Theoretical Computer Science 591 (2015) 85–98

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Generalized finite automata over real and complex numbers

Klaus Meer *, Ameen Naif [1]

Computer Science Institute, BTU Cottbus-Senftenberg, Platz der Deutschen Einheit 1, D-03046 Cottbus, Germany

ARTICLE INFO

Article history:
Received 6 October 2014
Received in revised form 22 April 2015
Accepted 3 May 2015
Available online 7 May 2015
Communicated by D. Perrin

Keywords:
Generalized finite automata
Real and complex number computations
Decision problems for automata over uncountable structures

ABSTRACT

In a recent work, Gandhi, Khoussainov, and Liu [7] introduced and studied a generalized model of finite automata able to work over arbitrary structures. As one relevant area of research for this model the authors identify studying such automata over particular structures such as real and algebraically closed fields.
In this paper we start investigations into this direction. We prove several structural results about sets accepted by such automata, and analyze decidability as well as complexity of several classical questions about automata in the new framework. Our results show quite a diverse picture when compared to the well known results for finite automata over finite alphabets.

1. Introduction

Finite automata represent one of the fundamental elementary models of algorithms in Computer Science. There is an elaborated theory about problems that can be solved both with and concerning finite automata which now usually is taught in basic theory courses. When dealing with algorithmic questions about finite automata like the word problem, the emptiness and finiteness problems, the equivalence problem or minimization of such automata, such questions are treated by analyzing the Turing model of computation as underlying computational model. Thus, statements like 'the equivalence problem for non-deterministic finite automata is NP-complete' are to be understood using complexity theory in the Turing model.

In recent years theoretical computer science has seen an increasing interest in alternative to the Turing machine models of computation. The reader might think of quantum computers [13], neural networks [8], analogue computers [4], several kinds of biologically inspired devices [14], and models for computations over the real and complex numbers. Models for the latter split into approaches based on the Turing machine like those followed in recursive analysis [15] and in algebraically inspired notions of algorithms [3,5].[2] One feature of such algorithm models is that they do not any longer exclusively work over finite alphabets as underlying structures. For example, the Blum–Shub–Smale (shortly BSS) model introduced in [3] can be and was used to define a computability notion for many different structures including $\mathbb{R}$ and $\mathbb{C}$. It is thus a reasonable question whether also the concept of a (deterministic and non-deterministic) finite automaton can be generalized to work over more general structures than just finite alphabets.

* Corresponding author.
E-mail addresses: meer@informatik.tu-cottbus.de (K. Meer), naifamee@tu-cottbus.de (A. Naif).

[1] A. Naif was supported by a 'Promotionsstipendium nach Graduiertenförderungsverordnung des Landes Brandenburg GradV'. The support is gratefully acknowledged.

[2] For all mentioned areas the given references are not thought to be exhaustive but should just serve as a starting point for readers being more interested in the corresponding models.

http://dx.doi.org/10.1016/j.tcs.2015.05.001
0304-3975/

In recent work Gandhi, Khoussainov, and Liu [7] introduce such a generalized model of finite automata called $(\mathcal{S}, k)$-automata. It is able to work over an arbitrary structure $\mathcal{S}$, and here in particular over infinite alphabets like the real numbers. A structure is characterized by an alphabet (also called universe) together with a finite number of binary functions and relations over that alphabet. Now, intuitively the model processes words over the underlying alphabet componentwise. Each single step is made of finitely many test operations relying on the fixed relations as well as finitely many computational operations relying on the fixed functions. For performing the latter an $(\mathcal{S}, k)$-automaton can use a finite number k of registers. As we shall see the latter ability adds significantly power to the model in comparison to 'old-fashioned' finite automata. An automaton then moves between finitely many states and finally accepts or rejects an input.

The motivation to study such generalizations is manifold. In [7] the authors discuss different previous approaches to design finite automata over infinite alphabets and their role in program verification and database theory. One goal is to look for a generalized framework that is able to homogenize at least some of these approaches. As the authors remark, many classical automata models like pushdown automata, Petri nets, visible pushdown automata can be simulated by the new model. Another major motivation results from work on algebraic models of computation over structures like the real and complex numbers. Here, the authors suggest their model as a finite automata variant of the Blum–Shub–Smale BSS model. They then ask to analyze such automata over structures like real or algebraically closed fields.

The latter will be the focus of the present work. We restrict our attention to generalized finite automata over two special structures denoted by $\mathcal{S}_{\mathbb{R}}$ and $\mathcal{S}_{\mathbb{C}}$ and defined precisely below. These are the suitable choices for relating $(\mathcal{S}, k)$-automata to the BSS model.

The paper is structured as follows. Section 2 recalls the generalized automata model from [7] for the above two structures and gives some basic examples. It collects as well the notions from BSS computability theory necessary to relate the automata model to the latter. We shall then study a bunch of questions well known from finite automata theory. Section 3 analyzes the word problem for deterministic and non-deterministic $(\mathcal{S}, k)$-automata in comparison to certain complexity classes in the BSS model. A complexity class coming into play very naturally here is the class DNP of problems that can be verified by so called digital nondeterminism. In particular, we shall get a somehow diverse picture concerning the classes of languages accepted by (non)-deterministic $(\mathcal{S}, k)$-automata. For example, we shall see that there are easy problems in $\mathrm{P}_{\mathbb{C}}$, the class of problems being polynomial time solvable in the complex number BSS model, that cannot be accepted by any non-deterministic automaton, whereas the complex Knapsack problem $KS_{\mathbb{C}}$ (a problem likely not located in $\mathrm{P}_{\mathbb{C}}$) can. This problem can as well be used to show that languages accepted by non-deterministic complex $(\mathcal{S}, k)$-automata are not closed under complementation. Towards obtaining this result we discuss certain structural properties of languages accepted by complex $(\mathcal{S}, k)$-automata. We discuss as well the real case, where (partially by different arguments) the same results hold as well. A second structural result for complex automata will give a kind of weak pumping lemma.

A number of undecidability results are shown for both structures in Section 4. Among them we find the emptiness problem, the equivalence problem, several reachability questions as well as the problem to minimize an $(\mathcal{S}, k)$-automaton. For some restricted classes of automata we also give decidability results. The paper closes with a discussion concerning open future questions.

2. Generalized finite automata over $\mathbb{R}$ and $\mathbb{C}$

We suppose the reader to be familiar with the basics of the Blum–Shub–Smale model of computation and complexity over $\mathbb{R}$ and $\mathbb{C}$. Very roughly, algorithms in this model work over finite strings of real or complex numbers, respectively. The operations either can be computational, in which case addition, subtraction, and multiplication is allowed; without much loss of generality we do not consider divisions in this paper to avoid technical inconveniences. Or an algorithm can branch depending on the result of a binary test operation. The latter either will be an inequality test of form 'is $x \geq 0$?' over the reals or an equality test 'is $x = 0$?' when working over the complex numbers. The size of a string is the number of components it has, the cost of an algorithm is the number of operations it performs until it halts. For more details see [2]. Notions being central for this paper shall be explained in more detail below.

The generalized finite automata introduced in [7] work over structures. Here, a structure $\mathcal{S}$ consists of a universe D as well as of finite sets of (binary) functions and relations over the universe. A more precise definition concerning the structures we are interested in follows below. An automaton informally works as follows. It reads a word from the universe, i.e., a finite string of components from D and processes each component once. Reading a component the automaton can set up some tests using the relations in the structure. The tests might involve a fixed set of constants from the universe D that the automaton can use. It can as well perform in a limited way computations on the current component. Towards this aim, there is a fixed number k of registers that can store elements from D. Those registers can be changed using their current value, the current input and the functions related to $\mathcal{S}$. After having read the entire input word the automaton accepts or rejects it depending on the state in which the computation stops. These automata can both be deterministic and non-deterministic.

Since the approach allows structures to have arbitrary universes, the model in particular easily can be adapted to define generalized finite automata over structures like $\mathbb{R}$ and $\mathbb{C}$. We shall in the rest of the paper focus on these universes. Our goal is to study questions for these automata in the framework of the BSS model of computations over those structures. We thus define two structures $\mathcal{S}_{\mathbb{R}}$ and $\mathcal{S}_{\mathbb{C}}$ according to the operations used in the BSS model over $\mathbb{R}$ and $\mathbb{C}$, respectively. As in

the original work [7] we equip all our structures as well with the projection operators pr_1, pr_2 which give back the first and the second component of a tuple, respectively.

Definition 1. Let $\mathcal{S}_{\mathbb{R}} := (\mathbb{R}, +, -, \bullet, pr_1, pr_2, \geq, =)$ denote the structure of the reals as ring with order. Similarly, the structure of the complex numbers as ring with equality is given by $\mathcal{S}_{\mathbb{C}} := (\mathbb{C}, +, -, \bullet, pr_1, pr_2, =)$.

In order to avoid technicalities for operation $-$ we allow both orders of the involved arguments, i.e., applying $-$ to two values x, v can mean $x - v$ or $v - x$. Similarly, the order test can be performed both as $x \leq v?$ and $v \leq x?$ As mentioned above without loss of generality we do not include division as an operation. This will not significantly change our results.

The following definition is from [7] but adapted for the special structures exclusively considered here.

Definition 2 *(Finite automata over $\mathcal{S}_{\mathbb{R}}$ and $\mathcal{S}_{\mathbb{C}}$).* (See [7].) Let $k \in \mathbb{N}$ be fixed.

a) A deterministic $(\mathcal{S}_{\mathbb{R}}, k)$-automaton $\mathcal{A}$ consists of the following objects:
   - a finite state space Q and an initial state $q_0 \in Q$,
   - a set $F \subseteq Q$ of accepting states,
   - a set of ℓ registers which contain fixed given constants $c_1, \ldots, c_\ell \in \mathbb{R}$,
   - a set of k registers which can store real numbers denoted by $v_1, \ldots, v_k$,
   - a transition function $\delta : Q \times \mathbb{R} \times \mathbb{R}^k \times \{0,1\}^{k+\ell} \mapsto Q \times \mathbb{R}^k$.

   The automaton processes elements of $\mathbb{R}^* := \bigsqcup_{n \geq 1} \mathbb{R}^n$, i.e., words of finite length with real components. For such an $(x_1, \ldots, x_n) \in \mathbb{R}^n$ it works as follows. The computation starts in q_0 with an initial configuration for the values $v_1, \ldots, v_k \in \mathbb{R}$, say all $v_i = 0$. Then, $\mathcal{A}$ reads the input components step by step. Suppose a value x is read in state $q \in Q$. Now the next state together with an update of the values v_i is computed as follows:
   - $\mathcal{A}$ performs the $k + \ell$ comparisons $x\sigma_1 v_1?, x\sigma_2 v_2?, \ldots, x\sigma_k v_k?, x\sigma_{k+1} c_1?, \ldots, x\sigma_{k+\ell} c_\ell?$, where $\sigma_i \in \{\geq, \leq, =\}$. This gives a vector $b \in \{0,1\}^{k+\ell}$, where a component 0 indicates that the comparison that was tested is violated whereas 1 codes that it is valid;
   - depending on state q and b the automaton moves to a state $q' \in Q$ (which could again be q) and updates the v_i applying one of the operations in the structure: $v_i \leftarrow x \circ_i v_i$. Here, $\circ_i \in \{+, -, \bullet, pr_1, pr_2\}, 1 \leq i \leq k$ depends on q and b only.

   When the final component of an input is read $\mathcal{A}$ performs the tests for this component and moves to its final state without any further computation. It accepts the input if this final state belongs to F, otherwise $\mathcal{A}$ rejects.

b) Non-deterministic $(\mathcal{S}_{\mathbb{R}}, k)$-automata are defined similarly with the only difference that δ becomes a relation in the following sense: If in state q the tests result in $b \in \{0,1\}^{k+\ell}$ the automaton can non-deterministically choose for the next state and the update operations one among finitely many tuples $(q', \circ_1, \ldots, \circ_k) \in Q \times \{+, -, \bullet, pr_1, pr_2\}^k$.
   As usual, a non-deterministic automaton accepts an input if there is at least one accepting computation.

c) (Non-)deterministic automata over $\mathcal{S}_{\mathbb{C}}$ are defined similarly, the only difference being that the tests all have to be equality tests.

d) For an automaton $\mathcal{A}$ the language of finite strings accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$. Clearly, $L(\mathcal{A}) \subseteq \mathbb{R}^*$ or $L(\mathcal{A}) \subseteq C^*$, depending on the structure considered.

Remark 1. a) A few words concerning the intuitive abilities and some technical aspects of generalized automata are appropriate here. Given k registers an automaton can perform calculations. Depending on the problem considered this ability can be relatively strong. It is, for example, easy to see that in appropriate structures like $\mathcal{S}_{\mathbb{R}}$ and $\mathcal{S}_{\mathbb{C}}$ one can count in certain situations. For example, a language like $\{(-1)^n 1^n | n \in \mathbb{N}\}$ easily is seen to be acceptable by such an automaton if 1 is available as constant. On the other hand input components still can be read only once. If we want to use it several times it has to be stored in one of the k registers in order to be used again. This of course might be a severe restriction.

b) Another technical aspect refers to the initialization of a computation. In the original definition in [7] one can start a computation with an arbitrary assignment for the registers v_i. However, this might mean that additional constants are introduced into calculations. Below we do not want to analyze which impact the use of an initialization in general has with respect to the constants used. For our results it seems not to change arguments significantly. Therefore, all our computations start with initial values 0 for the v_i. Nevertheless, the impact of different initializations seems an interesting problem to be analyzed further.

c) The final component of an input somehow is treated differently than all the others by the model. The reason is that after having read the final component only a test but no computation is performed. So in a certain sense an automaton is working only restrictedly with it. In some cases below we circumvent this effect by choosing a kind of dummy final component which will be the same for all inputs of a problem. That way the crucial parts of an input all are handled the same way by an automaton.

Since below we want to treat some elementary questions about such automata within the framework of real and complex BSS machines, we finish this section with the definition of a special non-deterministic complexity class in the BSS model that turns out to be interesting for analyzing non-deterministic generalized automata.

Definition 3 *(Digital non-determinism).* a) In the BSS-model over $\mathbb{R}$ a problem $L \subseteq \mathbb{R}^*$ belongs to the class $\mathrm{DNP}_{\mathbb{R}}$ of problems verifiable in polynomial time using digital non-determinism if there is a BSS algorithm M working as follows: M gets as its inputs tuples $(x, y) \in \mathbb{R}^* \times \{0,1\}^*$ and computes a result in $\{0,1\}$ interpreted as reject or accept, respectively. For an $x \in L$ there has to be an $y \in \{0,1\}^*$ such that $M(x,y) = 1$, for $x \notin L$ the result $M(x,y)$ has to be 0, no matter which y is chosen.

The running time of M has to be polynomial in the (algebraic) size of x.

b) Similarly, the complex counterpart $\mathrm{DNP}_{\mathbb{C}}$ is defined.

Digital non-determinism is a kind of restricted non-determinism in the real and complex BSS model. Here, usually the guess y is allowed to stem from $\mathbb{R}^*$ or $\mathbb{C}^*$ having polynomial length in the size of x. Most natural generalizations of discrete NP-complete problems to the real framework lead to problems in $\mathrm{DNP}_{\mathbb{R}}$; the Knapsack problem treated below is a typical example. However, there are important open questions related to this class. It is easy to see that $\mathrm{P}_{\mathbb{R}} \subseteq \mathrm{DNP}_{\mathbb{R}} \subseteq \mathrm{NP}_{\mathbb{R}}$, but it is currently only conjectured that both inclusions are strict. As a consequence, problems in $\mathrm{DNP}_{\mathbb{R}}$ are conjectured not to be $\mathrm{NP}_{\mathbb{R}}$-complete in the BSS model. The same is conjectured to be true for the classes $\mathrm{P}_{\mathbb{C}} \subseteq \mathrm{DNP}_{\mathbb{C}} \subseteq \mathrm{NP}_{\mathbb{C}}$.

3. Basic results: a structural theorem, a weak pumping lemma for complex automata, relation to semi-algebraic sets

Given the above definition of digital non-determinism our first result is immediate. The word problem for a fixed $(\mathcal{S}, k)$-automaton asks whether a given input $w \in D^*$ is accepted by the automaton.

Lemma 1. *The word problem belongs to class $\mathrm{P}_{\mathbb{R}}$ for deterministic $(\mathcal{S}_{\mathbb{R}}, k)$-automata and to class $\mathrm{DNP}_{\mathbb{R}}$ for non-deterministic such automata. This holds as well if the automaton is considered part of the input. It is analogously true for complex automata and the complex BSS model.*

Proof. For a deterministic automaton $\mathcal{A}$ using k registers and ℓ constants we only have to note that each step of such an automaton can be simulated by $O(k+\ell)$ operations of a BSS machine in the corresponding model. Since the automaton on input $w \in \mathbb{R}^n$ uses n steps, the running time of a BSS machine simulating the computation is polynomial in n in case $\mathcal{A}$ is fixed and polynomial in $n + \mathit{size}(\mathcal{A})$ otherwise.

A non-deterministic automaton by definition has a finite number of choices for a move given a state and the results of the test. This number depends on the automaton only. Therefore, asking for an accepting computation of an input $w \in \mathbb{R}^n$ can be coded using a discrete guess about the moves taken. The size of such a guess, i.e., the size of y in Definition 3 is polynomially bounded in $n + \mathit{size}(\mathcal{A})$ and the number of non-deterministic steps again is at most n. As above, this can be simulated by a non-deterministic BSS machine using digital non-determinism and running in polynomial time. The complex case follows from the same arguments. □

The previous result gives rise to several further questions which we shall treat next dealing with complex automata. Are all problems in $\mathrm{P}_{\mathbb{C}}$ acceptable by a non-deterministic $(\mathcal{S}_{\mathbb{C}}, k)$-automaton, are there any $\mathrm{DNP}_{\mathbb{C}}$ problems which potentially do not belong to $\mathrm{P}_{\mathbb{C}}$ but will be accepted by such an automaton, are non-deterministic automata closed under complementation?

We shall see that the answers to the above questions show a somewhat skew picture of the relation between acceptable languages in the generalized automata model and complexity classes in the BSS model.

Let us start with defining an extension of the classical NP-complete Knapsack problem.

Definition 4. The complex Knapsack problem $KS_{\mathbb{C}}$ is defined as follows: Given $n \in \mathbb{N}$ and n complex numbers $x_1, \ldots, x_n$, is there a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} x_i = 1$?

The real Knapsack problem $KS_{\mathbb{R}}$ is defined similarly.

Remark 2. Dealing with this problem using $(\mathcal{S}_{\mathbb{C}}, k)$-automata we consider inputs of the form $(x_1, \ldots, x_n, 1)$ of length $n+1$. This is done in order to guarantee that the numbers $x_1, \ldots, x_n$ are treated equally, see Remark 1 c) above. This could of course be done differently.

Note that the complexities of both $KS_{\mathbb{C}}$ and $KS_{\mathbb{R}}$ in the respective BSS model are unknown. This is the case for many reasonable extensions of classically NP-complete problems to the real or complex number model. On one hand such problems usually fall into the classes $\mathrm{DNP}_{\mathbb{K}}$ defined above, where $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. It is not known whether $\mathrm{DNP}_{\mathbb{K}}$ contains $\mathrm{NP}_{\mathbb{K}}$-complete problem for one of the settings (and actually conjectured to be false [9]), so the respective generalized Knapsack problems likely will neither be $\mathrm{NP}_{\mathbb{C}}$- nor $\mathrm{NP}_{\mathbb{R}}$-complete. On the other hand it is neither known whether problems being NP-complete in the Turing model can be solved more efficiently in the BSS model.

Example 1. $KS_{\mathbb{C}}$ can be accepted by a non-deterministic $(\mathcal{S}_{\mathbb{C}}, 1)$-automaton. The non-deterministic automaton uses 1 as its only constant and a single register v_1. When reading a new component x_i of the input the automaton non-deterministically chooses whether x_i should participate in the final sum or not. If it should, then x_i is added to v_1, otherwise v_1 remains unchanged. When $(x_1, \ldots, x_n)$ has been processed the automaton finally checks whether the last input component x_{n+1} equals 1. If not $\mathcal{A}$ rejects, if yes (i.e., $x_{n+1} = 1$) it is also compared with v_1. The automaton accepts iff $v_1 = 1$.

3.1. A structure theorem for complex automata

Our first major result will be a structural theorem concerning the languages accepted by complex deterministic and non-deterministic automata. As an easy consequence of this theorem it follows that the class of languages accepted by non-deterministic complex automata is not closed under complementation. For a particular restricted structure over the integers the corresponding result was shown in [7]. We prove it using topological arguments. More precisely, the Knapsack problem turns out to be a counterexample here. Before doing so we recall the definition of characteristic (or typical) paths for a BSS algorithm [2], adapted accordingly to $(\mathcal{S}_{\mathbb{C}}, k)$-automata. Characteristic paths are important since the sets of inputs that follow those paths have a useful topological structure.

Definition 5. a) Let $\mathcal{A}$ be a deterministic $(\mathcal{S}_{\mathbb{C}}, k)$-automaton using ℓ constants. Let $\mathcal{P}$ be a path of this automaton, i.e., a finite sequence $(q_0, q_1, \ldots, q_s)$ of states of $\mathcal{A}$ together with a sequence $(b^{(0)}, b^{(1)}, \ldots, b^{(s-1)})$ of test results in $\{0,1\}^{k+\ell}$ such that the automaton moves from q_i to q_{i+1} when $b^{(i)}$ represents the outcome of the tests. Here, q_0 denotes the start state of $\mathcal{A}$.

i) The path set $V_{\mathcal{P}}$ related to a path $\mathcal{P}$ of length s is the set of points in $\mathbb{C}^s$ that are branched by $\mathcal{A}$'s computation along $\mathcal{P}$.

ii) The characteristic path of length s of $\mathcal{A}$ is the one obtained if all $b^{(i)} = 0$, i.e., all $k + \ell$ equality tests performed at each step of the computation give result 'false'.

b) If $\mathcal{A}$ is a non-deterministic automaton any path that corresponds to a computation where all test results give $b^{(i)} = 0$ is called a characteristic path.

Note that characteristic paths always are realizable by some computation, i.e., the corresponding path sets are non-empty. This is true because at each step of a computation only a constant number of values are stored in the registers, so there is always a next complex input component being different from all of them.

Characteristic path sets have a very special structure. This structure is made more precise in the following definition.

Definition 6. For a set $L \subseteq \mathbb{C}^n, n \geq 1$ let $\mathcal{P}_1(L)$ denote the projection of L to the first component. Then L is called *recursively co-finite*, or *rcf* for short, if the following two conditions hold:

i) $\mathcal{P}_1(L)$ is co-finite;
ii) if $n > 1$ for any $x^* \in \mathcal{P}_1(L)$ the set

$$\{(x_2, \ldots, x_n) \in \mathbb{C}^{n-1} | (x^*, x_2, \ldots, x_n) \in L\}$$

is recursively co-finite.

In particular, a set $L \subseteq \mathbb{C}$ is rcf if it is co-finite.

We say that L is recursively co-finite *of cardinality* $s \in \mathbb{N}$ if the cardinalities of all the complements of projections involved in the above definition are less than or equal to s.

Now the following structural theorem can be proven.

Theorem 1. *Let $\mathcal{A}$ be an $(\mathcal{S}_{\mathbb{C}}, k)$-automaton with ℓ constant registers, $L(\mathcal{A}) \subseteq \mathbb{C}^*$ the language accepted by $\mathcal{A}$. For each $n \in \mathbb{N}$ let $L_n(\mathcal{A}) := L(\mathcal{A}) \cap \mathbb{C}^n$ and $\overline{L_n(\mathcal{A})}$ its complement in $\mathbb{C}^n$.*

a) If $\mathcal{A}$ is deterministic, then for each $n \in \mathbb{N}$ exactly one of the two sets $L_n(\mathcal{A})$ and $\overline{L_n(\mathcal{A})}$ contains a rcf set of cardinality at most $s := k + \ell$. In particular, for all n the cardinalities of the respective complements are bounded by a constant that is independent of n. Which of the two sets contains the rcf set can vary with n.

b) If $\mathcal{A}$ is non-deterministic, then there is a constant M such that for each $n \in \mathbb{N}$ exactly one of the two sets $L_n(\mathcal{A})$ and $\overline{L_n(\mathcal{A})}$ contains a rcf set of cardinality at most $O(M^n)$.

Moreover, for those n where the rcf set is contained in $L_n(\mathcal{A})$ the cardinalities of the respective complements can again be bounded by $s = k + \ell$.

Thus the difference between the statements for deterministic and non-deterministic automata is the cardinality bound for the rcf sets in $\overline{L_n(\mathcal{A})}$. In the non-deterministic case in general it cannot be bounded by a constant being independent of n.

Proof. a) Suppose first that $\mathcal{A}$ is deterministic. Fix n and consider the characteristic path γ_n of length n. Its path set V_{γ_n} is rcf with cardinality at most $s = k + \ell$; in each computational step i of $\mathcal{A}$ along the characteristic path if x_i denotes the current input component all but at most s many choices for $x_i \in \mathbb{C}$ will be branched further along γ_n. If path γ_n accepts, then $L_n(\mathcal{A})$ contains a rcf set, otherwise $\overline{L_n(\mathcal{A})}$ contains such a set.

It remains to show that only one of the two sets contains a rcf set. Towards this aim let $U \subset \mathbb{C}^n$ denote a subset of an arbitrary union of path sets of length n other than V_{γ_n}. We claim that U is not a rcf set. For any point $x \in U$ define $t(x) \in \{1, \ldots, n\}$ as index of the first component of x such that $\mathcal{A}$'s computation on x answers a test with $=$. Since by assumption $x \notin V_{\gamma_n}$ such an index exists. Now choose $x^* \in U$ with maximal value $t(x^*)$. For the point $(x_1^*, \ldots, x_{t(x^*)-1}^*)$ all tests so far have been answered by $\neq$, for component $x_{t(x^*)}^*$ an equality test is satisfied. If U would be rcf the projection of the set $\{(x_{t(x^*)}, \ldots, x_n) | (x_1^*, \ldots, x_{t(x^*)-1}^*, x_{t(x^*)}, \ldots, x_n) \in U\}$ to its first component has to be co-finite. Thus, there has to exist an $x_t \in \mathbb{C}$ such that all tests performed by $\mathcal{A}$ on input $(x_1^*, \ldots, x_{t(x^*)-1}^*, x_t)$ are answered negatively and $(x_1^*, \ldots, x_{t(x^*)-1}^*, x_t)$ can be extended to a point $\widehat{x}$ in U. However, for such a point $t(\widehat{x})$ would be larger than $t(x^*)$ thus contradicting our choice of x^*. It follows that U cannot be a rcf set.

b) Next, let $\mathcal{A}$ be non-deterministic. There are two cases to consider due to the fact that now several characteristic paths of a given length can exist. Let $n \in \mathbb{N}$ be fixed.

Case 1: All characteristic paths of length n reject. Then L_n has the same structure as in the deterministic case, i.e., it is the finite union of paths which are not characteristic. By the same argument as above L_n does not contain a rcf set.

Concerning the structure of $\overline{L_n}$ in Case 1 let $M \in \mathbb{N}$ denote the maximal number of non-deterministic choices automaton $\mathcal{A}$ can follow in one of its states. If φ is a characteristic path of $\mathcal{A}$ note that not necessarily $V_\varphi \subseteq \overline{L_n}$ since a point $x \in V_\varphi$ could be branched as well along an accepting non-characteristic path. Nevertheless, along a characteristic path in each step at most $k + \ell$ values are branched away from the path. There are at most M^n many characteristic paths of length n, thus at most $(k + \ell)M^n$ values for a fixed input component can be branched away from all characteristic paths (note that there might be inputs which can follow all characteristic paths of $\mathcal{A}$). It follows that $\overline{L_n}$ contains a rcf set of cardinality $O(M^n)$.

Case 2: There are characteristic paths which accept. Let φ denote one of them. Since the path accepts it follows $V_\varphi \subseteq L_n$. Clearly, V_φ is rcf of cardinality $k + \ell$. Finally, $\overline{L_n}$ does not contain a rcf set. Because if $U \subseteq \overline{L_n}$ would be rcf each point in U has to be branched away from the accepting characteristic path φ. The same argument as in the deterministic case gives a contradiction. □

We derive two further results from the theorem. Example 1 has shown that the Knapsack problem, which is conjectured not to be efficiently solvable in the BSS model, can be accepted by a non-deterministic automaton. However, the next result shows that easier problems cannot. Thus, the class of languages acceptable by non-deterministic (S, k)-automata is contained in $\mathrm{DNP}_\mathbb{R}$ or $\mathrm{DNP}_\mathbb{C}$, respectively, but lies kind of skew with respect to the class of languages decidable in polynomial time in the BSS model.

Corollary 1. *There are problems in complexity class $\mathrm{P}_\mathbb{C}$ which can not be accepted by any non-deterministic $(\mathcal{S}_\mathbb{C}, k)$-automaton. Similarly for $\mathrm{P}_\mathbb{R}$. As a consequence, the class of languages accepted by non-deterministic real or complex automata is strictly contained in $\mathrm{DNP}_\mathbb{C}$ and $\mathrm{DNP}_\mathbb{R}$, respectively.*

Proof. For the complex case define a language L as

$$L := \{(a_0, \ldots, a_n, x) \in \mathbb{C}^{n+2} | n \in \mathbb{N}, \sum_{i=0}^{n} a_i \cdot \left(x^{n^n}\right)^i = 0\}$$

By repeated squaring of x and subsequent evaluation of the univariate polynomial given through the a_i membership in L can be decided in the complex BSS model in polynomial time in n, i.e., $L \in \mathrm{P}_\mathbb{C}$. Now suppose a non-deterministic $(\mathcal{S}_\mathbb{C}, k)$-automaton $\mathcal{A}$ accepts L. As in the proof of Theorem 1 let M denote an upper bound on the number of non-deterministic choices in any state. For a canonical choice of $(a_0, \ldots, a_n) \in \mathbb{C}^{n+1}$ the polynomial $z \mapsto \sum_{i=0}^{n} a_i z^i$ has n different complex roots and each of them has n^n different n^n-th roots. Thus, there are $n \cdot n^n$ choices for x such that $(a_0, \ldots, a_n, x) \in L$. Now, on one hand $L \cap \mathbb{C}^{n+2}$ does not contain a rcf set since given $(a_0, \ldots, a_n) \neq 0$ there are always only finitely many choices for x yielding a point in L. On the other hand, for large enough n we have $n \cdot n^n > const \cdot M^n$ which contradicts as well part b) of Theorem 1. It follows that $\mathcal{A}$ cannot exist.

Over the real numbers define $L := \{(x_1, \ldots, x_n) | \sum_{i=1}^{n} x_i \in \{1, 2, \ldots, n^n\}\}$. Using binary search the problem easily is seen to belong to $\mathrm{P}_\mathbb{R}$. Suppose $\mathcal{A}$ is a non-deterministic $(\mathcal{S}_\mathbb{R}, k)$-automaton accepting L with at most $M \in \mathbb{N}$ choices in each of its states to choose non-deterministically a next state. Thus, for an input of length n the automaton can at most follow M^n different paths. At each step the tests can at most realize $O(k + \ell)$ different sign vectors. Thus, if we fix a part $(x_1, \ldots, x_{n-1})$ of the input, the set of $x_n \in \mathbb{R}$ such that $\mathcal{A}$ accepts $(x_1, \ldots, x_n)$ has at most $O(k + \ell) \cdot M^n$ many connected components, i.e.,

intervals and isolated points. But the corresponding projection of L can easily have n^n isolated points, for example if $\sum_{i=1}^{n-1} x_i$ does not belong to $\{1, 2, \ldots, n^n\}$. It follows that $\mathcal{A}$ cannot exist.

The last claim now follows from Lemma 1 and the containment of $\mathrm{P}_{\mathbb{R}}$ in $\mathrm{DNP}_{\mathbb{R}}$ and of $\mathrm{P}_{\mathbb{C}}$ in $\mathrm{DNP}_{\mathbb{C}}$. □

The above proof for the real case uses a certain structural property of accepted sets similar to rcf sets but seemingly weaker with respect to deriving interesting structural results. Therefore, we did not formulate it separately.

The following corollary shows another application of Theorem 1. Note that part a) follows from b), however we add the simple argument based on the previous theorem.

Corollary 2. *a) For all $k \in \mathbb{N}$ there is no deterministic $(\mathcal{S}_{\mathbb{C}}, k)$-automaton accepting $KS_{\mathbb{C}}$.*

b) For all $k \in \mathbb{N}$ there is no non-deterministic $(\mathcal{S}_{\mathbb{C}}, k)$-automaton accepting the complement of $KS_{\mathbb{C}}$, i.e., the set

$$\overline{KS_{\mathbb{C}}} = \{(x_1, \ldots, x_n, x_{n+1}) | n \in \mathbb{N} \text{ and either } x_{n+1} \neq 1 \text{ or } \forall S \subseteq \{1, \ldots, n\} \sum_{i \in S} x_i \neq 1\}.$$

c) The class of languages accepted by non-deterministic complex automata is stritcly larger than the class of languages accepted by deterministic such automata.

Proof. a) Our definition of $KS_{\mathbb{C}}$ requires as positive instances $n+1$-dimensional vectors whose final component is 1. Thus no rcf set can be a subset of $KS_{\mathbb{C}}$. According to the theorem the only characteristic path of a potential deterministic automaton has to contain an rcf set. Consider an input $(x_1, \ldots, x_{n-1}) \in \mathbb{C}^{n-1}$ such that all 2^{n-1} possible sums of components give a different result and such that the automaton follows the characteristic path when reading $(x_1, \ldots, x_{n-1})$. Clearly, such a sequence exists. Then there are 2^{n-1} many choices for x_n such that $(x_1, \ldots, x_n, 1) \in KS_{\mathbb{C}}$, but for component x_n the characteristic path can only branch away a constant number s of values for x_n. Thus, such an automaton cannot exist.

b) Suppose a non-deterministic automaton $\mathcal{A}$ accepts $\overline{KS_{\mathbb{C}}}$. The theorem implies that precisely one of the sets $KS_{\mathbb{C}}$ and $\overline{KS_{\mathbb{C}}}$ contains an rcf set. For $KS_{\mathbb{C}}$ this is not possible since the final component of an input in $KS_{\mathbb{C}}$ is forced to equal 1. For deriving as well a contradiction in the remaining case we need an additional argument. As in a) choose n large enough and $(x_1, \ldots, x_{n-1}) \in \mathbb{C}^{n-1}$ such that all 2^{n-1} possible sums of components give a different result, this time also different from 1. In addition we require that $\mathcal{A}$ follows for an infinite number of choices for x_n an accepting characteristic path when reading $(x_1, \ldots, x_{n-1})$. Note that fixing $(x_1, \ldots, x_{n-1})$ as above there is an infinite number of x_n such that $(x_1, \ldots, x_n, 1) \notin KS_{\mathbb{C}}$ and all such inputs must be accepted by $\mathcal{A}$. Thus the existence of such a path φ is guaranteed because there are only finitely many paths of a given length. Consider the two final computational steps of $\mathcal{A}$ when reading x_n and $x_{n+1} = 1$. Since $\mathcal{A}$ accepts for infinitely many choices of x_n along φ, for the final tests with $x_{n+1} = 1$ only those register values v_i have an influence that depend on the choice of x_n. Each of them, however, can only branch a single value away from φ. Since there are $2^{n-1} > k$ choices for x_n such that $(x_1, \ldots, x_n, 1) \in KS_{\mathbb{C}}$ the automaton can still branch most of them along φ and accept, thus leading to a contradiction.

Finally, claim c) directly follows since the class of languages accepted by a deterministic automaton clearly is closed under complementation. Example 1 together with parts a), b) imply the statement. □

Though the notion of characteristic path(s) makes sense for real automata as well it is not clear how to use it to obtain a meaningful structural result like Theorem 1. This is discussed a bit further in the final section. With respect to the real Knapsack problem, however, we can prove the same statement by applying well known results from algebraic complexity theory [1,12]. For sake of completeness we briefly include the corresponding proof below. Note that a much more general result for $KS_{\mathbb{R}}$ in the realm of the real BSS model has been shown in [6].

Proposition 1. *For all $k \in \mathbb{N}$ there is no non-deterministic $(\mathcal{S}_{\mathbb{R}}, k)$-automaton accepting the complement of $KS_{\mathbb{R}}$. Thus, the class of languages accepted by non-deterministic real automata is not closed under complementation. The real Knapsack problem is not accepted by a deterministic $(\mathcal{S}_{\mathbb{R}}, k)$-automaton.*

Proof. Suppose $\mathcal{A}$ to be a non-deterministic $(\mathcal{S}_{\mathbb{R}}, k)$-automaton for the complement. Consider inputs of length n. Due to the way our automata are working each computation is finished after $n+1$ steps (the $'+1'$ results from the dummy final component in our inputs). The proof now considers the computation of $\mathcal{A}$ as that of an algebraic decision tree, cf. [5], and applies known results to the latter. In fact, if $\mathcal{A}$ allows at most $M \in \mathbb{N}$ different non-deterministic choices at each step we can code each such choice by a number in the set $\{1, \ldots, M\}$ and consider $\mathcal{A}$ as an algebraic decision tree C_n working deterministically on inputs from $\mathbb{R}^{n+1} \times \{1, \ldots, M\}^n$. This tree clearly has a linear depth in n; the $k + \ell$ comparisons done at each step of the automaton increase the number of steps of the decision tree at most by acting as constant factor. To keep the following topological arguments clean without loss of generality we consider the Knapsack problem again in its original form of Definition 4, i.e., without having a final $(n+1)$-st component forced to equal 1. This is no restriction since any algebraic decision tree solving this problem solves the version used above for $(\mathcal{S}_{\mathbb{R}}, k)$-automata using at most one additional step.

Denote by A_n the set of inputs accepted by this tree, then $\overline{KS_{\mathbb{R}}} \cap \mathbb{R}^n$ is the projection of A_n to its first n many components. In particular, since the projection is continuous A_n has at least as many connected components as the complement. Now the claim follows by collecting known results about the number of connected components and lower bounds for the depth of algebraic decision trees. First, the number of connected components of $\overline{KS_{\mathbb{R}}} \cap \mathbb{R}^n$ is at least $2^{\Omega(n^2)}$, see [12]. Thus, this is also a lower bound for the number of connected components of A_n. Secondly, by [1] the depth of any algebraic decision tree for A_n is at least of order $\Omega(\log cc(A_n))$, where $cc(A_n)$ denotes the number of connected components of A_n. Putting both lower bounds together it follows that C_n must have a depth of order $\Omega(n^2)$, therefore any automaton $\mathcal{A}$ must perform a superlinear number of steps. This is not possible. The other claims follow directly as for the complex case. □

3.2. A weak pumping lemma

One major structural tool for establishing a language not to be regular in the classical finite automata framework is the pumping lemma. It is thus natural to ask whether a similar property holds for our generalized automata. However, a short consideration immediately implies that – if at all – such a statement has to be more involved. Consider the language $L := \{(x_1, 1, x_2, 1, \ldots, 1, x_n) \in \mathbb{C}^n | n \in \mathbb{N}, x_1 = 1, x_{i+1} = x_i + 1, 1 \leq i \leq n-1\}$, i.e., $(x_1, \ldots, x_n)$ represent initial segments of $\mathbb{N}$. L clearly is acceptable by a deterministic $(\mathcal{S}_{\mathbb{C}}, 1)$-automaton.[3] Now, if in a word $w \in L$ we pump any of its substrings the structure of the defining recursion formula for the components clearly is destroyed.

One major obstacle for obtaining a kind of pumping lemma is the ability to perform computations. Even if an automaton runs through a loop with respect to its state set it is by no means clear whether the loop is realizable even only once more repeating the same subsequence of input components. The reason is that in most cases the assignments of registers will change. And a different assignment clearly can result in a different computation path when reading the same part of an input repeatedly.

For complex automata and some loops it turns out that we can say a bit more. Here, once again the characteristic path of a deterministic automaton is helpful because many inputs follow it. We shall now show that a weak kind of pumping is possible. As drawback two features of the classical pumping lemma are lost. First, the pumping might not be possible for words in the language but for rejected words; and secondly, it cannot be guaranteed to hold for all words of a certain length. Nevertheless, we shall see that the statement can be used to show certain problems not being acceptable by deterministic complex automata.

Theorem 2. *Let $L \subseteq \mathbb{C}^*$ be accepted by a deterministic $(\mathcal{S}_{\mathbb{C}}, k)$-automaton $\mathcal{A}$. Then there is a word $w := uz \in \mathbb{C}^*$ such that either all $uz^t, t \in \mathbb{N}_0$ belong to L or they all belong to $\mathbb{C}^* \setminus L$. Moreover, u and z are vectors of at most K and $2K$ components, respectively, where K denotes the number of states of $\mathcal{A}$.*

Proof. Before going into detail we outline the main idea of the proof. We are looking for inputs that follow the characteristic path of $\mathcal{A}$ when the input dimension becomes larger. As mentioned earlier this path is realizable. For example, we could take a sequence of algebraically independent numbers. So there is a loop that can be realized as many times as we want. Let q denote the starting and final state of the first such loop; here, by first we mean first time the loop is completed. We fix this loop as the one we are interested in for the rest of the proof. Then there is a u of length at most K such that when reading u as its first input components $\mathcal{A}$ follows the characteristic path and enters q for a first time. The length of the loop is some $s \leq K$.

The problem, however, is that the above easy argument implies realizability of the characteristic path only when the input components can be changed all the time. Ad hoc there is no guarantee that we can follow the loop any given number of runs always taking the *same* complex vector $z \in \mathbb{C}^s$. The main task in the proof is to establish the existence of such a z. This will be done as follows: First, we show the existence of an open set $X \subseteq \mathbb{C}^s$ such that for each $x \in X$ automaton $\mathcal{A}$ on input ux follows the loop once. For this purpose we can use a sequence ux with the set of components being algebraically independent. In that case, no equality test will be answered positively, so ux follows the characteristic path. Since the test functions are continuous in the input components there is an open set X containing x such that for all $y \in X$ the input uy follows the characteristic path as well.

The main part of the proof now shows that for each additional run through the loop only a reasonably small set of points from X have to be removed because they might not be branched along the characteristic path when passing another time the loop.

Now towards the details. In the proof we restrict ourselves to an automaton $\mathcal{A}$ that uses no contants and a single register only. However, after it has been given it should be obvious that this is no restriction at all. We add a comment on this at the end.

Let u and $X \subseteq \mathbb{C}^s$ be as above. When $\mathcal{A}$ has read u it is in state q; let v^* denote the value of the register at that moment. For all $x \in X$ the computation on ux follows the characteristic path. Fix u and $x^* := (x_1^*, \ldots, x_s^*) \in X$ such that all components are algebraically independent. Our goal is to find a $z \in X$ such that uz^t for all $t \in \mathbb{N}_0$ follows the characteristic

[3] The intermediate 1's are used to avoid including the operation $+1$ in the structure; they could be removed if the operation is available.

path. To do so it must be guaranteed that for each run of $\mathcal{A}$ through the loop the current input component z_j never equals the current value in the register, for all $1 \le j \le s$. The latter of course can change with each new run through the loop. We thus have to analyze how the register value evolves.

Let us begin with some easy cases. First, if all computations performed during the loop are the projection pr_2 onto v, then v does not change. Since u and x^* have independent components all tests $x_j^* = v?$ are answered negatively and we are done. Secondly, suppose there is an operation $pr_1(x_j^*, v)$ performed and this is the only one that changes v, i.e., all other operations are pr_2. Then from this step on $v = x_j^*$ and in the next run through the loop the corresponding equality test is positive, so the computation leaves the characteristic path. This can easily be resolved replacing x^* by $x^*\tilde{x}^* \in \mathbb{C}^{2s}$ with all components independent (and thus different) and running twice through the loop. Now if $v = x_j^*$ in the next run $v = \tilde{x}_j^*$? will be tested with negative outcome; the projection pr_1 changes v's value into $\tilde{x}_j^*$. We then consider two consecutive runs through the loop as a new loop of double length. The only price to pay for this is the length of z in the theorem's statement which changes from at most K to at most $2K$.

Thirdly, if all operations are projections but different from the first two cases the statement trivially is correct. Finally, the case that projections occur but not exclusively is covered by the arguments that follow below. We therefore without loss of generality assume that during each step $j, 1 \le j \le s$ along the loop an arithmetic operation $v \circ_j x_j$ is performed. For sake of notational simplicity we only consider $\circ_j \in \{+, *\}$; subtractions do not change the arguments.

The way how v's value evolves during one sweep through the loop starting from initial value $v_t, t \ge 1$ can be described as follows:

$$v_{t+1} = [((v_t + a_1) * m_1 + a_2) * m_2 + \ldots + a_{s-1}] * m_{s-1} \circ_s x_s \tag{1}$$

Here, we have $a_i = x_i, m_i = 1$ in case $\circ_i = +$ and $a_i = 0, m_i = x_i$ if $\circ_i = *$. Moreover note that (1) includes s updates, one for each move along the states constituting the loop. The structure of the register value at intermediate steps can be easily extracted from the above formula, this will be used below.

Each additional run through the loop formally gives the same update starting from the respective value v_t. We now have to show that there is a point $z \in X$ such that all updates given by (1) are different from the respective components of z, no matter how often the loop is passed.

For all intermediate updates leading from $v_1 = v^*$ to v_2 this is true for all points in X. We now show for each $t \ge 1$ the following.

Claim. *Suppose $X^{(t)}$ is the subset of points $x \in X$ such that $\mathcal{A}$ for each input $ux^j, 0 \le j \le t$ follows the characteristic path and thus ends in state q. Then $X^{(t+1)}$ is obtained from $X^{(t)}$ by removing a set $R^{(t)}$ of points whose final component x_s belongs to a* finite *set.*

The claim implies the theorem: Since X is open, if at each loop such a set $R^{(t)}$ has to be removed, then $X \setminus \bigcup_{t \ge 1} R^{(t)}$ has a non-empty projection onto the s-th component. This is true since with respect to this component an at most countable set is removed from an interval. It follows that X contains a point z that follows the characteristic path for any given number of loops.

Proof of the claim. Suppose $x^* = (x_1^*, \ldots, x_{s-1}^*, x_s^*)$ is chosen from the open set X as explained before. Let us fix the first $s-1$ components and analyze for which values of x_s the input $u(x_1^*, \ldots, x_{s-1}^*, x_s)^t$ is branched along the loop for $t = 1, 2, 3, \ldots$ times. If $t = 1$ this is the case at least for x_s belonging to the open interval we obtain when projecting X to its final component.

Case 1: $\mathcal{A}$'s operation when reading x_s along the loop is an addition, i.e., $\circ_s = +$ and $v_{t+1} = f(v_t, x_1^*, \ldots, x_{s-1}^*) + x_s$ with f the appropriate function extracted from (1). In order to make sure that the computation follows the characteristic path all intermediate results given implicitly by (1) must be different from the respective component x_j^* and from x_s at the final step. This restricts the possible choices for x_s. The first condition when entering the loop for the next sweep is that $v \ne x_1^*$. This implies that only one value for x_s has to be avoided, namely $x_1^* - f(v_t, x_1^*, \ldots, x_{s-1}^*)$. By expanding the representation in (1) each intermediate result for the register value can easily be seen to be a degree one polynomial in x_s as variable. The coefficient of x_s is of the form $m_1^{\alpha_1} m_2^{\alpha_2} \ldots m_{s-1}^{\alpha_{s-1}}$ with some $\alpha_i = t - 1$ and the other $\alpha_i = t$, depending on where in the loop the computation currently resides. Thus the coefficient always is a product of some x_i^* with certain powers. The choice of the x_i^* as algebraically independent numbers guarantees this product to be always non-zero and different from 1. This implies that a comparison between the current value of v and the actual component $x_j^*, 1 \le j \le s-1$ always gives a negative result except for one assignment of x_s. This 'bad' value is the unique complex solution of a linear equation in x_s. The same holds for the final step in the loop and the comparison with x_s. As consequence, the computation continues to stay for one more step on the characteristic path.

Case 2: $\circ_s = *$ and $v_{t+1} = f(v_t, x_1^*, \ldots, x_{s-1}^*) * x_s$. A similar reasoning as before shows that if the computation runs for the t-th time through the loop ($t \ge 1$) the current register value is expressible as a polynomial of degree $t-1$ in x_s. More precisely, the highest coefficient, i.e., the coefficient of x_s^{t-1} has the form $f(v_t, x_1^*, \ldots, x_{s-1}^*) \cdot m_1^{\alpha_1} m_2^{\alpha_2} \ldots m_{s-1}^{\alpha_{s-1}}$. Here, again some $\alpha_i = t - 1$ and the other $\alpha_i = t$. Due to the choice of u and x^* the value $f(v_t, x_1^*, \ldots, x_{s-1}^*) \ne 0$ because f is a polynomial and there is no algebraic relation between the components. It follows that the comparison between the register value and one of the x_j^* or x_s only is positive for at most $t-1$ many choices of x_s. These choices have to be excluded in order to stay on the characteristic path.

The above reasoning shows that for each run through the loop all but a finite number of assignments to x_s are suitable in order to guarantee that the point $ux_1^* \dots x_{s-1}^* x_s$ is branched along the characteristic path of $\mathcal{A}$. Each such point is a suitable choice for uz. The Claim and thus the theorem follow.

Two final remarks are appropriate: If the automaton has k registers and ℓ constants the arguments apply in precisely the same way. Once again, only a finite number of values have to be forbidden for one of the variables x_i with respect to each register and each sweep through the loop. Moreover, for several registers it might be the case that instead of x_s another component has to be taken into account, for example, when one register value does not depend on x_s. This does not harm the above proof. □

An easy variation of the above argument also guarantees that suitable z can be chosen from an *open* set.

We end this section with an easy example showing how the weak pumping lemma can be applied. We are confident that other interesting examples can be treated that way as well.

Example 2. Consider the following modification of the Subset Sum problem. Define the language $L \subset \mathbb{C}^*$ to consist of all points $(x_1, \dots, x_n) \in \mathbb{C}^n$ such that there are two disjoint and non-empty sets $S_1, S_2 \subset \{1, \dots, n\}$ satisfying $\sum_{i \in S_1} x_i = \sum_{i \in S_2} x_i$. Then L cannot be accepted by a deterministic complex $(\mathcal{S}_{\mathbb{C}}, k)$-automaton. The proof of the weak pumping lemma implies that we can choose all components of u, z algebraically independent. Consequently, the input uz must be rejected since validity of the defining property for L implies an algebraic relation between the input components. But uz^2 clearly is an input in L since we can choose S_1 to cover the first occurrence of z and S_2 its second. This is not possible since it would imply the starting state of the loop to be at the same time accepting and rejecting. L thus cannot be accepted.

It is obvious that the same argument applies to any complex language L having the following structural property: for every $n \in \mathbb{N}$ language L contains vectors uz of algebraically independent components and of length $\geq n$, but not all uz^k with $k \in \mathbb{N}$ belong to L.

3.3. Semi-algebraic sets

Clearly, any set acceptable by a real $(\mathcal{S}_{\mathbb{R}}, k)$-automaton is the countable union of semi-algebraic sets, i.e., for each input dimension the accepted set is a finite Boolean combination of sets satisfying a system of polynomial equalities and inequalities. We shall now briefly argue that in a certain sense the converse holds as well. Of course, not every semi-algebraic set in $\mathbb{R}^n$ is acceptable by an automaton; Proposition 1 provides a counterexample. But at least each semi-algebraic set can be expressed as the projection of an acceptable set.

The proof idea for this statement actually is already hidden behind a result in [7]. However, the proof is not included in the conference version [7] and the statement therein does not obviously relate to the statement of Proposition 2 below, so we include the short proof adapted to our purposes.[4]

Recall that a basic semi-algebraic set in some $\mathbb{R}^n$ is a set of points satisfying finitely many polynomial (in)equalities. A semi-algebraic set is a finite union of basic semi-algebraic sets.

Proposition 2. *Let $S \subseteq \mathbb{R}^n$ be basic semi-algebraic defined by s polynomial (in)equalities. Then there exists a deterministic $(\mathcal{S}_{\mathbb{R}}, n + 2s)$-automaton $\mathcal{A}$ and a dimension m such that S is the projection of the set $L_m(\mathcal{A})$ onto its first n components.*

More precisely, $m = O(\ell \cdot d)$, where ℓ denotes the maximal number of monomials in each of the defining polynomials and d bounds their total degrees.

A similar statement holds for arbitrary semi-algebraic sets.

Proof. We restrict ourselves to outline the main idea only without including too much formalism describing the actual transitions of the automaton. First, consider a semi-algebraic set given via a single polynomial inequality $S := \{x \in \mathbb{R}^n \mid f(x) > 0\}$, where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a polynomial in variables $x_1, \dots, x_n$. The strict inequality can be replaced by any other relation present in $\mathcal{S}_{\mathbb{R}}$.

The idea taken from [7] is to simulate an evaluation of f in a given point $x \in \mathbb{R}^n$ by an $(\mathcal{S}_{\mathbb{R}}, n + 2)$-automaton step by step and finally check validity of the inequality. The only difficulty is the restricted storing capacity of the automaton which disallows to reuse inputs and intermediate results. In order to circumvent this limitation a larger space $\mathbb{R}^m$ has to be used. More precisely, take a representation of f as sum of ℓ monomials, each of the form $c_\alpha \cdot x^\alpha$, where $\alpha = (\alpha_1, \dots, \alpha_n) \in \mathbb{N}_0^n$, $x^\alpha := \prod_{i=1}^{n} x_i^{\alpha_i}$, and $c_\alpha \in \mathbb{R}$. The automaton uses registers $v_1, \dots, v_n$ to store the first n components $x \in \mathbb{R}^n$ of an input. It uses ℓ constant registers for the non-zero coefficients c_α of polynomial f. Now for each monomial indexed by α in a sequence of transitions the automaton checks whether the next part of its input corresponds to c_α, α_1 copies of x_1, α_2 copies of x_2 etc. If this is true the product $c_\alpha x^\alpha$ is evaluated factor by factor in register v_{n+1}: Start with copying the input

[4] We thank the authors for making available a draft of the full version of [7]; to the best of our knowledge this has not yet been published.

c_α to v_{n+1}, then multiply the latter with each new factor x_i seen in the current input component. Once the entire monomial has been evaluated (which happens after $\sum_{i=1}^{n} \alpha_i + 1$ steps) the value is added to register v_{n+2} and v_{n+1} is re-used to handle the next monomial. Finally, the automaton checks whether the result in v_{n+2} satisfies the inequality.

For a single polynomial condition the automaton needs $n+2$ registers and basically $O(\ell \cdot d)$ steps, where d is the total degree of f. Clearly, the automaton accepts an input if and only if the first n components give an $x \in S$ and the remaining part of the input is the assignment uniquely determined by the above process. Therefore, S is the projection of the set of accepted inputs to the first n dimensions.

If S consists of a conjunction of s many polynomial (in)equalities the automaton can be generalized using two more registers for each further condition that has to be checked; additionally, it has to use fixed registers for all coefficients occurring in one of the polynomials defining S.

Finally, for a general semi-algebraic set, i.e., a finite union of finite intersections of basic semi-algebraic sets, the construction can be extended because in [7] it is shown that the class of languages acceptable by deterministic $(\mathcal{S}_{\mathbb{R}}, k)$-automata is closed under finite unions and intersections. The bounds on the number of registers and the dimension of the set to be projected can be easily determined adapting the constructions in [7]. □

The design of the automaton obviously could be changed accordingly if another representation of f is chosen. This influences the number of necessary registers v_i. Above, this number basically is determined by the format $\sum\prod$ in which f is expressed. Note that the number of states in the above construction changes with the dimension n; this is due to the fact that each monomial is evaluated separately. Thus, the construction is not uniform in the sense that a single automaton can be designed for a family of semi-algebraic sets given in one or the other way uniformly (for example, via a BSS machine). It would be interesting to figure out whether at least for certain families of semi-algebraic sets such a uniform construction would be possible. It might as well be a promising future question to figure out whether this relation between semi-algebraic sets and recognizable languages could be used to obtain other interesting properties of the latter.

Finally, it is straightforward to see that for complex automata a similar proposition can be shown.

4. Undecidability results

We now turn to a bunch of undecidability results for the generalized automata model dealing with classical problems from finite automata theory. The basic of all these results is the following well known fundamental undecidability result for the BSS model, see [3].

Proposition 3. *The set $\mathbb{Q}^+$ of positive rational numbers is neither decidable in the real nor in the complex BSS model.*[5]

The undecidability results below are obtained by embedding the decidability question for the rationals into the problems under consideration. In all cases this will be done using in one or the other way a fundamental automaton that is described in the next result.

Proposition 4. *There is a deterministic $(\mathcal{S}_{\mathbb{R}}, 3)$-automaton $\mathcal{A}$ that accepts the language $L \subseteq \mathbb{R}^*$, defined as*

$$L := \{(r, x_1, x_2, \ldots, x_n, 0, t, s) | n, t, s \in \mathbb{N}, x_i \in \{-1, 1\},$$
$$s = \sum_{i, x_i = 1} x_i, t = \sum_{i, x_i = -1} |x_i|, r = \frac{s}{t}\}.$$

The automaton uses three constants $-1, 0, 1$.
L as well can be accepted by a deterministic $(\mathcal{S}_{\mathbb{C}}, 3)$-automaton when considered as language in $\mathbb{C}^*$.

Proof. Before describing $\mathcal{A}$ in more detail its way of functioning is outlined briefly. A tuple accepted by $\mathcal{A}$ as its first component must have a positive rational number r of form $\frac{s}{t}$. The correct values for s and t are determined by means of the intermediate components x_i which are used as counters: a value $x_i = 1$ is used by $\mathcal{A}$ to increase a counter for s by 1, $x_i = -1$ similarly is used for t. Those counters are realized in two of the registers of the automaton.

Now towards the details. From the following description it should be obvious how the automaton formally can be devised, so we do not specify each possible transition in detail. The automaton uses three registers v_1, v_2, v_3 that are initialized with 0. It uses as its constants $-1, 0, 1$ (this is not intended to be the minimal number possible to achieve the all-over goal). Any input that does not respect the formal constraints given in the definition of L is branched into a sink state. More precisely, the automaton checks all x_i to belong to $\{-1, 1\}$ by comparing a current x_i with the two constant

[5] We work with $\mathbb{Q}^+$ instead of $\mathbb{Q}$ for sake of simplicity below, not because of any particular importance using positive rationals only.

registers storing $-1, 1$. Similarly, $\mathcal{A}$ expects the sequence of x_i's to terminate reading a 0 component followed by two additional non-zero components s and t.

Let us then assume that an input satisfies these formal requirements (which of course can only be guaranteed after having read the entire input). $\mathcal{A}$ copies the first component r into register v_1. Now each time $\mathcal{A}$ reads a component $x_i = 1$ it adds the value 1 to register v_2, i.e., $v_2 \leftarrow v_2 + x_i$. Registers v_1 and v_3 are not changed in this case. Similarly, reading $x_i = -1$ register v_3 is increased by 1 using the operation $v_3 \leftarrow v_3 - x_i$ and v_1, v_2 remain unchanged. The first 0 read indicates that the automaton enters a new phase of its algorithm. Notice that if already $x_1 = 0$ the computation should end in a sink as well. In the next phase the automaton checks whether the numbers constructed so far in registers v_2, v_3 constitute a representation of r as fraction, thus yielding r to be a positive rational number. First $\mathcal{A}$ checks by a corresponding test whether $v_2 = t$. If not it moves into a sink state; otherwise t is a potential candidate for the correct denominator and the automaton performs the operation $v_1 \leftarrow v_1 \cdot t$. Then, it reads s and compares it to both v_3 and the updated value of v_1. Only if both these equality tests are satisfied the automaton runs into its unique accepting state, otherwise it moves again into a sink.

It is then obvious that $\mathcal{A}$ only accepts tuples of the corresponding form for which r is a positive rational and s, t represent a valid fraction for r. Since the automaton does not use inequality branches the algorithm works exactly the same in the complex model. □

The proposition immediately implies several undecidability results. Since the theorem deals with deterministic automata the corresponding problems are as well undecidable for non-deterministic automata. The size of an automaton can be taken as sum of its number of registers and number of states.

Theorem 3. *The following problems on $(\mathcal{S}_{\mathbb{R}}, 3)$-automata are undecidable in the real number BSS model. The analogue statements hold for complex automata and the complex BSS model; all $\mathcal{A}$ used below (except in part d) are deterministic $(\mathcal{S}_{\mathbb{R}}, 3)$-automata, q_0 denotes their respective initial state.*

a) Emptiness Problem: *Given $\mathcal{A}$, is $L(\mathcal{A}) = \emptyset$?*
b) Equivalence Problem: *Given two automata, do they accept the same language?*
c) Reachability Problem I: *Given $\mathcal{A}$ and a state p of $\mathcal{A}$, is there a computation of $\mathcal{A}$ that starts in q_0 and reaches p?*
d) Reachability Problem II: *There is an $(\mathcal{S}_{\mathbb{R}}, 4)$-automaton $\mathcal{A}$ (not part of the input) such that the following problem is undecidable: Given a state p of $\mathcal{A}$ and an assignment $v \in \mathbb{R}^k$ of the 4 registers of $\mathcal{A}$, is there a computation of $\mathcal{A}$ starting in its initial state with initialization $0 \in \mathbb{R}^4$ and leading to p attaining register values v?*
e) Minimization Problem: *Given $\mathcal{A}$, is it state minimal among all deterministic automata accepting $L(\mathcal{A})$?*
As consequence, there is no BSS algorithm minimizing any given generalized automaton.

Proof. All statements are implied by using suitable variants of the automaton constructed in Proposition 4.

For the emptiness problem consider as input an automaton $\mathcal{A}$ that uses in addition to constants $-1, 0, 1$ a constant $c \in \mathbb{R}$. This constant thus is part of the input and can be used to relate the emptiness problem with deciding the positive rationals. This can be done by modifying the automaton in Proposition 4 in such a way that in its first step it compares the first input component r with constant c. Only if $r = c$ the automaton continues to work as described in the proposition, otherwise it moves into a sink state. Now this $\mathcal{A}$ will accept a word if and only if c is rational. Thus deciding whether a given real number is a positive rational number can be reduced to deciding whether $L(\mathcal{A}) \neq \emptyset$. The latter problem is undecidable.

Claim b) is a direct consequence since one easily can construct an automaton that accepts no word from $\mathbb{R}^*$. Taking this automaton together with the one from a) as input the emptiness problem reduces to the equivalence problem.

Reachability problem I is easily seen to be undecidable as well using part a) since the automaton reaches its only accepting state iff the constant c is a positive rational.

Reachability problem II needs another modification of our standard automaton. It is necessary because now the automaton should be fixed, so we cannot code the rationals as decision problem by varying the automaton using different constants. Instead we code the rationals in the final desired register assignment as follows. First, recall that automaton $\mathcal{A}$ from Proposition 4 finishes an accepting computation on a tuple $(r, x_1, \ldots, x_n, 0, t, s)$ in its unique accepting state, say p, with register assignment $(r \cdot t, t, s)$. In that case $r = \frac{s}{t}$ is rational. However, we do not know in advance how s, t look like and whether they exist, so they cannot be used as the desired assignment for an instance of Reachability Problem II. Therefore, $\mathcal{A}$ is modified as follows giving a new $(\mathcal{S}_{\mathbb{R}}, 4)$-automaton $\mathcal{A}'$. This automaton uses one additional register in order to store twice the first component r read. The second copy is stored in register v_4 and this register will not be changed any more during the rest of the computation of $\mathcal{A}'$. If $\mathcal{A}$ has reached its final state p, then $\mathcal{A}'$ continues its computation requiring one additional 0-component as remaining input and using the projection operation to set registers $v_1 = v_2 = v_3 = 0$. The only accepting state of $\mathcal{A}'$ is a new state p' and it can only be reached from p in the above described way. If this is the case, then the four registers of $\mathcal{A}'$ have the assignment $(0, 0, 0, r)$, where r is the rational leading as first component of an input to the above final configuration. Thus, for the fixed automaton $\mathcal{A}'$ there is a computation leading from q_0 to state p' and resulting in a register assignment $(0, 0, 0, r)$ iff $r \in \mathbb{Q}^+$. It follows that the second version of the reachability problem is undecidable as well.

Finally, the minimization problem clearly cannot be computable for deterministic generalized automata; if it were one could decide the emptiness problem since a minimal automaton for the empty set has one state only. □

4.1. Conclusion and open questions

In this paper we have studied the generalized model of finite automata introduced in [7] in the framework of BSS computability and complexity. The focus has been on real and complex number computability. Our results show that a lot of classical questions about finite automata in the generalized framework have different answers. Among them we find both different complexity and computability results. In addition, they lead to a lot of further open questions, a few of which are outlined below.

Another kind of reachability problem than those of Theorem 3 was studied in [7]. There, the question is whether given a (non-deterministic) automaton and a computation path there is an input such that the automaton follows the given path with its computation. One of the main results in [7] is that this problem is decidable. Since the path is part of the instance there is a finite number of steps to be performed, i.e., the dimension of a suitable input $x \in \mathbb{R}^*$ for the automaton's computation to realize the path is given. Then the problem translates into an existential formula in first order logic over the reals. The formula just asks for the existence of an input realizing the required computational steps. Thus the problem is decidable by quantifier elimination. The same holds over $\mathbb{C}$. The difference with the above Reachability problem II is that we do not know in advance (a bound for) the length of a potential accepting path. The problem thus looks a bit similar to the real Halting Problem [3]. It would then be interesting to analyze whether reachability problems can be of the same degree of undecidability than the real Halting Problem. Note, however, that the rationals are known to be of a weaker degree of undecidability [11]. This question seems interesting also from the BSS side since not many problems are known that are of the same difficulty of the Halting Problem, see [10] for one such.

Theorem 3 gives as well rise to investigate the limits of the respective undecidability results, i.e., for which kind of restricted automata some of the problems might turn out to be decidable. Here, restrictions for example can apply to the number of registers and/or the number of constants used by the automaton. One easy result into this direction is the following.

Lemma 2. *For $(\mathcal{S}_{\mathbb{C}}, 1)$-automata and $(\mathcal{S}_{\mathbb{R}}, 1)$-automata that use no constants the emptiness problem is decidable in polynomial time in the size of the automaton over the corresponding structure.*

Proof. At each step of a computation there is only one equality test performed which compares the current input component from $\mathbb{C}$ with the current register value. Both outcomes of the test thus are possible and all transitions can be realized. The question whether an accepting state can be reached thus reduces to the question whether there is a path in the automaton's transition graph leading from the initial state to one of the accepting states. The argument for real automata is the same. □

Note that the above problem is purely discrete if the initial configuration contains no complex data. This holds as well if the automaton has k registers but no constants. Since all purely discrete problems are decidable in the BSS model [3] (though not necessarily in polynomial time) restrictions of the problems treated in Theorem 3 only become interesting if either constants are present or the initial configuration is part of the input as well. This of course does not apply to the second reachability problem since here the final register values are part of an instance. In general, we could also wonder about the impact arbitrary initial assignments to the registers have, for example, with respect to the interplay with the set of constants used.

Next, we have seen that the word problem for non-deterministic real automata belongs to complexity class $\mathrm{DNP}_{\mathbb{R}}$. On the other hand it is likely not solvable in polynomial time because it covers problems like $KS_{\mathbb{R}}$. So the question is whether it is $\mathrm{DNP}_{\mathbb{R}}$-complete under polynomial time reductions? Similarly for the complex case.

Other open questions relate to the weak pumping lemma and further structural properties of languages accepted. Is there a similar result for real automata? One can easily define characteristic paths in the real setting as well. Instead of requiring all tests to give a negative answer one could demand that the tests establish the current input component to be larger (or smaller) than all values it is compared to. This conditions even could be mixed with changing states. However, it is not clear to us whether a meaningful statement about the evolvement of register values could be deducted for computations along such real characteristic paths. Another problem related of course would be a stronger pumping lemma, i.e., one dealing with *accepting* computations. Once again, a main difficulty here seems to be to control the register values. And even more ambitious: What's about a Myhill–Nerode like characterization of languages accepted by $(\mathcal{S}, k)$-automata? Though Theorem 3 indicates that such a result would likely look very different from the classical one, since it might not result in computable properties like state minimization, it would certainly be interesting to find such characterizations.

References

[1] M. Ben-Or, Lower bounds for algebraic decision trees, in: Proc. 15th ACM STOC, 1983, pp. 80–86.
[2] L. Blum, F. Cucker, M. Shub, S. Smale, Complexity and Real Computation, Springer, 1998.

[3] L. Blum, M. Shub, S. Smale, On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, Bull. Amer. Math. Soc. (N.S.) 21 (1989) 1–46.
[4] O. Bournez, M.L. Campagnolo, A survey on continuous time computations, in: B. Cooper, B. Löwe, A. Sorbi (Eds.), New Computational Paradigms. Changing Conceptions of What is Computable, Springer-Verlag, New York, 2008, pp. 383–423.
[5] P. Bürgisser, M. Clausen, M.A. Shokrollahi, Algebraic Complexity Theory, Grundlehren, vol. 315, Springer, 1997.
[6] F. Cucker, M. Shub, Generalized Knapsack problems and fixed degree separation, Theoret. Comput. Sci. 161 (1996) 301–306.
[7] A. Gandhi, B. Khoussainov, J. Liu, Finite automata over structures. Extended abstract, in: M. Agrawal, S.B. Cooper, A. Li (Eds.), Proc. 9th Annual Conference on Theory and Applications of Models of Computation, TAMC, Beijing, 2012, in: Springer LNCS, vol. 7287, 2012, pp. 373–384.
[8] S. Haykin, Neural Networks – A Comprehensive Foundation, 2nd edition, Prentice Hall, 1999.
[9] K. Meer, On the complexity of quadratic programming in real number models of computation, Theoret. Comput. Sci. 133 (1) (1994) 85–94.
[10] K. Meer, M. Ziegler, Real computational universality: the word problem for a class of groups with infinite presentation, Found. Comput. Math. 9 (5) (2009) 599–609.
[11] K. Meer, M. Ziegler, An explicit solution to Post's problem over the reals, J. Complexity 24 (1) (2008) 3–15.
[12] F. Meyer auf der Heide, Lower bounds for solving linear diophantine equations on random access machines, J. ACM 32 (4) (1985) 929–937.
[13] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2000.
[14] G. Paun, Membrane Computing: An Introduction, Springer, 2002.
[15] K. Weihrauch, Computable Analysis: An Introduction, Springer, 2000.