IFIG Research Report

Institut für Informatik

A Note on the Computational Complexity of Some Problems for Self-Verifying Finite Automata

Markus Holzer Sebastian Jakobi

IFIG Research Report 1702
April 2017

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

IFIG Research Report
IFIG Research Report 1702, April 2017

A Note on the Computational Complexity of Some Problems for Self-Verifying Finite Automata

Markus Holzer[1] and Sebastian Jakobi[2]

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

Abstract. We consider the computational complexity of some problems for self-verifying finite automata (svFAs). In particular, we answer a question stated in the open problem session of the Workshop on Descriptional Complexity of Formal Systems 2015 held in Waterloo, Ontario, Canada, on the complexity of the promise version of the general membership problem for svFAs, showing that this problem is NL-complete.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata*; F.1.3 [**Computation by Abstract Devices**]: Complexity Measures And Classes—*Reducibility and completeness*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Decision problems*;

Additional Key Words and Phrases: self verifying automata, computational complexity, membership problem, emptiness problem, universality problem

[1] E-mail: holzer@informatik.uni-giessen.de
[2] E-mail: sebastian.jakobi@informatik.uni-giessen.de

1 Introduction

We consider the computational complexity of some problems for self-verifying finite automata (svFA), such as the problem to determine whether a given finite state machine is a self-verifying automaton, which is PSPACE-complete. Moreover, we show that the promise versions of the general membership, non-emptiness, and universality problem for self-verifying finite automata is NL-complete. Here the promise is that the given input is in fact a self-verifying device. In particular, the NL-completeness on the universality problem for self-verifying automata nicely contrasts the PSPACE-completeness of the same problem for ordinary nondeterministic finite automata. We also we study counting problems for self-verifying finite automata. In the course of action we prove as a byproduct also some results on self-verifying pushdown automata.

The note is organized as follows: In the next section we introduce the necessary notations on finite automata and computational complexity theory. Then we present our results: we start by generalizing svFAs to self-verifying Turing machines, proving in passing that NL can be alternatively characterized as those languages that are accepted by self-verifying logspace bounded Turing machines. This alternative characterization allows us to define a variant of a reachability problem which can easily be converted into the promise version on the general membership problem for svFAs. As a byproduct of this characterization we also prove some results on self-verifying pushdown automata. Finally we also give precise computational bounds on the promise versions of the non-emptiness and the universality problem for svFAs and consider the computational complexity of counting problems on self-verifying finite automata.

2 Preliminaries

We recall some definitions on finite automata as contained in [6]. A *nondeterministic finite automaton* (NFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of *states*, Σ is the finite set of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, and $\delta\colon Q \times \Sigma \to 2^Q$ is the *transition function*. The *language accepted* by the finite automaton A is defined as $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \,\}$, where the transition function is recursively extended to $\delta\colon Q \times \Sigma^* \to 2^Q$. A finite automaton is *deterministic* (DFA) if and only if $|\delta(q, a)| = 1$, for all states $q \in Q$ and letters $a \in \Sigma$. Then we simply write $\delta(q, a) = p$ instead of $\delta(q, a) = \{p\}$, assuming that the transition function is a total mapping $\delta\colon Q \times \Sigma \to Q$.

Self-verifying automata were introduced in [4]. A *self-verifying finite automaton* (svFA) is a six-tuple $A = (Q, \Sigma, \delta, q_0, F_+, F_-)$, where Q, Σ, δ, and q_0 are the same as for NFAs, $F_+ \subseteq Q$ is the set of *accepting* or *"yes"* states, $F_- \subseteq Q$ is the set of *rejecting* or *"no"* states, and $F_+ \cap F_- = \emptyset$. The states in $F_+ \cup F_-$ are called *final*, and the remaining states in Q, i.e., $Q \setminus (F_+ \cup F_-)$, are called *neutral* or *"don't know"* states. Moreover, it is required that for each input string w in Σ^*, there exists at least one computation ending in a final state, i.e. in an accepting or in a rejecting state, that is, $\delta(q_0, w) \cap (F_+ \cup F_-) \neq \emptyset$, and there are no words w such that both $\delta(q_0, w) \cap F_+$ and $\delta(q_0, w) \cap F_-$ are nonempty—this will be called the *self-verifying property*. The *language accepted* by the self-verifying finite automaton A is defined as $L_+(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \cap F_+ \neq \emptyset \,\}$, while the *language rejected* by A is $L_-(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \cap F_- \neq \emptyset \,\}$. For both definitions the transition function is recursively extended to $\delta\colon Q \times \Sigma^* \to 2^Q$ as usual. Notice that $L_+(A)$ and $L_-(A)$ form a partition of Σ^* because of the self-verifying property. We may also write $L(A)$ instead of $L_+(A)$ for the language accepted by A.

It is easy to see that the family of languages accepted by DFAs, svFAs, and NFAs are all equal, but their descriptional complexity may vary, which may induce differences in the computational complexity of standard problems thereof. In this note we are interested on the

computational complexity of some problems for svFAs. To this end we assume the reader to be familiar with the basics in computational complexity theory. Consider the inclusion chain $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$. Here L (NL, respectively) refers to the set of problems accepted by deterministic (nondeterministic, respectively) logspace bounded Turing machines, P (NP, respectively) is the set of problems accepted by deterministic (nondeterministic, respectively) polynomial time bounded Turing machines, and PSPACE is the set of problems accepted by deterministic or nondeterministic polynomial space bounded Turing machines. Further, for a complexity class C, the set coC is the set of complements of languages from C. Hardness and completeness is always meant w.r.t. deterministic logspace bounded many-one reducibility.

Finally, we recall some known facts on the complexity of standard problems for DFAs and NFAs. Most problems for DFAs are easy. In particular, the general membership is L-complete, the non-emptiness problem, and the universality problem are NL-complete [3, 9]. The former two problems are NL-complete for NFAs [9], while the latter one becomes PSPACE-complete [10], and thus highly intractable.

3 Results

Our first result for svFAs is to determine, whether a given finite state device with accepting and rejecting states fulfills the self-verifying property. This problem is shown to be already PSPACE-complete.

Theorem 1. *Given a finite state device A with accepting and rejecting states, decide whether A is a self-verifying automaton, i.e., whether it fulfills the self-verifying property, is* PSPACE*-complete.*

Proof. The containment in PSPACE is seen as follows: From A we construct two NFAs B and B' such that $L(B) = L_+(A)$ and $L(B') = L_-(A)$ by simply defining the accepting states properly. Then we have to check (i) $L(B) \cup L(B') = \Sigma^*$ and (ii) $L(B) \cap L(B') = \emptyset$. By the standard cross product construction one builds two NFAs C and C' accepting $L(B) \cup L(B')$ and $L(B) \cap L(B')$, respectively, of polynomial size. Then checking C for universality and C' for emptiness results in a test for the self-verifying property for A. Since the former one is solvable in PSPACE and the latter one in NL, the containment in PSPACE follows.

It remains to show PSPACE-hardness. We present a reduction from the universality problem for NFAs. Given a NFA A we interpret this automaton as a finite state device with accepting states and neutral states, but with an empty set of rejecting states. Assume that Σ is the input alphabet of A. Then the following holds: if $L(A) = \Sigma^*$, then the newly interpreted device is obviously an svFA. On the other hand, if $L(A) \neq \Sigma^*$, then there is a word $w \notin L(A)$, and thus, the newly interpreted device cannot fulfill the self-verifying property, since it is violated by w: there is no accepting computation by assumption and there is no rejecting computation, since the device under consideration does not have rejecting states at all. This shows PSPACE-hardness. □

This result shows that problems on svFAs are already PSPACE-hard, since the check whether the input obeys the self-verifying property is that complicated. Thus, in the forthcoming we only consider promise problems, i.e., problems where the input already is in fact an svFA. Next we study the computational complexity of the general membership problem for self-verifying finite automata, which is the problem of deciding for a given self-verifying finite automaton A and an input word w, whether $w \in L(A)$ holds.

Theorem 2. *The promise version of the general membership problem for self-verifying finite automata is* NL*-complete.*

In order to prove this result we proceed as follows: (i) we generalize svFAs to self-verifying Turing machines, and moreover, to self-verifying complexity classes, then (ii) we show that NL coincides with the class of all languages accepted by logspace bounded self-verifying Turing machines, (iii) the previous characterization induces a special variant of a graph reachability problem to be NL-complete, which (iv) is then used to prove Theorem 2.

The definition of self-verifying automata generalizes to Turing machines. In general, a self-verifying device A is allowed to give three possible answers "yes," "no," and "don't know." The machine A is *not* allowed to make mistakes: if the answer is "yes," then the input is in $L(A)$. On the other hand, if the answer is "no," then the input cannot be in the language $L(A)$. For every input there is at least one computation that does not finish with the answer "don't know." Notice that a self-verifying machine may have different computations on the same input, just like a nondeterministic machine. It is allowed that a computation ends with the answer "yes" (or "no") and another computation on the same input ends with "don't know," but it *cannot* be the case that there are computations on the same input such that one answers "yes" and the other answers "no."

Now we can introduce the self-verifying variants for space and time bounded complexity classes C, by referring to it as svC. For example, svNL refers to the class of languages accepted by logspace bounded self-verifying Turing machines. Then the first lemma on the way to prove the previous theorem reads as follows:

Lemma 3. $\mathsf{svNL} = \mathsf{NL}$.

Proof. The inclusion from left to right is obvious. For the converse inclusion, that is, $\mathsf{NL} \subseteq \mathsf{svNL}$, we argue as follows. Let $L \subseteq \Sigma^*$ be in NL. Then there is a nondeterministic logspace bounded Turing machine M that accepts L, i.e., $L = L(M)$. Since NL is closed under complementation [8, 13], there is also a nondeterministic logspace bounded Turing machine $\overline{M}$ that accepts the complement of L. In other words $L(\overline{M}) = \Sigma^* \setminus L$. Without loss of generality we may assume that both Turing machines M and $\overline{M}$ are polynomial time bounded.

From M and $\overline{M}$ we construct a self-verifying machine M' for L. The Turing machine M' operates as follows: it runs the Turing machine M on the given input w. If M halts in an accepting configuration, then the Turing machine M' answers "yes" and halts. In this way the input is accepted. Otherwise, if the computation of M halts in a non-accepting configuration, the Turing machine $\overline{M}$ is started. In case $\overline{M}$ halts in an accepting configuration, then M' answers "no" and thus rejects the input. Otherwise, the Turing machine M' answers "don't know." By construction it is easy to see that M' is logarithmically space bounded and moreover, is a self-verifying Turing machine. □

Remark 4. The proof of the previous lemma can be generalized such that the following identity holds

$$\mathsf{svC} = \mathsf{C} \cap \mathsf{coC},$$

where C is a two-way complexity class that is able to restart the computation from the very beginning, svC is the self-verifying variant of C, and coC refers to the complement class of C. In

particular this shows

$$\begin{aligned} \mathsf{svL} &= \mathsf{L}, \\ \mathsf{svP} &= \mathsf{P}, \\ \mathsf{svNP} &= \mathsf{NP} \cap \mathsf{coNP}, \end{aligned}$$

and

$$\mathsf{svPSPACE} = \mathsf{PSPACE}.$$

The details are left to the reader. Moreover one can show that

$$\mathsf{svNAuxPDASpTi}(\log n, \operatorname{pol} n) = \mathsf{NAuxPDASpTi}(\log n, \operatorname{pol} n),$$

where $\mathsf{NAuxPDASpTi}(s(n), t(n))$ refers to the class of languages that are accepted by nondeterministic auxiliary pushdown automata with $s(n)$ space and $t(n)$ time—this is a resource bounded Turing machine with a separate resource unbounded pushdown store. This result is based on the fact that this class is closed under complementation [2] and the above described simulation. A similar result also holds for the deterministic version of auxiliary pushdown automata. Both of these classes are of special interest, beause they are closely related to context-free languages. In [12] it was shown that

$$\begin{aligned} \mathrm{LOG}(\mathsf{DCFL}) &= \mathsf{DAuxPDASpTi}(\log n, \operatorname{pol} n) \\ \mathrm{LOG}(\mathsf{CFL}) &= \mathsf{NAuxPDASpTi}(\log n, \operatorname{pol} n), \end{aligned}$$

where DCFL (CFL, respectively) refers to the class of languages accepted by deterministic (nondeterministic, respectively) pushdown automata and $\mathrm{LOG}(\mathsf{C})$ refers to the closure of C under deterministic logarithmic space boundedn many-one reduction, that is,

$$\mathrm{LOG}(\mathsf{C}) = \{\, L \mid L \leq_m^{\log} L', \text{ for some } L' \in \mathsf{C} \,\}.$$

In the furthcoming we abbreviate nondeterministic pushdown automata by NPDA and deterministic pushdown automata by DPDA, respectively. This nice relation between the formal language class of the (determinstic) context-free languages and the appropriate complexity classes rises the question what can be said about the logspace closure of the self-verifying variants of the (determinstic) context-free languages. Let svDCFL (svCFL, respectively) refer to the class of languages accepted by self-verifying DPDAs (NPDAs, respectively). Since this classes are defined by one-way automata, the above given simulation used in the proof of Lemma 3 is not applicable. Neverthless, by a slight modificaiton of this simulation we find the following situation.

Theorem 5. $\mathsf{svDCFL} = \mathsf{DCFL} \subset \mathsf{svCFL} = \mathsf{CFL} \cap \mathsf{coCFL} \subset \mathsf{CFL}$.

Proof. Let $L \subseteq \Sigma^*$ be accepted by a self-verifying pushdown automaton. Regardless whether this automaton is deterministic or nondeterministic one obviously obtains two automata, namely one for the language L and an other one for the complement of L, i.e., for the language $\Sigma^* \setminus L$. Hence we get the inclusions

$$\mathsf{svDCFL} \subseteq \mathsf{DCFL} \cap \mathsf{coDCFL} \quad \text{and} \quad \mathsf{svCFL} \subseteq \mathsf{CFL} \cap \mathsf{coCFL}.$$

Because DCFL is closed under complementation [5], the former inclusion simplifies to $\mathsf{svDCFL} \subseteq \mathsf{DCFL}$. Moreover, interpreting an ordinary DPDA as a self-verifying automaton by identifying the accepting states with "yes" answers and the non-accepting states with "no" answers show

the converse inclusion $\mathsf{DCFL} \subseteq \mathsf{svDCFL}$. Thus, $\mathsf{svDCFL} = \mathsf{DCFL}$. For self-verifying NPDAs we use a slightly different argumentation. Let $L \in \mathsf{CFL} \cap \mathsf{svCFL}$ be a language over the alphabet Σ. Then there is a NPDA A ($\overline{A}$, respectively) for the language L ($\Sigma^* \setminus L$, respectively). From A and $\overline{A}$ we construct a self-verifying pushdown autamaton A' for L. The automaton A' operates as follows: it guesses either to run A or $\overline{A}$ on the given inpt w. If it runs A and it halts in an accepting configuration, then the pushdown automaton A' answers "yes" and halts. In case $\overline{A}$ was choosen to be executed and it halts in an accepting configuration, then the device A' answers "no" and halts. In all other cases A' returns "I don't know" as answer. By construction it is easy to see that A' is a self-verifying NPDA accepting L. Thus, $\mathsf{CFL} \cap \mathsf{coCFL} \subseteq \mathsf{CFL}$ and therefore $\mathsf{svCFL} = \mathsf{CFL} \cap \mathsf{coCFL}$. Finally, the remaining inclusion $\mathsf{svCFL} \subseteq \mathsf{CFL}$ is trivial.

It remains to prove the strictness of the inclusions. The strictness of the first inlcusion $\mathsf{DCFL} \subseteq \mathsf{svCFL}$ can be seen by the language

$$L = \{\, a^n b^n c \mid n \geq 0 \,\} \cup \{\, a^n b^{2n} d \mid n \geq 0 \,\}.$$

It is well known that this language is *not* determinsitic context free. A self-verifying NPDA accepts this language as follows: first it guesses whether the input ends with a letter c or a d. In the former case it verifies deterministically whether the input is of the form $a^n b^n c$, for some $n \geq 0$. If this is the case, the it answers "yes;" if this is not the case, then the answer is "no" in case the last letter was a c, while it is "I don't know" otherwise. The latter case is treated in a similar fashion. The self-verifying pushdown machine checks determinisically whether the input is of the form $a^n b^{2n} d$, for some $n \geq 0$. If it is of the form, then the automaton answers "yes." If it is not of this form and ends with a d, then the answer is "no;" otherwise it is "I don't know." It is obvious that the constructed automaton accepts L. Therefore $\mathsf{DCFL} \subset \mathsf{svCFL}$. The strictness of $\mathsf{svCFL} \subseteq \mathsf{CFL}$ easily follows, since CFL is *not* closed under complementation in general, and thus $\mathsf{CFL} \cap \mathsf{coCFL}$, which is equal to svCFL, is a strict subset of CFL. Therefore $\mathsf{svCFL} \subset \mathsf{CFL}$. □

The proof reveals that the characterization $\mathsf{svC} = \mathsf{C} \cap \mathsf{coC}$ also holds in case C is a one-way formal language or complexity class that is

1. deterministic or
2. nondeterministic and is able to perform a guess at the very beginning of the computation in order to start two subcomputations.

Yet another application of this characterization would, e.g., the class of languages accepted by self-verifying nondeterminsitic one-turn pushdown automata, which is then equal to $\mathsf{LIN} \cap \mathsf{coLIN}$, where LIN refers to the class of linear context-free languages. With a similar proof as Theorem 5 one can show the next result, where svDLIN (svLIN, respectively) refer to the class of languages accepted by self-verifying one-turn DPDAs (one-turn NPDAs, respectively).

Theorem 6. $\mathsf{svDLIN} = \mathsf{DLIN} \subset \mathsf{svLIN} = \mathsf{LIN} \cap \mathsf{coLIN} \subset \mathsf{LIN}$. □

Now let us come back to the logspace many-one closure of svCFL. The next theorem shows that this is equal to $\mathrm{LOG}(\mathsf{CFL}) = \mathsf{NAuxPDASpTi}(\log n, \mathrm{pol}\, n)$.

Theorem 7. $\mathrm{LOG}(\mathsf{svCFL}) = \mathrm{LOG}(\mathsf{CFL})$.

Proof. We have $\mathrm{LOG}(\mathsf{CFL}) = \mathsf{NAuxPDASpTi}(\log n, \mathrm{pol}\, n)$, which in turn is equal to

$$\mathsf{svNAuxPDASpTi}(\log n, \mathrm{pol}\, n)$$

by our previous investigation. Therefore it suffices to show

$$\mathrm{LOG}(\mathsf{svCFL}) = \mathsf{svNAuxPDASpTi}(\log n, \mathrm{pol}\, n),$$

where the inclusion from left to right already follows by $\mathsf{svCFL} \subset \mathsf{CFL}$. Here we use the fact that the inclusion $\mathsf{C} \subseteq \mathsf{D}$ implies $\mathrm{LOG}(\mathsf{C}) \subseteq \mathrm{LOG}(\mathsf{D})$, and the aforementioned equalities. Thus it remains to show that $\mathsf{svNAuxPDASpTi}(\log n, \mathrm{pol}\, n) \subseteq \mathrm{LOG}(\mathsf{svCFL})$. By well-known techniques auxiliary pushdown automata with logarithmic space bounded working tape are identical to (two-way) k-head pushdown automaton, for some $k \geq 1$. This also holds true for a polynomial time constraints and for self-verifying devices. In [12] the following results were shown:

1. If L is accepted by a two-way NPDA in polynomial time, then $L \leq_m^{\log} L'$ for some context-free language L'.
2. Language L is accepted by a two-way $2k$-head NPDA in polynomial time if and only if $T(L) = \{\, (\#w\$)^{|\#w\$|} \mid w \in L \,\}$ is accepted by a two-way k-head NPDA in polynomial time.

These results imply that $\mathsf{NAuxPDASpTi}(\log n, \mathrm{pol}\, n) \subseteq \mathrm{LOG}(\mathsf{CFL})$. A careful inspection of the proofs of these statements reveal that they also work for self-verifying pushdown automata and self-verifying context-free languages. Therefore, we conclude

$$\mathsf{svNAuxPDASpTi}(\log n, \mathrm{pol}\, n) \subseteq \mathrm{LOG}(\mathsf{svCFL}),$$

which proves the stated claim. □

Finally we show that the self-verifying property for pushdown automata is *not* decidable, in general, and becomes decidable when ocnsidering unary pushwon automata. For the former result we use a tool that is widely used to proof undecability results, namely the set $\mathsf{VAL}(M)$ of *valid computations* of a Turing machine M, which basically contains the histories of accepting Turing machine computations, see, e.g., [7]. Now we are ready for the next theorem.

Theorem 8. *Given a pushdown automaton A with accepting and rejecting states, it is undecidable ito determine whether A is a self-verifying automaton, i.e., whether it fulfills the self-verifying property. The problem becomes decidable if A is a unary pushdown automaton.*

Proof. Let M be a Turing machine. We consider the set $\mathsf{INVAL}(M)$ of *invalid computation* of M, which is the complement of $\mathsf{VAL}(M)$ for a suitable encoding of the alphabet. Obviously, for some Turing machine M the set $\mathsf{INVAL}(M)$ is universal if and only if $L(M)$ is empty. Moreover for a given Turing machine M one can effectively construct a context-free grammar or equivalently a nondetermnistic pushdown automaton A accepting $\mathsf{INVAL}(M)$—see, e.g., [7]. Now interpret A as a pushdown automaton with accepting and neutral states, but with an empty set of rejecting states. But then we have $L(M) = \emptyset$ if and only if $L(A)$ is universal. In case $L(A)$ is universal, the newly interpreted device is a self-verifying pushdown automaton; otherwise if $L(A)$ is not universal, there is a word $w \notin L(A)$ and thus A is not a self-verifying pushdown automaton since the self-verifying property is violated by w. Therefore $L(M) = \emptyset$ if and only if A is a self-veryfing pushdown automaton. Since checking emptiness for Turing machines is undecidable, we conclude that it is undeciable to deterime whether a pushdown automaton with accepting and rejecting states fulfills the self-verifying property.

In case the pushdown automaton A with accepting and rejeting states accepts unary languages only, we argue as follows: it is well known that a context-free language definded over a one-letter alphabet is regular and that moreover one can effectively transform it into an equivalent nondeterministic finite automaton—see,e.g., [11]. Thus one can construct NFAs B and B'

such that $L(B) = L_+(A)$ and $L(B') = L_-(A)$. Then A satisfies the self-verifying property if and only if (i) $L(B) \cup L(B')$ is universal w.r.t. the one-eltter alphabet and (ii) $L(B) \cap L(B') = \emptyset$. Because these two properties are obviously decidable for finite state devices, the claim follows. □

This closes our detour on the computational complexity of self-verifying devices. □

Next we introduce a variant of the NL-complete graph reachability problem that incorporates the self-verifying property:

– SwitchGap: Given a direct graph G, source node s and two target nodes t_+ and t_- with the promise that there is a path from s to either t_+ or t_-, but not to both. Is there a path from the source s to target node t_+?

Observe, that either node t_+ or t_- is reachable from s, but not simultaneously—this will be used later to control the self-verifying property of the to be constructed automaton. We now show that SwitchGap is NL-complete.

Lemma 9. SwitchGap *is* NL*-complete, even if the two target nodes of the instance do not have any out-going transitions.*

Proof. The containment in NL is obvious, since we have to search for a path linking s and t_+, as for the usual graph accessibility problem. Therefore SwitchGap is contained in NL. Observe, that we can even check in NL whether the given instance fulfills the promise on the reachability of the target nodes. In order to show this one uses the fact that NL is closed under complementation [8, 13]. The details are left to the reader.

For the NL-hardness we argue as follows: let M be a nondeterministic logspace bounded Turing machine. By Lemma 3 we can construct an equivalent logspace bounded self-verifying Turing machine M', with $L(M') = L(M)$. We may safely assume that M' has only one halting configuration which answers "yes" and one other halting configuration that answers "no." Let us refer to these configurations as C_{yes} and C_{no}, respectively. For a given word w we construct the configuration graph of the Turing machine M' in the usual way. Let G be this graph. Then G together with the unique initial configuration C_{init} and the target nodes C_{yes} and C_{no} form an instance of the SwitchGap problem. Observe that, since C_{yes} and C_{no} are halting configurations, both configurations do not have out-going edges in the configuration graph G. It is easy to see that this instance can be constructed within deterministic logspace and that it satisfies the promise of the problem under consideration. This shows that SwitchGap is NL-hard, and with the containment of the problem in NL we deduce that SwitchGap is NL-complete. □

Now we are ready to prove one of the main theorems of this note.

Proof (of Theorem 2). The containment of the general membership problem for self-verifying finite automata in NL is obvious, since already the general membership problem of ordinary nondeterministic finite automata belongs to this complexity class. It remains to show NL-hardness.

Let $G = (V, E)$ with the source node s and the target nodes t_+ and t_- be an instance of the SwitchGap problem. Without loss of generality we may assume that both target nodes do not have any out-going edges. From that instance we construct a self-verifying finite automaton A and a word w such that there is a path from s to t_+ if and only if w is accepted by A. We first modify the SwitchGap instance such that every node has at least one out-going edge, by introducing self-loops. This does not change the solvability status of the problem under

consideration. Moreover, by this we ensure that if there is a path from s to t_+ (or t_-), then there is a path that has length $|V| - 1$.

We now construct the automaton $A = (Q, \Sigma, \delta, q_0, F_+, F_-)$ with input alphabet $\Sigma = \{a\}$, state set

$$Q = V \times \{1, 2, \ldots, |V| + 1\},$$

from which $q_0 = \langle s, 1\rangle$ is the initial state, and the transition function reads as

$$\delta(\langle u, i\rangle, a) = \begin{cases} \bigcup_{(u,v)\in E}\{\langle v, i+1\rangle\} & \text{if } 1 \leq i \leq |V| \\ \{\langle u, |V|+1\rangle\} & \text{otherwise.} \end{cases}$$

Almost all states of A will answer "no." To be more precise, all states $\langle v, i\rangle$ with second component $i \neq |V|$ will answer "no." These states take care about words that are too short or too long. All states $\langle v, |V|\rangle$, except the two states $\langle t_+, |V|\rangle$ and $\langle t_-, |V|\rangle$ are "don't know" answering states. Finally, state $\langle t_+, |V|\rangle$ answers "yes" and state $\langle t_-, |V|\rangle$ answers "no." Thus,

$$F_+ = \{\, \langle t_+, |V|\rangle\}$$

and

$$F_R = \{\, \langle v, i\rangle \mid v \in V \text{ and } i \in \{1, 2, \ldots, |V| - 1, |V| + 1\}\,\} \cup \{\langle t_-, |V|\rangle\}.$$

The input word for the instance of the general membership problem is $a^{|V|-1}$. This completes the description of problem instance, which clearly can be constructed in deterministic logspace. By construction, automaton A is an svFA, which can either accept only the word $a^{|V|-1}$ or no word at all. Hence the graph G, together with the source node s and both target nodes t_+ and t_- is a positive instance of SwitchGap if and only if the svFA A accepts the word $a^{|V|-1}$. This shows NL-hardness and our claim follows. □

A slight modification of the previous proof also shows that the promise variant of the universality problem for svFAs is NL-complete, too. Containment in NL is obvious, since one has only to verify that there is no rejecting computation at all. This is a simple reachability problem and thus can be solved in nondeterministic logspace. For the hardness, one has only to re-define the accepting and rejecting states of the constructed automaton A such that all words that are too short or too long will be accepted, while the states $\langle t_+, |V|\rangle$ and $\langle t_-, |V|\rangle$ remain as they were. This results in the following theorem

Theorem 10. *The promise version of the universality problem for self-verifying finite automata is* NL*-complete.* □

Finally, it is easy to see that also the promise version of the (non-)emptiness problem for svFA is NL-complete, too, since the problem is already NL-hard for DFAs and contained in NL.

Theorem 11. *The promise version of the emptiness problem for self-verifying finite automata is* NL*-complete.* □

In the remainder of this note we consider counting problems. Counting problems for determinsitic and nondeterminstic finite automata were studied in [1]. In particular, the *census function problem* and the *complement of the census function problem* were investigated. The former problem is defined as follos: given a finite automaton A and a 1^n, how many words of length n are accepted by A? The difference to the complement variant is, that one asks for how many words are *not* accepted by the automaton? For DFAs both problems are #L-complete,

while for NFAs the census problem is span-L-complete and the complement of the census problem is even #P-complete [1]. Here #L (#P, respectively) refers to the class of those functions f, where f is the number of accepting paths of an NL (NP, respectively) machine and span-L is the class of those functions computable as $|S|$, where S is the set of output values returned by the accepting paths of an NL transducer. The inclusion chain $\#\mathsf{L} \subseteq \text{span-L} \subseteq \#\mathsf{P}$ is known. For more on counting classes we refer to [1] and [14, 15]. For svFAs we find the following situation.

Theorem 12. *The promise version of the census and the complement of the census problem for self-verifying finite automata are* span-L*-complete.*

Proof. First we consider the census problem for svFAs. The span-L uper bound immediately follows from the above mentioned result of [1], because svFAs are special kinds of NFAs. For the lower bound we cannot rely on the fact $\mathsf{NL} = \mathsf{svNL}$, since this equality is a language equivalence and not a functional equivalence. Neverthless, a close inspection of the proof of Theorem 3 reveals that it preserves the output on accepting paths of the NL machine, which in our case is a transducer. Thus, any function in span-L can be implemented on a svNL transducer. Hence, proceeding as in the proof of Lemma 9 one may construct an svFA(instead of a SwitchGap instance), together with a word 1^n, where n is the running time of the svNL transducer. This shows the span-L lower bound, and therefore we deduce that the census problem for svFAs is span-L-complete.

For the complement of the census problem we first exchange accepting and rejecting states of the underlying svFA. Here this can be done without any further cost, since we are dealing a self-verifying devices. Then we argue in similar veins as above. Hence, proving span-L-completeness in this case, too. □

References

1. C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoret. Comput. Sci.*, 107(1):3–30, 1993.
2. G. Buntrock, L. A. Hemachandra, and D. Siefkes. Using inductive counting to simulate nondeterministic computation. *Inform. Comput.*, 102:102–117, 1993.
3. S. Cho and D. T. Huynh. The parallel complexity of finite-state automata problems. *Inform. Comput.*, 97:1–22, 1992.
4. P. Duris, J. Hromkovic, J. D. P. Rolim, and G. Schnitger. Las Vegas versus determinism for one-way communication complexity, finite automata, and polynomial-time computations. In R. Reischuk and M. Morvan, editors, *Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science*, number 1200 in LNCS, pages 117–128, Lübeck, Germany, February–March 1997. Springer.
5. S. Ginsburg and S. Greibach. Deterministic context free languages. *Inform. Control*, 9(6):620–648, 1966.
6. M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.
7. M. Holzer and M. Kutrib. Descriptional complexity—an introductory survey. In C. Martín-Vide, editor, *Scientific Applications of Language Methods*, pages 1–58. World Scientific, 2010.
8. N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.
9. N. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. System Sci.*, 11:68–85, 1975.
10. A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE Society Press, 1972.
11. G. Pighizzini, J. SHallit, and M.-W. Wang. Unary context-free grammars and pushwon automata, descriptional complexity and auxiliary space lower bounds. *J. Comput. System Sci.*, 65:393–414, 2002.
12. I. H. Sudborough. On the tape complexity of deterministic context-free languages. *J. ACM*, 25(3):405–414, 1978.
13. R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inform.*, 26(3):279–284, 1988.
14. L. G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8(2):189–201, 1979.
15. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

Institut für Informatik

Justus-Liebig-Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany

Recent Reports

(Further reports are available at `www.informatik.uni-giessen.de`.)

S. Beier, M. Holzer, M. Kutrib, *On the Descriptional Complexity of Operations on Semilinear Sets*, Report 1701, April 2017.

M. Holzer, S. Jakobi, M. Wendlandt, *On the Computational Complexity of Partial Word Automata Problems*, Report 1404, May 2014.

H. Gruber, M. Holzer, *Regular Expressions From Deterministic Finite Automata, Revisited*, Report 1403, May 2014.

M. Kutrib, A. Malcher, M. Wendlandt, *Deterministic Set Automata*, Report 1402, April 2014.

M. Holzer, S. Jakobi, *Minimal and Hyper-Minimal Biautomata*, Report 1401, March 2014.

J. Kari, M. Kutrib, A. Malcher (Eds.), *19th International Workshop on Cellular Automata and Discrete Complex Systems AUTOMATA 2013 Exploratory Papers*, Report 1302, September 2013.

M. Holzer, S. Jakobi, *Minimization, Characterizations, and Nondeterminism for Biautomata*, Report 1301, April 2013.

A. Malcher, K. Meckel, C. Mereghetti, B. Palano, *Descriptional Complexity of Pushdown Store Languages*, Report 1203, May 2012.

M. Holzer, S. Jakobi, *On the Complexity of Rolling Block and Alice Mazes*, Report 1202, March 2012.

M. Holzer, S. Jakobi, *Grid Graphs with Diagonal Edges and the Complexity of Xmas Mazes*, Report 1201, January 2012.

H. Gruber, S. Gulan, *Simplifying Regular Expressions: A Quantitative Perspective*, Report 0904, August 2009.

M. Kutrib, A. Malcher, *Cellular Automata with Sparse Communication*, Report 0903, May 2009.

M. Holzer, A. Maletti, *An $n \log n$ Algorithm for Hyper-Minimizing States in a (Minimized) Deterministic Automaton*, Report 0902, April 2009.

H. Gruber, M. Holzer, *Tight Bounds on the Descriptional Complexity of Regular Expressions*, Report 0901, February 2009.

M. Holzer, M. Kutrib, and A. Malcher (Eds.), *18. Theorietag Automaten und Formale Sprachen*, Report 0801, September 2008.

M. Holzer, M. Kutrib, *Flip-Pushdown Automata: Nondeterminism is Better than Determinism*, Report 0301, February 2003

M. Holzer, M. Kutrib, *Flip-Pushdown Automata: $k + 1$ Pushdown Reversals are Better Than k*, Report 0206, November 2002

M. Holzer, M. Kutrib, *Nondeterministic Descriptional Complexity of Regular Languages*, Report 0205, September 2002

H. Bordihn, M. Holzer, M. Kutrib, *Economy of Description for Basic Constructions on Rational Transductions*, Report 0204, July 2002

M. Kutrib, J.-T. Löwe, *String Transformation for n-dimensional Image Compression*, Report 0203, May 2002

A. Klein, M. Kutrib, *Grammars with Scattered Nonterminals*, Report 0202, February 2002

A. Klein, M. Kutrib, *Self-Assembling Finite Automata*, Report 0201, January 2002

M. Holzer, M. Kutrib, *Unary Language Operations and its Nondeterministic State Complexity*, Report 0107, November 2001

A. Klein, M. Kutrib, *Fast One-Way Cellular Automata*, Report 0106, September 2001

M. Holzer, M. Kutrib, *Improving Raster Image Run-Length Encoding Using Data Order*, Report 0105, July 2001

M. Kutrib, *Refining Nondeterminism Below Linear-Time*, Report 0104, June 2001

M. Holzer, M. Kutrib, *State Complexity of Basic Operations on Nondeterministic Finite Automata*, Report 0103, April 2001

M. Kutrib, J.-T. Löwe, *Massively Parallel Fault Tolerant Computations on Syntactical Patterns*, Report 0102, March 2001

A. Klein, M. Kutrib, *A Time Hierarchy for Bounded One-Way Cellular Automata*, Report 0101, January 2001

M. Kutrib, *Below Linear-Time: Dimensions versus Time*, Report 0005, November 2000