mathbb{Z}, 11)$-automata is undecidable.
(b) The emptiness problem for deterministic acyclic $(\mathcal{S}_\mathbb{N}, 12)$-automata is undecidable.

A natural question is the decidability of the emptiness problem if we remove the acyclicity constraint. We denote with $+1$ and -1 the binary operations on $\mathbb{N}^2$ such that $+1(x, y) = x + 1$ and $-1(x, y) = x - 1$ (Note that -1 is not total). The next theorem shows that if we remove the acyclicity constraint, the emptiness problem is undecidable for $\mathcal{S}$-automata with a small number of registers.

Theorem 18. *Let $\mathcal{S}_1 = (\mathbb{N}; +1, -1, =, \mathrm{pr}_1, 0)$ and $\mathcal{S}_2 = (\mathbb{N}, +1, =, \mathrm{pr}_1, \mathrm{pr}_2, 0)$.*

(a) The emptiness problem for deterministic $(\mathcal{S}_1, 2)$-automata is undecidable.
(b) The emptiness problem for deterministic $(\mathcal{S}_2, 4)$-automata is undecidable.

The next question is whether the emptiness problem is undecidable if we lower the number of register even further.

Theorem 19. *Let $\mathcal{S}$ be the structure $(\mathbb{N}; +, \times, \mathrm{pr}_1, \mathrm{pr}_2, =, \leq, c_1, \ldots, c_\ell)$ where $c_1, \ldots, c_\ell$ are constants in $\mathbb{N}$. The emptiness problem for $(\mathcal{S}, 1)$-automata is decidable.*

Another way of restricting the automata is to put constraints on the allowable transitions of the automata. We show in the following that by allowing only those transitions that compare the input or a changing register with constants, the emptiness problem may become decidable. Our motivation is to analyze those algorithm in which comparisons occur only between variables and a fixed number of constant values. For example we may allow the comparison $a < 5$ but not the comparison $a < b$ where a, b are variables. With this in mind, we now introduce a class of automata which we call the *constant comparing automata*. We would like to show such automata have a decidable emptiness problem. For the sake of convenience we add the relation $U = \mathbb{N}^2$ to our structures. This can be done without any loss of generality.

Definition 20. *Let $\mathcal{S}$ be a structure that contains $=$ as an atomic relation and ℓ constants $c_1, \ldots, c_\ell$ in its signature. A constant comparing $(\mathcal{S}, k)$-automaton is $\mathcal{A} = (Q, \alpha, \overline{y}, \Delta, q_0, F)$ such that for every $q \in Q$, $\alpha(q) = (R_1, \ldots, R_{k+\ell})$ satisfies the following conditions:*

- *There is at most one $i \in \{1, \ldots, k\}$ such that R_i is the $=$ relation.*
- *For all $j \in \{1, \ldots, k\}$ apart from i (if it exists) we have $R_j = U$.*

The $(\mathcal{S}, k)$-automaton $\mathcal{A}$ is a strongly constant comparing $\mathcal{S}$-automaton *if no such i exists.*

Note that by definition an $(\mathcal{S}, 1)$-automaton is also a constant comparing $\mathcal{S}$-automaton. Hence the next theorem can be viewed as a generalization of Theorem 19.

Theorem 21. *Let $\mathcal{S}$ be the structure $(\mathbb{N}; +, \times, \mathrm{pr}_1, \mathrm{pr}_2, =, \leq, U, c_1, \ldots, c_\ell)$ where $c_1, \ldots, c_\ell$ are arbitrary constants in $\mathbb{N}$. The emptiness problem for constant comparing $\mathcal{S}$-automata is decidable.*

Note that the structure $\mathcal{S}$ in Theorem 21 does not contain subtraction as part of the signature. If subtraction is added to the signature, one may show by slightly modifying the proof of Theorem 18(a) that the emptiness problem becomes undecidable. However, the emptiness problem becomes decidable if we restrict to strongly constant comparing $\mathcal{S}$-automata as shown by the following theorem.

Theorem 22. *Let* $\mathcal{S} = (\mathbb{N}; +, -, \mathrm{pr}_1, =, \leq, c_1, \ldots, c_\ell)$ *where* $c_1, \ldots, c_\ell$ *are constants in* $\mathbb{N}$.

(a) The emptiness problem for constant comparing $(\mathcal{S}, 2)$*-automata is undecidable if* $\ell \geq 2$.
(b) The emptiness problem for strongly constant comparing $\mathcal{S}$*-automata is decidable.*

6 Discussion and Future Work

One natural direction for future work is to obtain more generic results on the emptiness problem. This may require to identify the common properties of the automata over different structures discussed in this paper, and see how different existing types of automata with external memory (e.g. bounded reversal counter machines, flat counter automata, pushdown automata) fit into this general framework.

Another interesting direction for future work is to identify structures for which this type of automata enjoy closure under the set operations (even in the nondeterministic case) and hence identify connections of these automata with certain logic over the underlying structures.

A third possible direction is to analyze automata over structures whose domains are not natural numbers. Some interesting examples of such structures include real closed fields, the boolean algebra of finite and co-finite subsets with the subset predicate etc.

In this paper we have focused our attention on the decidability of emptiness problem for our automata model. However other classical automata-theoretic decidability problems such as the universality problem, the language inclusion problem and the equivalence problem are also topics for future work.

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. Bulletin of the American Mathematical Society 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. IEEE Computer Society (2006)

4. Bojanczyk, M., David, C., Muscholl, M., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. In: Proceedings of PODS 2006, pp. 10–19. ACM (2006)
5. Bournez, O., Cucker, F., Jacobé de Naurois, P., 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. Comon, S., Jurski, Y.: Multiple Counters Automata, Safety Analysis and Presburger Arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998)
8. Ershov, Y., Goncharov, S., Marek, V., Nerode, A., Remmel, J.: Handbook of Recursive Mathematics: Recursive Model Theory. Studies in Logic and the Foundations of Mathematics. North-Holland (1998)
9. Figueira, D.: Reasoning on words and trees with data. Ph.D. Thesis, ENS Cachan, France (2010)
10. Ibarra, O.: Reversal-bounded multicounter machines and their decision problems. J. ACM 25(1), 116–133 (1978)
11. Kaminsky, M., Francez, N.: Finite memory automata. Theor. Comp. Sci. 134(2), 329–363 (1994)
12. Ishihara, H., Khousainov, B., Rubin, S.: Some Results on Automatic Structures. In: Proceedings of LICS 2002, p. 235. IEEE Computer Society (2002)
13. Leroux, J.: The general vector addition system reachability problem by presburger inductive invariants. In: Procedings of LICS 2009, pp. 4–13. IEEE Computer Society (2009)
14. 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)
15. Matiyasevich, Y.: Hilbert's Tenth Problem. MIT Press, Cambridge (1993)
16. Minsky, M.: Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. Annals of Math. 74(3) (1961)
17. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Tran. Comput. Logic 15(3), 403–435 (2004)
18. Point, F.: On Decidable Extensions of Presburger Arithmetic: From A. Bertrand Numeration Systems to Pisot Numbers. J. Symb. Log. 65(3), 1347–1374 (2000)
19. 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)
20. Segoufin, L., Torunczyk, S.: Automata based verification over linearly ordered data domains. In: Proceedings of STACS 2011. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 81–92 (2011)
21. Tan, T.: Graph reachability and pebble automata over infinite alphabets. In: Proceedings of LICS 2009, pp. 157–166. IEEE Computer Society (2009)