IFIG
RESEARCH
REPORT

INSTITUT FÜR INFORMATIK

DETERMINISTIC SET AUTOMATA

Martin Kutrib Andreas Malcher Matthias Wendlandt

IFIG RESEARCH REPORT 1402
APRIL 2014

Institut für Informatik
JLU Gießen
Arndtstraße 2
35392 Giessen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

IFIG Research Report
IFIG Research Report 1402, April 2014

Deterministic Set Automata

Martin Kutrib,[1] Andreas Malcher,[2] and Matthias Wendlandt[3]

Institut für Informatik, Universität Giessen
Arndtstraße 2, 35392 Giessen, Germany

Abstract. We consider the model of deterministic set automata which are basically deterministic finite automata equipped with a set as an additional storage medium. The basic operations on the set are the insertion of elements, the removing of elements, and the test whether or not an element is in the set. We investigate the computational power of deterministic set automata and compare the language class accepted with the context-free languages and classes of languages accepted by queue automata. As results the incomparability to all classes considered is obtained. In the second part of the paper, we examine the closure properties of the class of DSA languages under Boolean operations. Finally, we show that deterministic set automata may be an interesting model from a practical point of view by proving that their emptiness problem is decidable.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Decision problems*;

Additional Key Words and Phrases: set automata, closure properties, emptiness problem, decidability

[1]E-mail: kutrib@informatik.uni-giessen.de
[2]E-mail: malcher@informatik.uni-giessen.de
[3]E-mail: matthias.wendlandt@informatik.uni-giessen.de

1 Introduction

The regular languages and their corresponding automata model, deterministic and nondeterministic finite automata, are well investigated [6]. It is well known that this family of languages has many desirable properties. For example, all commonly studied decidability questions are decidable and the regular languages are closed under almost all commonly studied operations such as, for example, the Boolean operation, concatenation, (inverse) homomorphism, and substitution. From a practical point of view, finite automata are in particular interesting, since many of the decidability questions are decidable in polynomial time and, in addition, an effective minimization algorithm is known for deterministic finite automata.

But with respect to the computational power, this model is quite weak since it builds the lowermost level of the Chomsky hierarchy. Hence, much efforts have been made to find models that extend the computational power of regular languages by adding storage media, but keep as many 'good' properties as possible. Consider, for example, the extension by a stack [4] or by a pushdown store [2], which leads to the context-free languages. For both models nondeterministic variants are more powerful than deterministic variants, which is in contrast to finite automata. Moreover, some positive closure properties are lost. On the other hand, the decidability of emptiness and finiteness is preserved [4, 6, 9]. In addition, equivalence is decidable for deterministic pushdown automata [10], but not for the nondeterministic variant [6].

Another extension studied is that of a queue. In their general definition queue automata can accept the class of recursively enumerable languages for which all non-trivial decidability questions are undecidable. A meaningful restriction for queue automata is considered in [1] where quasi-real-time computations are studied. Another restriction is investigated in [7] where the number of turns, that is, the changes between an enqueuing and a dequeuing phase, is bounded by some constant. Both restrictions lead to language classes less powerful than the class of recursively enumerable languages. With the latter restriction it is possible to decide the emptiness problem [7].

The paper [3] introduces bag automata which are basically finite automata equipped with a finite number of bags in which the automaton can put symbols and also multiple versions of symbols. The symbols are stored as multisets and, therefore, the order in which they are added to the bags is not remembered. This model is quite powerful, because it is possible to simulate certain counter machines. If the model is restricted to so-called well-formed bag automata, a language class in between the (deterministic) one-counter and the (deterministic) context-free languages is obtained.

In this paper, we consider *deterministic set automata* (DSA) that extend deterministic finite automata by adding the storage medium of a set which, in contrast to bag automata, allows words to be stored and is not a multiset. As operations on the set it is possible to add elements, to remove elements, and to test whether some element is in the set. To prepare a set operation the DSA can write on a one-way write-only tape. For the set operation the contents of that tape are taken and added to the set, removed from the set, or tested. After the set operation, the writing tape is reset to the empty tape and a new set operation may be prepared. A similar model has been introduced by Lange and Reinhardt in [8]. In contrast to DSA, their model may work nondeterministically, allows no remove operations, and a test operation implicitly adds the word tested to the set. The main results in [8] are the decidability of emptiness for the model considered and the closure of the corresponding language class under the operations homomorphism, inverse homomorphism, and intersection with regular languages.

This paper is organized as follows. After the definition of the model and some examples in Section 2, we compare DSA with pushdown automata, quasi-real-time queue automata, and queue automata with finite turns with regard to their computational power. As result we obtain the incomparability with all classes investigated. This shows that DSA can accept languages which are not accepted by the other models. In Section 4 we consider closure properties. It turns out that the language class accepted by DSA is closed under complement and union with regular languages as well as intersection with regular languages, but is not closed under general union and general intersection. Finally, we show that emptiness is decidable for DSA which is a pleasant property from a theoretical as well as from a practical point of view.

2 Preliminaries

We write Σ^* for the set of all words over the finite alphabet Σ. The empty word is denoted by λ, and we set $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The reversal of a word w is denoted by w^R, and for the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions.

A set automaton is a system consisting of a finite state control, a one-way writing tape where transductions of parts of the input can be temporarily stored, and a data structure *set* where words of arbitrary length can be stored. At each time step, it is possible to either write a transduction of the current input letter to the end of the writing tape, to insert or remove the word written on the tape to or from the set, or to test whether the word written on the tape belongs to the set. Each time a set operation {`in`, `out`, or `test`} is done, the content of the writing tape is erased and its head is reset to the left end.

Definition 1. *A* deterministic set automaton*, abbreviated as* DSA*, is a system* $M = \langle S, \Sigma, \Gamma, \triangleleft, \delta, s_0, F\rangle$*, where*

1. S *is the finite set of* internal states*,*
2. Σ *is the finite set of* input symbols*,*
3. Γ *is the finite set of* tape symbols*,*
4. $\triangleleft \notin \Sigma$ *is the* right endmarker*,*
5. $s_0 \in S$ *is the* initial state*,*
6. $F \subseteq S$ *is the set of* accepting states*, and*
7. $\delta : S \times (\Sigma \cup \{\lambda, \triangleleft\}) \to (S \times (\Gamma^* \cup \{\mathtt{in}, \mathtt{out}\})) \cup (S \times \{\mathtt{test}\} \times S)$ *is the partial transition function, where* `in` *is the instruction to add the content of the tape to the set,* `out` *is the instruction to remove the content of the tape from the set, and* `test` *is the instruction to test whether or not the content of the tape is in the set. If the transition function is defined for some pair* (s, λ) *with* $s \in S$*, then it is not defined for any pair* (s, a) *with* $a \in \Sigma \cup \{\triangleleft\}$*.*

A *configuration* of a DSA $M = \langle S, \Sigma, \Gamma, \lhd, \delta, s_0, F\rangle$ is a quadruple $(s, v, z, \mathbb{S})$, where $s \in S$ is the current state, $v \in \{\Sigma^*\lhd\} \cup \{\lambda\}$ is unread part of the input, $z \in \Gamma^*$ is the content of the tape, and $\mathbb{S} \subseteq \Gamma^*$ is the finite set of stored words. The *initial configuration* for an input string w is set to $(s_0, w\lhd, \lambda, \emptyset)$. During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $s, s', s'' \in S$, $x \in \Sigma \cup \{\lambda, \lhd\}$, $v \in \{\Sigma^*\lhd\} \cup \{\lambda\}$, $z, z' \in \Gamma^*$, and $\mathbb{S} \subseteq \Gamma^*$. We set

1. $(s, xv, z, \mathbb{S}) \vdash (s', v, zz', \mathbb{S})$, if $\delta(s, x) = (s', z')$,
2. $(s, xv, z, \mathbb{S}) \vdash (s', v, \lambda, \mathbb{S} \cup \{z\})$, if $\delta(s, x) = (s', \texttt{in})$,
3. $(s, xv, z, \mathbb{S}) \vdash (s', v, \lambda, \mathbb{S} \setminus \{z\})$, if $\delta(s, x) = (s', \texttt{out})$,
4. $(s, xv, z, \mathbb{S}) \vdash (s', v, \lambda, \mathbb{S})$, if $\delta(s, x) = (s', \texttt{test}, s'')$ and $z \in \mathbb{S}$,
5. $(s, xv, z, \mathbb{S}) \vdash (s'', v, \lambda, \mathbb{S})$, if $\delta(s, x) = (s', \texttt{test}, s'')$ and $z \notin \mathbb{S}$.

We denote the reflexive and transitive closure of $\vdash$ by $\vdash^*$. It should be noted that an instruction to remove some z from $\mathbb{S}$ does not test whether $z \in \mathbb{S}$; it only ensures that $z \notin \mathbb{S}$ after the operation. The language accepted by the DSA M is the set $L(M)$ of words for which there exists a computation beginning in the initial configuration and ending in a configuration in which the whole input is read and an accepting state is entered. Formally:

$$L(M) = \{\, w \in \Sigma^* \mid (s_0, w\lhd, \lambda, \emptyset) \vdash^* (s_f, \lambda, z, \mathbb{S}) \text{ with } s_f \in F, z \in \Gamma^*, \mathbb{S} \subseteq \Gamma^* \,\}.$$

The family of all languages accepted by DSA is denoted by $\mathscr{L}(\text{DSA})$.

Example 2. Language $L_1 = \{\, wcw \mid w \in \{a, b\}^* \,\}$ is accepted by a DSA. The idea is to read the whole sequence up to the letter c and to copy it to the tape. When the input head arrives at the c, it stores the word w on the tape in its set. Then it reads the second subword consisting of a's and b's and copies it also to the tape. When the input head arrives at the right endmarker, it tests whether the content on the tape is in the set. If this is the case, then the input is accepted and otherwise rejected. □

Example 3. Language $L_2 = \{\, a^n b^m \$_0 c^n \mid m, n \geq 1 \,\} \cup \{\, a^n b^m \$_1 c^m \mid m, n \geq 1 \,\}$ is accepted by a DSA. First, the automaton writes for every a in the input an a on the tape. When the first b is read, it adds the word on the tape to the set. Then the automaton writes for every b in the input an b on the tape until the dollar is in the input. Subsequently, the word on the tape, consisting of b's, is added to the set. Depending on whether there has been a $\$_0$ or a $\$_1$ in the input the automaton writes an a or a b for each c in the input on the tape. In the last step, the automaton checks whether the word on the tape is in the set. If the test is successful, the input is accepted and otherwise rejected. □

Example 4. Language $L_3 = \{\, a^n b^n c^n \mid n \geq 1 \,\}$ is accepted by a DSA in such a way that it writes for every a in the input an a on the tape. When it reads the first b, it adds the content of the tape to the set. Then, it writes for every b in the input an a on the tape and when it reads the first c, it tests whether the word on the tape is in the set. If this is not the case, the input is rejected. Otherwise, for every c in the input an a is written on the tape. If at the end again the word on the tape is in the set, then the input is accepted and otherwise rejected. □

3 Computational Power of Deterministic Set Automata

In this section, we study the computational power of deterministic set automata. Hence, we compare the model with the known models of pushdown automata and queue automata. Since in general queue automata characterize the recursively enumerable languages, we compare our model with the restricted versions of quasi-real-time queue automata and queue automata with a finite number of turns, that is, the number of changes between enqueuing and dequeuing periods is bounded by a fixed number.

Let us first consider unary languages. It is known that pushdown automata accept only semilinear unary languages, hence regular languages, whereas even quasi-real-time queue automata may accept non-semilinear unary languages [1].

Theorem 5. *Every unary language accepted by a* DSA *is semilinear.*

Proof. Let $M = \langle S, \{a\}, \Gamma, \lhd, \delta, s_0, F\rangle$ be a DSA accepting a unary language and k be the length of a longest word that M can write in one step on the tape. We may assume that M accepts an infinite language, since finite unary languages are semilinear. Let w be an input such that $|w| > |S|$. When processing this input, the automaton necessarily has to enter a loop. We consider two cases.

First, we assume that there is no situation occurring in which the automaton performs an operation $\{\texttt{in}, \texttt{out}, \texttt{test}\}$ on the set. Then, there will never be a word in the set and M can be easily transformed into an equivalent deterministic finite automaton. Second, we assume that M performs an `in`-, `out`, or `test`-operation after which the content of the tape is deleted. In each computation step, M can write at most k symbols on the tape. Due to the unary input, M can distinguish between at most $|S|$ different situations. Thus, the words written on the tape and possibly added to the set are at most of length $k \cdot |S|$. Hence, we can construct a deterministic finite automaton that simulates M by storing the content on the tape as well as the finite number of words in the set in its state. Since the languages accepted by finite automata are semilinear, the theorem follows. □

With this result we are able to show that the family of languages accepted by deterministic set automata is incomparable with the family of languages accepted by quasi-real-time queue automata.

Theorem 6. *The family of languages accepted by* DSA *is incomparable with the family of languages accepted by quasi-real-time queue automata.*

Proof. The non-semilinear unary language $\{\, a^n \mid n \text{ is a Fibonacci number}\,\}$ is accepted by some quasi-real-time queue automaton [1]. Since by Theorem 5, DSA do not accept non-semilinear unary languages, it remains to show the other direction. The witness language is language L_2 of Example 3 and it is shown in the Appendix that L_2 is not accepted by any quasi-real-time queue automaton. □

In the next proof as well as in the section for the decidability we need that set automata are in a special form where each state carries the information whether the last action on the set was a `test`-, `in`-, or `out`-operation or was a write operation on the tape. Additionally, it is distinguished between successful and unsuccessful `test`-operations.

Definition 7. *A* DSA M *is in* action normal form*, if the initial state of M is only visited once at the beginning of the computation and each other state indicates uniquely which action the automaton M did in the last computation step. The states are marked with a corresponding subscript* `test+`*,* `test-`*,* `in`*, or* `out`*. Non-marked states are interpreted as states where the last action was a write operation on the tape.*

Lemma 8. *Any* DSA $M = \langle S, \Sigma, \Gamma, \lhd, \delta, s_0, F\rangle$ *can be converted into an equivalent* DSA $M' = \langle S', \Sigma, \Gamma, \lhd, \delta', s_0', F'\rangle$ *in action normal form.*

Proof. In a first step, we construct a DSA $M'' = \langle S'', \Sigma, \Gamma, \lhd, \delta'', s_0'', F''\rangle$ with a new initial state s_0'' which is visited at most once. Let $S'' = S \cup \{s_0''\}$ with $s_0'' \notin S$. If $s_0 \in F$ we define $F'' = F \cup \{s_0''\}$, and set $F'' = F$ otherwise. The transition function δ'' is defined as follows for $s_i, s_j, s_k \in S$, $a \in \Sigma \cup \{\lambda, \lhd\}$, and $z \in \Gamma^* \cup \{\mathtt{in}, \mathtt{out}\}$.

1. $\delta''(s_0'', \lambda) = (s_0, \lambda)$,
2. $\delta''(s_i, a) = (s_j, z)$, if $\delta(s_i, a) = (s_j, z)$,
3. $\delta''(s_i, a) = (s_j, \mathtt{test}, s_k)$, if $\delta(s_i, a) = (s_j, \mathtt{test}, s_k)$.

In the next step, we define a state set consisting of five pairwise disjoint sets $S_{\mathtt{in}}$, $S_{\mathtt{out}}$, $S_{\mathtt{test+}}$, $S_{\mathtt{test-}}$, and $S_{\mathtt{write}}$. The idea is that we introduce a new state for every state connected with a non-writing operation, whereas the states of S'' indicate writing operations. So, let $S_{\mathtt{write}} = S''$, $S_{\mathtt{test}} = S_{\mathtt{test+}} \cup S_{\mathtt{test-}}$ and $S' = S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}} \cup S_{\mathtt{write}}$. We set $s_0' = s_0''$ and $F' = F''$. For the definition of δ', consider $s_i, s_j, s_k \in S''$, $a \in \Sigma \cup \{\lambda, \lhd\}$, and $z \in \Gamma^*$.

1. $\delta'(s_i, a) = (s_{j_{\mathtt{in}}}, \mathtt{in})$ and $\delta'(s_{j_{\mathtt{in}}}, \lambda) = (s_j, \lambda)$, if $\delta''(s_i, a) = (s_j, \mathtt{in})$,
2. $\delta'(s_i, a) = (s_{j_{\mathtt{out}}}, \mathtt{out})$ and $\delta'(s_{j_{\mathtt{out}}}, \lambda) = (s_j, \lambda)$, if $\delta''(s_i, a) = (s_j, \mathtt{out})$,
3. $\delta'(s_i, a) = (s_{j_{\mathtt{test+}}}, \mathtt{test}, s_{k_{\mathtt{test-}}})$, $\delta'(s_{j_{\mathtt{test+}}}, \lambda) = (s_j, \lambda)$, and $\delta'(s_{k_{\mathtt{test-}}}, \lambda) = (s_k, \lambda)$, if $\delta''(s_i, a) = (s_j, \mathtt{test}, s_k)$,
4. $\delta'(s_i, a) = (s_j, z)$, if $\delta''(s_i, a) = (s_j, z)$.

The DSA M' is still a deterministic automaton, since newly introduced λ-transitions start only from newly introduced states. Moreover, M' is in action normal form and is equivalent to M, since the same transitions as in M are performed and the additional λ-transitions do not affect the language accepted. □

Our next goal is to achieve another normal form for DSA which plays a crucial role in further proofs. Let $M = \langle S, \Sigma, \Gamma, \lhd, \delta, s_0, F\rangle$ be a DSA in action normal form. Thus, $S = S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}} \cup S_{\mathtt{write}}$. We note that the tape is empty at the beginning of the computation as well as after each operation on the set. Now we build sets of the form L_{s_i,s_j} with $s_i \in \{s_0\} \cup S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}}$ and $s_j \in S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}}$ that describe all words that can be written on the tape when the computation starts in state s_i with empty tape and ends in state s_j, and in between no other state performing an operation on the set is entered. Formally we define

$$\begin{aligned} L_{s_i,s_j} = \{\, w \in \Gamma^* \mid {} & \text{there is } u \in \Sigma^* \text{ such that } (s_i, u, \lambda, \mathbb{S}) \vdash (s_{i+1}, u_1, w_1, \mathbb{S}) \\ & \vdash^* (s_{i+(n-1)}, u_{n-1}, w_{n-1}, \mathbb{S}) \vdash (s_{i+n}, u_n, w_n, \mathbb{S}) \vdash (s_j, \lambda, \lambda, \mathbb{S}'), \\ & \text{and } s_{i+1}, s_{i+2}, \ldots, s_{i+n} \notin S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}} \,\}. \end{aligned}$$

All these regular sets L_{s_i,s_j} can be built from M.

We say that a DSA M is in *infinite action normal form* if M is in action normal form and all sets L_{s_i,s_j} are infinite. The next lemma, whose proof may be found in the Appendix, says that we always may assume that a DSA is in infinite action normal form.

Lemma 9. *Any* DSA M *can be converted into an equivalent* DSA M' *in infinite action normal form.*

The context-free languages and their important subclass of deterministic context-free languages are one of the best studied families of languages.

Theorem 10. *The family of languages accepted by* DSA *is incomparable with the (deterministic) context-free languages.*

Proof. By Example 2, the non-context-free language $L_1 = \{\, wcw \mid w \in \{a,b\}^* \,\}$ is accepted by some DSA. So, it suffices to show that the deterministic context-free language $L = \{\, wcw^R \mid w \in \{a,b\}^* \,\}$ is not accepted by any deterministic set automaton. The detailed proof may be found in the Appendix. □

Next, we derive the incomparability of the family of languages accepted by deterministic set automata with the family of languages accepted by queue automata with finite turns as follows.

Theorem 11. *The family of languages accepted by* DSA *is incomparable with the family of languages accepted by queue automata with finite turns.*

Proof. We consider language L_3 of Example 4. Let us assume that L_3 is accepted by some finite-turn deterministic queue automaton. It is shown in [7] that any k-turn deterministic queue automaton can be converted into an equivalent $2k$-flip deterministic flip-pushdown automaton which is basically a deterministic pushdown automaton with the additional ability to reverse the current contents of the pushdown store. Thus, L_3 can be accepted by such an automaton with a finite number of flips. On the other hand, it is shown in [5] that L_3 cannot even be accepted by any nondeterministic flip-pushdown automaton with a finite number of flips. Hence, L_3 is not accepted by any finite-turn deterministic queue automaton.

Let us now consider the union $L = L' \cup L''$ with $L' = \{\, a^n b^m c^n \mid m,n \geq 1 \,\}$ and $L'' = \{\, a^n b^m c^{n+m} \mid m,n \geq 1 \,\}$. It is not difficult to construct a queue automaton with one turn which accepts L. In the following, we show that L is not accepted by any DSA. Assuming the opposite, there is a DSA M with state set S that accepts L. Let M be in infinite action normal form. Clearly, language L is not regular and is not accepted by any finite automaton. Since M is deterministic and has only a finite number of states, we conclude that M has to enter a loop while computing the a-sequence of an accepted input which is long enough. Moreover, after a constant number of computation steps M has to write something on the tape. Otherwise, M is in the same configuration for two different sequences a^i and a^j with $i \neq j$. Thus, for suitably chosen $m \geq 1$, M would also accept $a^i b^m c^j$ with $i + m \neq j$ which is a contradiction. At some time, M has to perform an `in`-operation, since otherwise M could be simulated by a finite automaton. If M performs an `in`-operation before entering the loop or being in the loop, then the length of the added word is bounded by some constant. Since M is in infinite action normal form, this case cannot occur.

So, the first possibility to leave the loop is after reading a constant number of b's. In this case, let w_a be the word added to the set. For an input long enough, M will again enter a loop while reading b's which may be left not before reading a constant number of c's. As before, there is no operation on the set between adding w_a and leaving the loop. The only useful operation at this moment is another `in`-operation adding some word w_b to the set. Any `out`- or `test`-operation would empty the tape without storing information about the number of b's. It is not hard to construct witnesses for the fact that M cannot accept L in this case. As before, we conclude that M now enters a loop while reading the c's, which cannot be left before reaching the endmarker. Moreover, there is once more no operation on the set between adding w_b and leaving the loop. Let w_c be the content of the working tape at this moment. Altogether we have that M performs at most three operations on the set, where only the last one may be different from an `in`-operation. If the last operation is not a test, then M can be simulated by a deterministic finite automaton, a contradiction. So, we may assume that there are either two `in`-operations adding w_a and w_b to the set, or just one `in`-operation adding some word w_{ab} to the set after leaving the first or second loop, followed by a `test`-operation with w_c on the tape. Since there are arbitrarily many c-sequences in words not belonging to L for which the test will be negative, the test has to be positive for words in L. Consider an input $a^\ell b^m c^{\ell+m} \in L$ for ℓ, m large enough. If the test reveals $w_a = w_c$, then input $a^\ell b^{2m} c^{\ell+m} \notin L$ is accepted as well. Similarly, if the test reveals $w_b = w_c$, then input $a^{2(\ell+m)} b^m c^{\ell+m} \notin L$ is accepted. So, the only possibility left is that there is just one `in`-operation. Now we consider an input $a^\ell b^m c^\ell \in L$ for ℓ, m large enough. If we are not concerned with one of the previous cases, the length of the word w_{ab} depends linearly on ℓ and m. Therefore, if the test reveals $w_{ab} = w_c$, then it is negative on input $a^\ell b^{m'} c^\ell \in L$, for large m', and the input is rejected. □

4 Closure Properties

In this section, we investigate the closure properties of DSA with respect to the Boolean operations. The proof of closure under complementation may be found in the Appendix.

Lemma 12. *The family of languages accepted by* DSA *is closed under complementation.*

Lemma 13. *The family of languages accepted by* DSA *is not closed under union and intersection.*

Proof. In the proof of Theorem 11, we have shown that the language $L = L' \cup L''$ with $L' = \{\, a^n b^m c^n \mid m, n \geq 1 \,\}$ and $L'' = \{\, a^n b^m c^{n+m} \mid m, n \geq 1 \,\}$ is not accepted by any DSA. Similar to the construction in Example 3, we can construct DSA accepting L' as well as L''. Thus, we obtain that $\mathscr{L}(\text{DSA})$ is not closed under union. Moreover, since $\mathscr{L}(\text{DSA})$ is closed under complementation by Lemma 12, it cannot be closed under intersection. □

Lemma 14. *The family of languages accepted by* DSA *is closed under intersection with regular languages and under union with regular languages.*

Proof. A DSA can simulate a given deterministic finite automaton in parallel to its computation by using the standard cross product construction. Thus, family $\mathscr{L}(\text{DSA})$ is closed under intersection with regular languages. Since $\mathscr{L}(\text{DSA})$ is closed under complementation by Lemma 12, it is closed under union with regular languages as well. □

5 Decidability of Emptiness

Here we turn to show that emptiness is decidable for deterministic set automata. This is of interest both from a theoretical and practical point of view. For example, it is known that emptiness is decidable for pushdown automata and stack automata [4], but is undecidable for deterministic quasi-real-time queue automata [1] and deterministic one-way multi-head finite automata.

Theorem 15. *It is decidable whether or not a given deterministic set automaton accepts the empty language.*

Proof. Given a deterministic set automaton $M = \langle S, \Sigma, \Gamma, \lhd, \delta, s_0, F \rangle$, the basic idea is to construct a meta automaton M' and to explore all possible paths up to a certain length in its state graph in order to find a path from the initial state to some accepting state. If such a path does not exist, the accepted language is empty. The proof is structured in multiple steps.

In a first step, the DSA M is transformed into the meta automaton M' whose states are the initial state and the `in`- `out`-, and `test`-states of M. The edges of M' are labeled with regular languages. The language of an edge connecting state s_i with s_j represents all strings that can be written on the tape when a computation of M passes directly from s_i to s_j.

The next step is to elaborate some properties of accepting paths of M and M'. In particular, it is shown that there exists an accepting path of bounded length if there is an accepting path at all.

Finally, we show which of the paths of M' can be expanded to paths of M, that is, how a path can be evaluated to represent an accepting computation of M. If there is no such path, then the language accepted by M is empty and non-empty otherwise.

Construction of the meta automaton. We assume that M is in infinite action normal form so that $S = S_{\texttt{in}} \cup S_{\texttt{out}} \cup S_{\texttt{test}} \cup S_{\texttt{write}}$. For convenience we define $\bar{S} = S_{\texttt{in}} \cup S_{\texttt{out}} \cup S_{\texttt{test}}$.

The state set S' of the meta automaton M' is set to be $\{s_0\} \cup \bar{S}$. If there is a path *in* M from state s_i *of* S' to s_j *of* S' without passing through any other state of S' in between, then an edge from s_i to s_j is included in M'. The edges are labeled with the languages L_{s_i,s_j} as constructed in the proof of Lemma 9. That is, the languages represent all strings that can be written on the tape when a computation of M passes directly from s_i to s_j. Since M is in infinite action normal form, every edge is labeled by an infinite language. It is worth mentioning that we do not care about the actual input here, but for the argumentation it is understood that there always is a suitable input. Moreover, we recall that the set of `test`-states is the disjoint union of `test+` and `test-` states indicating whether the preceding test was successful or not.

The set of accepting states of M' is defined as follows. First, all states from S' that are accepting in M are accepting in M' as well. Second, for every edge connecting state s_i with s_j in M', state s_j is defined to be accepting in M' if some accepting state is passed through in a direct path from s_i to s_j in M. Similarly, a state s_i of M' is defined to be accepting, if there is some path in M starting in s_i that never reaches any other state of M', passes through an accepting state of M, and reaches a state of M in which the computation of M ends.

Properties of accepting paths I. By construction of M', every accepting path in M corresponds to an accepting path in M' where the states from $S_{\texttt{write}}$ are omitted. Conversely, every accepting path in M' can be expanded to an accepting path in M again. Since the main idea is to find an accepting path in M', we are interested in rules for adding edges of M' to a path that can be expanded.

For the first rule, we recall that at every time step in a computation of M there are only finitely many strings in its set $\mathbb{S}$. On the other hand, every edge connecting two states in M' is labeled by an infinite language. So, it is always possible to find some word in the language that is not in the set of M. Therefore, it is always possible to add an edge that connects to a `test-`-state to an expandable path in M'. The set $\mathbb{S}$ is not changed by using an edge to a `test-`-state.

The next rule concerns edges to `out`-states. Let p be an accepting path in M. Then there is an accepting path in M so that any `out`-operation does not remove anything from the set $\mathbb{S}$. This path is constructed as follows. Assume there is a sub-path of p connecting directly state $s_i \in S'$ with the `out`-state s_j. If the string written on the tape while moving along the sub-path belongs to the current set $\mathbb{S}$, the sub-path is replaced by another one which writes a string on the tape that does not belong to $\mathbb{S}$. Since M is in infinite action normal form, such a sub-path connecting s_i and s_j directly always exists. This replacement does not change the rest of the computation along p as long as `test+`-, `out`-, and `in`-states are concerned. It may, however, change the computation for `test-`-states since now there is one string more in $\mathbb{S}$. Fortunately, by the first rule all further sub-paths of p that connect to a `test-`-state can be replaced by another one that writes a string not belonging to $\mathbb{S}$ on the tape. Repeating this process until all `out`-operations have the property claimed concludes the construction. So, for finding an accepting path in M', from now on we safely may assume that the set $\mathbb{S}$ is not changed by using an edge to an `out`-state.

We conclude that we have only to consider states from $S_{\texttt{in}}$ and $S_{\texttt{test+}}$ for the further reasoning.

Properties of accepting paths II. We turn to `in`- and `test+`-states. Assume there is an edge in M' connecting state s_i with the `test+`-state s_j. If this edge belongs to an accepting path, a string $w \in L_{s_i,s_j}$ is tested positively and, thus, belongs to the current set $\mathbb{S}$. Since we may assume that `out`-and `test-`-states do not change the content of $\mathbb{S}$, the edge can appear arbitrarily often in the accepting path, always with test string w.

In general, an edge connecting state s_i with the `test+`-state s_j can be added to an accepting path provided that some word w from the language L_{s_i,s_j} belongs to the current set $\mathbb{S}$. Again, since we may assume that `out`-and `test-`-states do not change the content of $\mathbb{S}$, it is sufficient that the word w has been inserted into $\mathbb{S}$ before. To this end, the intersection of L_{s_i,s_j} and some language $L_{s'_i,s'_j}$ labeling an edge connecting state s'_i with an `in`-state s'_j has to be non-empty. Furthermore, this edge has to belong to the preceding part of the accepting path. However, this condition is too weak. Consider the following scenario. If $L_{s'_i,s'_j} = a^* \cup b^*$ and $L_{s_i,s_j} = a^*$, then their intersection is non-empty. Now let there be another edge connecting state s''_i with the `test+`-state s''_j and $L_{s''_i,s''_j} = b^*$. So, for both `test+`-states the condition is met and everything seems to be fine. But since either a word of the form a^* or of the form b^* is inserted, only one test can be positive. Nevertheless, due to the observation in the previous paragraph, it is sufficient to know that for any occurrence of a new edge connecting to a `test+`-state, there is one specific edge in the preceding part of the accepting path, that inserts the positively tested word. So, let t denote the number of edges in M' that connect to a `test+`-state. Then it is sufficient that any edge connecting to an `in`-state occurs at most t times in an accepting path to satisfy any following test.

Properties of accepting paths III. Let e denote the number of edges in M' that connect to an `in`-state. If there is an accepting path in M' at all, there is an accepting path whose length is at most $(t+1) \cdot e \cdot |S| + |S|$.

To give evidence for the claim assume there is an accepting path in M' whose length exceeds $(t+1) \cdot e \cdot |S| + |S|$. We analyze the path from left to right and keep track in some vector $c \in (\{0, 1, \ldots, t\} \cup \{t^>\})^e$ how many times each edge connecting to an `in`-state occurs. We start with zero vector c_0 and increase a component by one if the corresponding edge occurs. If the component is t or $t^>$ it is set to $t^>$.

Since the path is longer than $|S|$ it contains a loop. If there is any loop that does not change the vector, we know from above that we safely can remove the loop from the path since the operations in the loop do not affect the remaining operations. So, we have found a shorter accepting path. If it is still longer than $(t+1) \cdot e \cdot |S| + |S|$ we repeat the reasoning.

On the other hand, assume that all loops increase at least one component of the vector. Then, after at most $(t+1) \cdot e$ many loops, each of length at most $|S|$, the vector cannot change anymore. So, the total length of the path is at most $(t+1) \cdot e \cdot |S| + |S|$ and the claim follows.

Searching for an accepting path. Basically, we consider the computation tree built from the state graph of the meta automaton M' and perform a depth-first search to explore an accepting path. We start at the root which is associated with the initial state of M'. Whenever the current state of a path is s and $\ell \geq 1$ states can be reached from s, that is, there are edges between s and states $s_1, s_2, \ldots, s_\ell$, the corresponding node of the tree has the successor nodes $s_1, s_2, \ldots, s_\ell$. Each node of the tree is labeled by a set of elements of the form $\mathbb{N} \times S'^2 \times \Gamma^*$, where the last component is a regular language. The root is labeled by the empty set. Such an element (x, s_i, s_j, R) of a label represents the information that x strings from the language R are in the set $\mathbb{S}$, and that these strings have been inserted while exploring the edge from s_i to s_j.

When the search extends the current path by visiting a node associated with state s_j reached from a node associated with state s_i, several cases are distinguished. If s_j is an `out`$-$ or `test-`-state the label of s_i is copied to s_j.

If s_j is an `in`-state the label of s_i is copied to s_j. If this label contains already an element of the form (x, s_i, s_j, R) the first component is increased by one. Otherwise, the tuple $(1, s_i, s_j, L_{s_i,s_j})$ is added to the label.

If s_j is a `test+`-state reached from s_i for the first time, we have to verify that the test can be positive. To this end, the intersection $L_{s_i,s_j} \cap L_m$ is built for each of the $m \geq 0$ tuples (x_m, s_m, s'_m, L_m) in the label of s_i. If all these intersections are empty, the exploration of the current path is stopped. Otherwise, first the node associated with s_j currently reached in the tree is removed. Second, let $I_1, I_2, \ldots, I_n$, $n \geq 1$, be the non-empty intersections. Then n new nodes associated with s_j are inserted as successors of s_i. The label of s_i is copied to each of the new nodes. In addition, the label of node k is updated by adding the tuple $(1, s_i, s_j, I_k)$, $1 \leq k \leq n$, and decreasing the first component of (x_k, s_k, s'_k, L_k) by one. If $x_k = 0$, the tuple (x_k, s_k, s'_k, L_k) is removed from the label.

If s_j is a `test+`-state reached again from s_i, the same word as before can be used for the successful test. So, only the label of s_i is copied to s_j.

In this way, the situation discussed at the end of *Properties of accepting paths II* is covered.

It remains to be explained how the exploration of a branch of the depth-first search is stopped in the case of a non-accepting computation path. To this end, we have the following criteria. The exploration of a path is stopped if there are no outgoing edges from the current state, all intersections constructed are empty after entering a `test+`-state, or the length of the path exceeds $(t+1) \cdot e \cdot |S| + |S|$. In particular, the second criterion applies if there never can be words in the set for which the test is positive. Whenever an accepting state is entered, the exploration of the graph is stopped and we know that $L(M)$ is not empty.

If $L(M)$ is not empty, then there is an accepting path of length at most $(t+1) \cdot e \cdot |S| + |S|$ which will be found by the procedure above.

If the depth-first search stops since the exploration of all paths is stopped due to the above criteria, then we know that there are no accepting paths up to length $(t+1) \cdot e \cdot |S| + |S|$ which implies that $L(M)$ is empty. If $L(M)$ is empty, then the exploration of any path will be stopped by the above criteria. Altogether, we have obtained a procedure which decides whether or not $L(M)$ is empty. $\square$

References

1. Cherubini, A., Citrini, C., Crespi-Reghizzi, S., Mandrioli, D.: QRT FIFO automata, breadth-first grammars and their relations. Theoret. Comput. Sci. 85, 171–203 (1991)
2. Chomsky, N.: On certain formal properties of grammars. Inform. Control 2, 137–167 (1959)
3. Daley, M., Eramian, M.G., McQuillan, I.: The bag automaton: A model of nondeterministic storage. J. Autom., Lang. Comb. 13, 185–206 (2008)

4. Ginsburg, S., Greibach, S.A., Harrison, M.A.: One-way stack automata. J. ACM 14, 389–418 (1967)
5. Holzer, M., Kutrib, M.: Flip-pushdown automata: $k+1$ pushdown reversals are better than k. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) International Colloquium on Automata, Languages and Programming (ICALP 2003). LNCS, vol. 2719, pp. 490–501. Springer (2003)
6. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
7. Kutrib, M., Malcher, A., Mereghetti, C., Palano, B., Wendlandt, M.: Input-driven queue automata: Finite turns, decidability, and closure properties. In: Konstantinidis, S. (ed.) Implementation and Application of Automata (CIAA 2013). LNCS, vol. 7982, pp. 232–243. Springer (2013)
8. Lange, K.-J., Reinhardt, K.: Automaten mit der Datenstruktur Menge. In: Kutrib, M., Worsch, T. (eds.) 5. Theorietag Automaten und Formale Sprachen. pp. 159–167. Universität Giessen, Giessen, Germany (1995)
9. Ogden, W.F.: Intercalation theorems for stack languages. In: Proceedings of the First Annual ACM Symposium on Theory of Computing (STOC 1969). pp. 31–42. ACM Press, New York (1969)
10. Sénizergues, G.: $L(A) = L(B)$? decidability results from complete formal systems. Theoret. Comput. Sci. 251, 1–166 (2001)

Appendix

Proof (of Theorem 6). In contrast to the assertion assume M is a quasi-real-time queue automaton accepting L_2 with a set of states S. We consider a computation on input $w = a^j b^{j'} v$ with $v \in \Sigma^*$. After M has read the sequence a^j there has to be a word z enqueued so that its length $|z|$ depends on the length of the given input word a^j. Assuming the contrary, there are two different accepted words $w' = a^i b \$_1 c^i$ and $w'' = a^{i'} b \$_1 c^{i'}$ with $i \neq i'$ such that after reading a^i and $a^{i'}$, the DSA M is in the same configuration. But this is a contradiction to the assumption, because in this case M accepts as well the word $a^i b \$_1 c^{i'}$ with $i \neq i'$ that is not in the language. It can be argued in the same way that a word z' has to be enqueued for the b sequence where $|z'|$ depends on the number of b's in the given word w. So after reading $a^j b^{j'}$ of the input, there has to be zz' in the queue where z is the front and z' is the tail of the queue. If the next symbol is a $\$_1$ then M has to compare the number of c's with the number of b's in the word. We may conclude that while reading the remaining c's the queue has to be dequeued since otherwise L_2 could be accepted by some finite automaton and, hence, would be regular. Let $a^i b^j \$_1 c^j$ with $i, j \geq |S|$ be an input word such that the length of the enqueued word z is larger than $2j \cdot |S|$. Clearly, M can dequeue at most $|S|$ symbols from the queue for each input letter. So, M has to be in an accepting state after reading $a^i b^j \$_1 c^j$ and in the front of the queue there is still a word $\tilde{z}$ such that $z = \tilde{z}' \tilde{z}$ and $|\tilde{z}| > j \cdot |S|$. Thus, there is also another word $a^i b^j \$_1 c^{j+j'}$ with $i, j \geq |S|$ and $j' \geq 1$ such that M is in the same accepting state. Since $j \neq j + j'$, this is a contradiction and establishes our claim. $\square$

Proof (of Lemma 9). We sketch the conversion and start by assuming that M is in action normal form. First, we construct all regular languages L_{s_i,s_j} and test their finiteness. Next, we determine the length k of a longest word with respect to all finite languages. To avoid operations corresponding to finite sets L_{s_i,s_j}, we have to ensure that no `in`-, `out`-, or `test`-operations are performed for words on the tape that have a length of at most k. To this end, we construct an equivalent DSA M' that is able to store all words up to length k in its state set. In this way, the set $\mathbb{S}$ is simulated by states for such words. Additionally, by using a buffer in its states, M' writes nothing on the tape until the word to be written on the tape is larger than k. In this case M' writes all symbols in the buffer in one step on the tape, empties its buffer, and continues the computation. By construction, M' is equivalent to M, since all set operations corresponding to words of length at most k are performed in the state set, and all set operations corresponding to words of length larger than k are still performed on the tape and the set. Moreover, M' still works deterministically and is in infinite action normal form. $\square$

Proof (of Theorem 10). We show that $L = \{ wcw^R \mid w \in \{a,b\}^* \}$ is not accepted by any deterministic set automaton. In the following, we are often arguing with two parts of a word in L. So, we call the sequence up to the middle marker c the first part of the word, and the remaining sequence the second part of the word.

The proof is by contradiction. Assuming that L is accepted by some DSA M, we will show as a first step that L is accepted by M in such a way that all possible set operations performed on the first part of the input are a finite number of in-operations, and on the second part are a finite number test-operations. In a second step, based on M an equivalent one-way multi-head finite automaton accepting L is constructed. This leads to a contradiction.

Let L be accepted by a DSA $M = \langle S, \Sigma, \Gamma, \lhd, \delta, s_0, F\rangle$ which is in infinite action normal form. Thus, $S = S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}} \cup S_{\mathtt{write}}$ and we know that all sets L_{s_i,s_j} with $s_i \in \{s_0\} \cup S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}}$ and $s_j \in S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}}$ are infinite.

Next, we will show that there are no test-operations in the first part of an accepted input. Let us first discuss the case when some test in the first part is negative. We consider the subcomputation $(s_i, uvcw^R, \lambda, \mathbb{S}) \vdash^* (s_j, vcw^R, \lambda, \mathbb{S}')$ on input $u'uvcw^R \in L$ with $w = u'uv$, $s_j \in S_{\mathtt{test}}$, $s_i \in \{s_0\} \cup S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}}$, and assume that the test has a negative result. Since L_{s_i,s_j} is infinite, there are infinitely many input sequences whose transductions belong to L_{s_i,s_j}. Therefore, there exists some $u'' \in \{a,b\}^+$ with $u \neq u''$, so that the DSA also accepts the input $u'u''vcw^R$, which is a contradiction. We conclude that every test in the first part has to be successful. Now assume as before that the DSA is in state s_i after the processing of the input prefix u'. For any input $\tilde{u}v$ there is a word $u'\tilde{u}vc(u'\tilde{u}v)^R \in L$. Since there are only finitely many words in $\mathbb{S}$, the test has to be successful, and L_{s_i,s_j} is infinite, we can conclude that there are two different words $\tilde{u}$ and $\hat{u}$ whose transduction on the tape is the same. This implies that M is in the same configuration after reading $u'\tilde{u}$ and after reading $u'\hat{u}$. Thus, both words $u'\tilde{u}vc(u'\tilde{u}v)^R$ and $u'\hat{u}vc(u'\tilde{u}v)^R$ are accepted, a contradiction. Hence, we may assume that there is never a test-operation in the first part of accepted inputs.

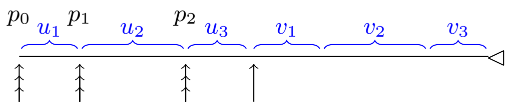
In a similar way it can be proved that for accepting computations there are never out-operations in the first or in the second part of the input, and that there are never in-operations in the second part of the input.

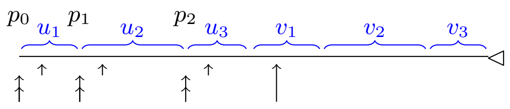
Next, we turn to show that M can perform only a constant number of in-operations in the first part of accepting computations. Assume contrarily that M performs $k > |S|^2$ input operations in the first part of some input. Then there are two states $s_i \in S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}}$ and $s_j \in S_{\mathtt{in}}$ such that M runs from s_i to s_j twice. Consider such a computation on an input $w = zcz^R$ with $z = vuv'u'v''$, where M is in state s_i when it reads the first symbol of u, and is in state s_j when it reads the first symbol of v', and is again in state s_i when it reads the first symbol of u', and is again in state s_j when it reads the first symbol of v''. Then we can conclude out of the fact that M does not perform any test- or out-operations while computing the subword v', that M is in the same configuration after reading the subwords $vuv'u'v''$ and $vu'v'u'v''$. Choosing $u \neq u'$, which is always possible since L_{s_i,s_j} is infinite, we obtain a contradiction, because both words $zcz^R \in L$ with $z = vuv'u'v'$ and $vu'v'u'v''cz^R \notin L$ would be accepted.

Next, we turn to prove that there are only a constant number of tests in the second part of accepted inputs. First, we show that M never performs a negative test-operation in the second part of an accepted input. Assuming the contrary, there is a word $wcuvu' \in L$ such that M is in state $s_i \in S_{\mathtt{in}} \cup S_{\mathtt{out}} \cup S_{\mathtt{test}}$ with empty tape after reading wcu. Now, M reads v, writes some z on the tape, tests z, and enters some state $s_j \in S_{\texttt{test-}}$ as result of a negative test. Thus, M is in configuration $(s_j, u', \lambda, \mathbb{S})$. Since L_{s_i,s_j} is infinite and the content of the set $\mathbb{S}$ is finite, there is a another word $wcuv'u'$ with $v \neq v'$ such that M is also in configuration $(s_j, u', \lambda, \mathbb{S})$ after reading v'. This is a contradiction, since then $wcuv'u' \notin L$ would be accepted. So, we can conclude that all `test`-operations performed in the second part of accepted inputs are positive. Assume now that the number of tests is greater than $|S|$. Then one `test`-state is entered at least twice. We may assume that in the computation on an accepted word $w = zcz^R$ with $z^R = vuv'$ M reaches some state $s_j \in S_{\texttt{test}}$ when it reads the first symbol of u and again when reading the first symbol of v'. Therefore all words $zcvu^iv'$ with $i \geq 1$ are accepted as well, since we know that there are no `out`-operations in the second part. Choosing $i = 2$ leads to a contradiction.

Now we know that M never performs `test`- or `out`-operations in the first part of accepted inputs and never performs `in`- or `out`-operations in the second part of accepted inputs. Furthermore, at most $|S|^2$ `in`-operations in the first part as well as at most $|S|$ `test`-operations in the second part are performed. In the following, we describe how M can be simulated by a one-way multi-head finite automaton.

Let uv be an input word and let $u_1, u_2, \ldots, u_n$ be the subwords of u whose transductions are added to the set by `in`-operations, and $v_1, v_2, \ldots, v_m$ be the subwords of v whose transductions are tested. We construct a one-way multi-head finite automaton M' that leaves $|S|$ many heads at position $p_0 = 1$, that is, at the beginning of the input word, reads the input using some head h, starts to simulate M omitting the simulation of the tape, and leaves $|S|$ many heads in the first part of the input at every moment when M empties its tape and adds some u_i to its set, except for the last `in`-operation. In the following, these positions are denoted by $p_1, p_2, \ldots, p_{n-1}$. Moreover, for $0 \leq i \leq |S| - 1$, $h_{i,j}$ denotes the jth head (out of $|S|$ heads) that has been left at position p_i. The states $s_1, s_2, \ldots, s_{n-1}$ the DSA M is in at these moments are stored in the state set of M'. Let us first assume that exactly $n = |S|^2$ `in`-operations are performed. By counting the number of `in`-operations in the state set, we know when the last `in`-operation has been performed. At that moment, M starts to write v_1 on the tape which is eventually tested with the contents of the set. To simulate this behavior by M', we use the heads $h_{0,1}, h_{1,1}, \ldots h_{|S|^2-1,1}$ to start in states $s_0, s_1, s_2, \ldots, s_{n-1}$ at positions $p_0, p_1, p_2, \ldots, p_{n-1}$ to compare the transductions of the words $u_1, u_2, \ldots, u_n$ with the transduction of v_1 read by head h. If an agreement is found when some state $s_j \in S_{\texttt{test}}$ is entered, that is, M has added some word to the set which is now positively tested, then the simulation is continued by comparing the transductions of the words $u_1, u_2, \ldots, u_n$ with the transduction of v_2 using the heads $h_{0,2}, h_{1,2}, \ldots h_{|S|^2-1,2}$. This behavior is continued until all m tests have been simulated successfully. Finally, it is checked with head h whether M enters an accepting state. In this case M' accepts the input and rejects otherwise. The following two pictures show the situation of dropping all heads in the first part, and the simulation for the test of the transduction of v_1. The rightmost head is head h.





Let us now discuss the case when $n < |S|^2$ `in`-operations have been performed. In this case, it not clear which `in`-operation is the last one that starts the comparing phase. To manage this case, we drop another $|S|^2$ heads at every position $p_0, p_1, \ldots, p_{n-1}$ and interpret every `in`-operation as the last operation which starts the comparing phase. If the next operation is an `in`-operation, we start a new comparing phase with a new set of heads. If the next operation is a `test`-operation, we continue the comparing phase with a new set of heads. Altogether, we need at most $|S|^3 + |S|^4 + 1$ heads.

In summary, the simulation shows that L is accepted by a one-way multi-head finite automaton. This is a contradiction, since it is known that L is not accepted by any one-way multi-head finite automaton. □

Proof (of Lemma 12). The closure under complementation for deterministic finite automata can be easily proved by interchanging accepting and rejecting states. We cannot translate this idea directly to DSA, because mainly three problems may occur. First, the given DSA may not read its input completely by either entering a configuration in which no next move is defined (1) or by entering an infinite λ-loop (2). Second, the given DSA may perform λ-steps leading from an accepting state to a rejecting state and back (3).

Now, let M be a DSA for which we want to construct a DSA accepting its complement. To overcome problem (1), we introduce a new non-accepting state s_{rej} to which all undefined transitions of M are directed. Additionally, we define further moves from s_{rej} which shift the input head to the end of the input. For problem (3), we note that M can accept at the earliest after reading the endmarker. Thereafter additional λ-steps may be possible. We now want to achieve that in this case M enters an accepting state as soon as possible which cannot be left. To this end, a new accepting state s_{acc} is added for with the transition function is undefined. Moreover, we double the state set of M and store in every state the information whether or not the endmarker has been passed. If we now have a transition entering an accepting state with the knowledge that the endmarker has been passed, we redirect such a transition to enter state s_{acc}. By these modifications we have obtained an equivalent DSA M' in which problems (1) and (3) do no longer occur. However, M' may enter infinite λ-loops. Next, we transform M' into infinite action normal form and note that by the construction problems of type (1) and (3) are not occurring. Let us distinguish two cases: first, we assume that on infinite λ-loops only states from $S_{\texttt{write}}$ can be visited. By an inspection of the transition function we can check in advance which states from $S_{\texttt{write}}$ will end in an infinite λ-loop. Then, any transition ending in such a state will be redirected to s_{rej}. Second, we assume that we have an infinite λ-loop in which some state $s_1 \in S_{\texttt{in}} \cup S_{\texttt{out}} \cup S_{\texttt{test}}$ is entered. Let $s_2 \in S_{\texttt{in}} \cup S_{\texttt{out}} \cup S_{\texttt{test}}$ be the next, not necessarily different, non-writing state along the λ-loop. Then, we consider the language L_{s_1,s_2} which has to be infinite due to the infinite action normal form. On the other hand, M' is deterministic, no input is read while moving from s_1 to s_2, and the tape is empty when starting in s_1 and when having reached s_2. Hence, it is only possible to write one word on the tape while moving from s_1 to s_2. This implies that L_{s_1,s_2} is finite. Thus, this case cannot occur.

Now, we have obtained that any computation ends when the whole input and the endmarker is read either in an accepting or non-accepting state which cannot be left once entered. Thus, the standard technique for constructing an automaton that accepts the complement can be used: all accepting states become non-accepting states and all non-accepting states become accepting states. $\square$