I F I G
Research
Report

Institut für Informatik

On the Computational Complexity of Partial Word Automata Problems

Markus Holzer Sebastian Jakobi Matthias Wendlandt

IFIG Research Report 1404
May 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

IFIG Research Report
IFIG Research Report 1404, May 2014

On the Computational Complexity of Partial Word Automata Problems

Markus Holzer,[1] Sebastian Jakobi,[2] and Matthias Wendlandt[3]

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

Abstract. We consider computational complexity of problems related to partial word automata. Roughly speaking, a partial word is a word in which some positions are unspecified, and a partial word automaton is a finite automaton that accepts a partial word language—here the unspecified positions in the word are represented by a "hole" symbol $\diamond$. A partial word language L' can be transformed into an ordinary language L by using a $\diamond$-substitution. In particular, we investigate the complexity of the compression or minimization problem for partial word automata, which is known to be NP-hard. We improve on the previously known complexity on this problem, by showing PSPACE-completeness. In fact, it turns out that almost all problems related to partial word automata, such as, e.g., equivalence and universality, are already PSPACE-complete. Moreover, we also study these problems under the further restriction that the involved automata accept only finite languages. In this case, the complexity of the studied problems drop from PSPACE-completeness down to coNP-hardness and containment in Σ_2^{P} depending on the problem investigated.

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: partial words, finite automata, computational complexity, language problems, minimization problem

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

1 Introduction

While a word is just a sequence of letters form an alphabet, a *partial* word or a word with "don't cares" is a word in which certain positions are unspecified. Usually, these undefined positions are represented by a "hole" symbol $\diamond$. Partial words were introduced [9] in the mid 1970s. During the last two decades a vast amount on literature appeared on the combinatorics and algorithms on partial words—we refer to [5] for further reading. In a recent attempt [8], partial words were linked to regular languages. The motivation for this approach was to use these partial words to compress the representation of ordinary word languages. To this end, one has to specify the possible contents of the holes. For instance, the finite four-element language $L = \{ab, bb, ac, bc\}$ can be represented by the two-element language $L' = \{\diamond b, \diamond c\}$ on partial words, when replacing a hole $\diamond$ by either the letter a or b, resulting in the words ab, ac in the former case and in the words bb, bc in the latter. Mathematically speaking, the replacement of the holes can be modeled by a substitution $\sigma : (\Sigma \cup \{\diamond\})^* \to 2^{\Sigma^*}$ satisfying $\sigma(a) = a$, for $a \in \Sigma$ and $\sigma(\diamond) \subseteq \Sigma$, which is called a $\diamond$-*substitution*. More generally, this approach asks whether a language L can be compressed, i.e., if there is a partial word language L' smaller in size than L and a $\diamond$-substitution σ such that $L = \sigma(L')$. For instance, if one uses the number of words in a finite (partial) language L as a size measure, then the language $L = \{ab, ba\}$ over the alphabet $\{a, b\}$ cannot be compressed at all, which is seen by an easy argument. With this formulation the problem can be seen as a minimization problem. The answer to this question obviously triggers algorithmical, descriptional complexity theoretical, and computational complexity theoretical considerations, since the answer is not always positive.

For the representation of the (partial) word languages it is convenient to use deterministic finite automata (DFAs). In fact, besides [8], also [2] and [6] stick to this representation. In the former two papers, descriptional and computational complexity issues related to the compression based on partial words are studied, while in the latter paper [6], an approximation algorithm for the compression of finite languages is given. Moreover, there it is also mentioned, that the minimization problem, that is, asking whether a language given by a DFA that is associated to a $\diamond$-substitution σ can be compressed down to state size k by using σ on a partial word DFA, for short $\diamond$-DFA, is computationally intractable, namely NP-hard. This is deduced from a more general result on minimization of finite automata due to [4] that shows that minimization is NP-hard for all finite automata classes that subsume the class of unambiguous finite automata, allow at most one state with a nondeterministic transition for at most one alphabet symbol, and is restricted to visit this nondeterministic state at most once in a computation. In [8], it was also shown that the equivalence problem on partial word automata, that is, given a DFA A, a $\diamond$-DFA B, and $\diamond$-substitution σ, decide whether $L(A) = \sigma(L(B))$ holds, is coNP-hard. From the computation complexity point of view these are the only results known for partial word automata, up to our knowledge.

The aim of the present paper is to just about complete the picture on decision problems related to partial word automata. In particular, we investigate

the above mentioned compression or minimization problem and refine the previous intractability result to PSPACE-completeness. In fact, it turns out that almost all problems related to partial word automata such as equivalence and universality are PSPACE-complete, already for deterministic devices. A notable exception is the emptiness problem for partial word automata, which turns out to be NL-complete. Also the specific problem for DFAs on the essential σ-definability introduced in [8] is PSPACE-complete. Here a language $L \subseteq \Sigma^*$ is *essentially σ-definable* for a $\diamond$-substitution σ, if there is a partial word language $L' \subseteq (\Sigma \cup \{\diamond\})^*$, where every word in L' has a hole, such that $L = \sigma(L')$. All these results are in sharp contrast to results on ordinary DFAs, because most problems for DFAs are efficiently solvable. When restricting all these problems to devices that accept finite languages a drop to the lower levels of the polynomial hierarchy appears. To be more precise, the emptiness problem for partial word automata accepting finite languages remains NL-complete as in the general case. For the other considered problems, it turns out that there is a subtle difference, whether the $\diamond$-substitution is part of the input or not. The equivalence problem for partial word automata accepting finite languages is coNP-complete in case the $\diamond$-substitution is part of the input. If the existence of a $\diamond$-substitution that makes the two input automata equivalent is asked for, the problem is contained in Σ_2^{P} and remains coNP-hard. While for partial word automata problems in the general case, there is no difference in complexity if the $\diamond$-substitution is part of the input or is existentially asked for, here we are only able to prove an non-matching upper and lower bound. A similar situation appears for partial word automata accepting finite languages in case of the compressibility or minimization problem and for the question of being essentially σ-definable. These non-matching upper and lower bounds on some of the problems on partial word automata problem for finite languages is somehow related to the minimization problem on ordinary nondeterministic finite automata: here the exact complexity status is not completely revealed since it is NP-hard [1] and contained in Σ_2^{P}, because the equivalence problem for NFAs accepting finite languages is coNP-complete [15].

The paper is organized as follows. In the upcoming section we give basic definitions used throughout the paper. Section 3 studies basic language problems such as emptiness, universality, and equivalence for partial word automata, and in Section 4 we consider the problem of deciding whether a language is essentially σ-definable. Then Section 5 is devoted to the study of minimization problems for partial word automata. Finally, in Section 6 we reconsider all above mentioned problems for the special case, where all automata accept finite languages. Due to space constraints almost all proofs can be found in the Appendix.

2 Preliminaries

Let Σ be an alphabet. A *word* w over alphabet Σ is a possibly empty sequence $w = a_1 a_2 \dots a_n$ of elements $a_i \in \Sigma$, for $1 \leq i \leq n$, called *letters*. The length of a word w is denoted by $|w|$ and is equal to the number of letters of w; the *empty* word denoted by λ has length 0. The *concatenation* of two words v

and w is denoted by $v \cdot w$ and is equal to vw. The set Σ^* is the set of all words over Σ. It is a *monoid* with concatenation and identity element λ. A *language* L over Σ is a subset of Σ^*. The concatenation operation on words naturally extends to languages; more precisely, if L and M are languages over Σ, then $L \cdot M$ is defined to be $\{ vw \mid v \in L \text{ and } w \in M \}$. The language L is *regular* if $L = L(A)$ for some deterministic or nondeterministic finite automaton A. 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 $\delta : Q \times \Sigma^* \to 2^Q$. 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 $\delta\colon Q \times \Sigma \to Q$ is a total mapping.

Let $\diamond$ be a new symbol with $\diamond \notin \Sigma$, which is called the *hole symbol* or *hole*, for short. Set $\Sigma_\diamond := \Sigma \cup \{\diamond\}$. A word w over $\Sigma_\diamond$ is said to be a *partial word*. All the above introduced notations naturally carry over to partial words. In particular, a set $L \subseteq \Sigma_\diamond^*$ is a *partial word language*, and it is *regular* if and only if it is accepted by a DFA or NFA A with input alphabet $\Sigma_\diamond$ that is, $L = L(A)$. Thus, we treat the hole symbol $\diamond$ as an ordinary input letter. In order to distinguish between finite automata accepting ordinary languages from those accepting partial word languages, we refer to the latter one as $\diamond$-DFA or $\diamond$-NFA, respectively. Partial word languages can be transformed to ordinary languages by using $\diamond$-substitutions over Σ. A *$\diamond$-substitution over Σ* is a mapping $\sigma : \Sigma_\diamond^* \to 2^{\Sigma^*}$ satisfying (i) $\sigma(a) = \{a\}$, for every $a \in \Sigma$, (ii) $\sigma(\diamond) \subseteq \Sigma$, and (iii) $\sigma(vw) = \sigma(v) \cdot \sigma(w)$, for every $v, w \in \Sigma_\diamond^*$. Thus, σ is fully defined by $\sigma(\diamond)$. Applying σ to a partial word language $L \subseteq \Sigma_\diamond^*$ results in an ordinary language on words $\sigma(L) \subseteq \Sigma^*$. Sometimes we call the partial word language L the "compressed" version of $\sigma(L)$.

We classify problems on partial word automata w.r.t. their computational complexity. Consider the inclusion chain $\mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \Sigma_2^\mathsf{P} \subseteq \mathsf{PSPACE}$. Here NL refers to the set of problems accepted by nondeterministic 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. Moreover, Σ_2^P refers to the second level of the polynomial hierarchy, that is the set of problems accepted by polynomial time bounded alternating Turing machines, that start in an existential configuration and are allowed to alternate at most once. 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 reducibility.

Finally, let us recall what is known from a computational complexity perspective for problems on ordinary finite automata. Basic decision problems concerning the language accepted by automata are the emptiness, universality, and equivalence problems. Concerning the complexity of these decision problems for

finite automata, the following is known. The emptiness problem for finite automata is NL-complete, no matter whether DFAs or NFAs are considered [12]. For the other problems the situation is different. The universality problem and the equivalence problem are NL-complete for DFAs [7], and PSPACE-complete for NFAs [14]. For the decision version of the minimization problem for finite automata it is known that the DFA-to-DFA minimization is NL-complete [7], while the problem becomes PSPACE-complete [11] if at least one of the involved automata is allowed to be nondeterministic. For finite automata accepting finite languages, the situation is slightly different and more complex. In some cases a drop in complexity to compared to the general case is known. The emptiness problem for finite automata accepting finite languages remains NL-complete as in the general case, regardless whether DFAs or NFAs are considered. The bounded-universality problem is coNP-complete for NFAs [15], while it is NL-complete for DFAs [7]. Here bounded-universality asks whether the device under consideration accepts $\Sigma^{\leq \ell}$, for some given ℓ. Finally, the DFA-to-DFA minimization problem for automata accepting finite languages remains NL-complete [7] as in the general case. If NFAs accepting finite languages are considered, it is known that the minimization problem is contained in Σ_2^{P}, and is NP-hard [1]. Recently the lower bound was improved to DP-hardness, even if the input is a DFA [10]. The complexity class DP includes both NP and coNP.

Most of the hardness results in our paper are obtained by reductions from the *union universality problem* for DFAs, which is to decide for given DFAs $A_1, A_2, \ldots, A_n$ with common input alphabet Σ, whether $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$ holds. This problem is known to be PSPACE-complete, even if Σ is a binary alphabet [11, 13].

3 Basic Language Problems

We study the complexity of corresponding problems for partial word automata. We use the following notation. Let $\mathcal{X}$ be the class of $\diamond$-DFAs or the class of $\diamond$-NFAs. The *emptiness problem* $\mathcal{X}$-eq-$\emptyset$ is to decide for a given partial word automaton $A \in \mathcal{X}$ and a given $\diamond$-substitution σ, whether $\sigma(L(A)) = \emptyset$. Here the $\diamond$-substitution σ is given as input. However, one could also consider the following "existential"-variant of this problem: the problem $\mathcal{X}$-eq-$\emptyset_{(\exists\sigma)}$ is to decide for a given partial word automaton $A \in \mathcal{X}$, whether *there exists* some $\diamond$-substitution σ such that $\sigma(L(A)) = \emptyset$. Similarly, the problem $\mathcal{X}$-eq-$\emptyset_{(\forall\sigma)}$ asks whether $L(A) = \emptyset$ holds *for all* "appropriate" $\diamond$-substitutions σ. Here it has to be specified, what an appropriate $\diamond$-substitution should be. If A is a partial word automaton with input alphabet $\Sigma_\diamond$, then the $\diamond$-substitution σ satisfies $\sigma(\diamond) \subseteq \Sigma$. But still the question is, whether the *empty* $\diamond$-substitution σ with $\sigma(\diamond) = \emptyset$ is appropriate or not. If not stated otherwise, we only consider *non-empty* $\diamond$-substitutions in this case. Although at a first glance this seems to be a negligibility, it turns out that this issue may induce a significant difference in the complexity of the corresponding decision problems. Analogously to emptiness problems, we define the *universality problems* $\mathcal{X}$-eq-Σ^*, $\mathcal{X}$-eq-$\Sigma^*{}_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\Sigma^*{}_{(\forall\sigma)}$. Moreover, for automata classes $\mathcal{X}$ and $\mathcal{Y}$, which can be the classes of DFAs, NFAs, $\diamond$-DFAs, or $\diamond$-NFAs, we define the *equivalence problems*

$\mathcal{X}$-eq-$\mathcal{Y}$, $\mathcal{X}$-eq-$\mathcal{Y}_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\mathcal{Y}_{(\forall\sigma)}$. Here a $\diamond$-substitution σ of course only applies to particular word automata. For example the problem $\diamond$-DFA-eq-DFA$_{(\exists\sigma)}$ is to decide for a given $\diamond$-DFA A and a given DFA B, whether there exists a $\diamond$-substitution σ such hat $\sigma(L(A)) = L(B)$.

We first show that the different variants of the emptiness problem are NL-complete.

Theorem 1 (Emptiness). *For $\mathcal{X} \in \{\diamond\text{-DFA}, \diamond\text{-NFA}\}$ the emptiness problems $\mathcal{X}$-eq-$\emptyset$, $\mathcal{X}$-eq-$\emptyset_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\emptyset_{(\forall\sigma)}$ are* NL*-complete.*

Proof. First notice that if A is a $\diamond$-NFA with input alphabet $\Sigma_\diamond$ and σ is a $\diamond$-substitution with $\sigma(\diamond) \neq \emptyset$ then clearly the language $\sigma(L(A))$ is empty if and only if the partial word language $L(A)$ is empty. If $\sigma(\diamond) = \emptyset$ then the language $\sigma(L(A))$ is empty if and only if $\sigma(L(A)) \cap \Sigma^*$ is empty. Since the emptiness problem for NFAs is NL-complete, it follows that the emptiness problems for partial word automata can be solved in NL. For NL-hardness we use the fact that the emptiness problem for DFAs is NL-complete [12], even if the given DFA accepts a finite language. Since any DFA can be seen as an incomplete $\diamond$-DFA, NL-hardness of the three problems from the theorem easily follows by a simple reduction: given a DFA B, we construct a $\diamond$-DFA A by introducing a new non-accepting sink state, and $\diamond$-transitions to this state from all other states. Clearly $\sigma(L(A))$ is empty if and only if $L(B)$ is empty, no matter which $\diamond$-substitution σ is chosen. This also proves the lower bound for the corresponding $\diamond$-NFA problems. □

Next we study universality and equivalence problems for partial word automata. It turns out that all of these problems are PSPACE-complete, except for two degenerate variants, where the empty substitution is taken into account. Our main results on the universality problems will be summarized in Theorem 4, and results on equivalence problems will be summarized in Theorem 5. Before we come to these results, we prove two lemmas which provide upper and lower bounds for the complexities of the universality and equivalence problems. We start with PSPACE upper bounds for the equivalence problems.

Lemma 2. *The equivalence problems $\diamond$-NFA-eq-$\diamond$-NFA, $\diamond$-NFA-eq-$\diamond$-NFA$_{(\exists\sigma)}$, and $\diamond$-NFA-eq-$\diamond$-NFA$_{(\forall\sigma)}$ can be solved in* PSPACE*.*

Proof. The given automata A and B can easily be transformed into NFAs A' and B', with $L(A') = \sigma(L(A))$ and $L(B') = \sigma(L(B))$, by replacing all $\diamond$-transitions with transitions on symbols from $\sigma(\diamond)$. Hence the first problem from the lemma can be reduced to deciding equivalence of two NFAs, which can be solved in PSPACE. Moreover, the other two problems can be solved in PSPACE by iterating over all $\diamond$-substitutions σ with $\sigma(\diamond) \subseteq \Sigma$, where Σ is the set of input symbols different from $\diamond$ that appear in A or B. □

Next we provide PSPACE lower bounds for universality problems.

Lemma 3. *The problems $\diamond$-DFA-eq-Σ^*, $\diamond$-DFA-eq-$\Sigma^*{}_{(\exists\sigma)}$, and $\diamond$-DFA-eq-$\Sigma^*{}_{(\forall\sigma)}$ are* PSPACE*-hard.*

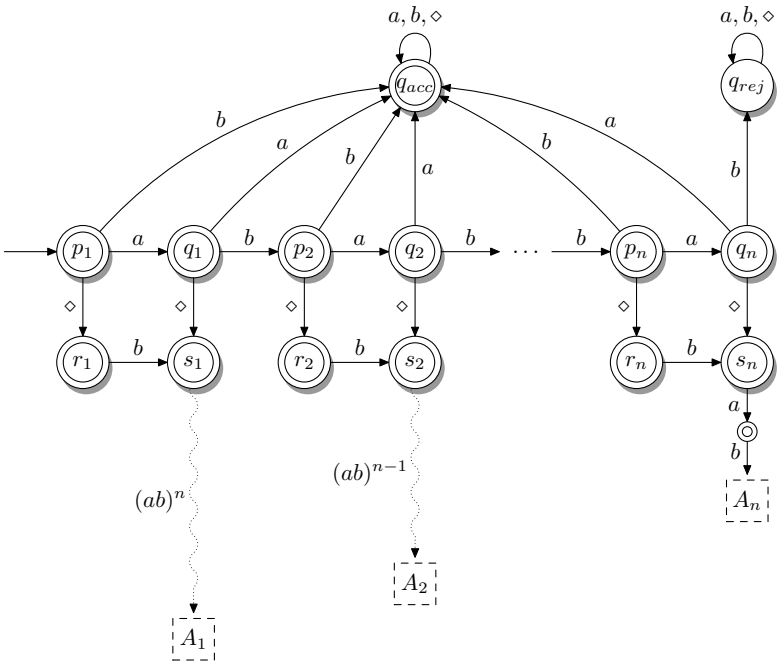

Fig. 1. The $\diamond$-DFA A constructed from the DFAs $A_1, A_2, \ldots, A_n$ over the common input alphabet $\{a, b\}$. For all numbers i, with $1 \leq i \leq n$, the state s_i is connected to the initial state of the DFA A_i by a path of length $2 \cdot (n + 1 - i)$ that consists of alternating a- and b-transitions. All states on these paths are accepting states. Transitions on a and b which are not shown lead to the accepting sink state q_{acc}, and $\diamond$-transitions which are not shown lead to the rejecting sink state q_{rej}.

Proof. Let us begin with the third problem of the lemma. We prove PSPACE-hardness by a reduction from the union universality problem for DFAs, which is well known to be PSPACE-complete, even if restricted to DFAs over a binary input alphabet. Therefore, let $A_1, A_2, \ldots, A_n$ be DFAs with common input alphabet $\Sigma = \{a, b\}$. We construct a $\diamond$-DFA A as shown in Figure 1. Clearly this construction can be carried out by a logarithmic space bounded Turing machine. In the following we prove that A satisfies $\sigma(L(A)) = \Sigma^*$ for all $\diamond$-substitutions σ with $\emptyset \neq \sigma(\diamond) \subseteq \Sigma$ if and only if $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$.

First assume that $\bigcup_{i=1}^{n} L(A_i) \neq \Sigma^*$, and choose $w \in \Sigma^*$ such that for all i with $1 \leq i \leq n$, we have $w \notin L(A_i)$. Then, for all appropriate $\diamond$-substitutions σ, the word $(ab)^{n+1}w$ does not belong to the language $\sigma(L(A))$. To see this, it is sufficient to consider the $\diamond$-substitution σ_Σ, with $\sigma_\Sigma(\diamond) = \{a, b\}$. By replacing in A all $\diamond$-transitions by transitions on a and b we obtain an NFA B for the language $\sigma_\Sigma(L(A))$. Now assume that B accepts the word $(ab)^{n+1}w$. After reading the prefix $(ab)^{n+1}$ the automaton B can only reach the initial states of the DFAs A_i, for $1 \leq i \leq n$, and the non-accepting sink state q_{rej}. Of course q_{rej} does not lead to an accepting state on any input, so the suffix w can only be accepted from some initial state of a DFA A_i, with $1 \leq i \leq n$. But this means that $w \in L(A_i)$, which is a contradiction. Thus, we have $w \notin \sigma_\Sigma(L(A))$, i.e., $\sigma(L(A)) = \Sigma^*$ does not hold for all appropriate $\diamond$-substitutions σ—in fact, we have shown that this equality does not hold for *any* $\diamond$-substitution.

For the reverse implication assume that $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$. Further let σ be a $\diamond$-substitution with $\emptyset \neq \sigma(\diamond) \subseteq \Sigma$. Notice that we have $a \in \sigma(\diamond)$ or $b \in \sigma(\diamond)$ because $\sigma(\diamond)$ is not empty. We have to show that $\sigma(L(A)) = \Sigma^*$. Therefore let B be the NFA for $\sigma(L(A))$ constructed from A by replacing the $\diamond$-transitions by transitions on all symbols from $\sigma(\diamond)$, and let $w \in \Sigma^*$.

First assume that w can be written as $w = (ab)^{n+1}v$ for some word $v \in \Sigma^*$. We have $v \in L(A_i)$ for some integer i with $1 \leq i \leq n$ because $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$. Then w can be accepted by B as follows. Reading the prefix $(ab)^{i-1}$ takes B to state p_i. From there, by reading ab, the state s_i can be reached either by passing through state q_i, if $b \in \sigma(\diamond)$, or by passing through state r_i, if $a \in \Sigma(\diamond)$. From state s_i the initial state of the DFA A_i is reached after reading $(ab)^{n+1-i}$, and the suffix v is accepted from there because $v \in L(A_i)$. Hence the word $w = (ab)^{n+1}v$ belongs to $\sigma(L(A))$.

Finally assume that w does not have the prefix $(ab)^{n+1}$. Then either the length of w is at most $2n + 1$, or its prefix of length $2n + 2$ does not obey the ab-structure. In the latter case the automaton B reaches the *accepting* sink state q_{acc}—recall that all transitions on symbols a and b which are not shown in Figure 1 lead to q_{acc}. In the remaining case, where the length of w is at most $2n+1$, the word is also accepted by B because either w leads to q_{acc} (due to violating the ab-structure) or w leads to some state "between" p_1 and the initial state of A_n. Therefore $\sigma(L(A)) = \Sigma^*$, which concludes the proof that the third problem of the lemma is PSPACE-hard.

In fact, we have also shown PSPACE-hardness for the second problem from the lemma because we have seen that if $\bigcup_{i=1}^{n} L(A_i) \neq \Sigma^*$, then *no* σ-substitution satisfies $\sigma(L(A)) = \Sigma^*$. To obtain a reduction that also proves the first problem of the lemma to be PSPACE-hard, we simply give the $\diamond$-substitution σ with $\sigma(\diamond) = \{a, b\}$ as additional input. □

Now we are ready to state our main results of this section. We start with the universality problems.

Theorem 4 (Universality). *For $\mathcal{X} \in \{\diamond\text{-DFA}, \diamond\text{-NFA}\}$ the universality problems $\mathcal{X}$-eq-Σ^*, $\mathcal{X}$-eq-$\Sigma^*{}_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\Sigma^*{}_{(\forall\sigma)}$ are* PSPACE*-complete.*

Proof. Lemma 3 provides the lower bounds of PSPACE-hardness. Containment of the problems in PSPACE can be deduced from Lemma 2, by choosing the $\diamond$-NFA B in the problem descriptions of that lemma to be the single-state $\diamond$-DFA accepting the language $\Sigma^*_\diamond$. □

Our results on the different equivalence problems are as follows.

Theorem 5 (Equivalence). *The equivalence problems $\mathcal{X}$-eq-$\mathcal{Y}$, $\mathcal{X}$-eq-$\mathcal{Y}_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\mathcal{Y}_{(\forall\sigma)}$, for $\mathcal{X} \in \{\diamond\text{-DFA}, \diamond\text{-NFA}\}$ and $\mathcal{Y} \in \{\diamond\text{-DFA}, \diamond\text{-NFA}, \text{DFA}, \text{NFA}\}$, are* PSPACE*-complete.*

Proof. For proving PSPACE-hardness, we reduce the universality problems from Lemma 3 to the equivalence problems of the present theorem by choosing the automaton B as the single-state DFA (or $\diamond$-DFA) for the language Σ^* (or $\Sigma^*_\diamond$, respectively). Containment in PSPACE follows from Lemma 2. □

Finally we show that the "for all"-variants of the universality and equivalence problems for $\diamond$-DFAs become much easier if also the empty $\diamond$-substitution σ with $\sigma(\diamond) = \emptyset$ is considered:

Theorem 6. *The problem of deciding for a given $\diamond$-DFA A, and a DFA B with input alphabet Σ, whether for all $\diamond$-substitutions σ, with $\sigma(\diamond) \subseteq \Sigma$, we have $\sigma(L(A)) = L(B)$, is* NL*-complete.*

Proof. By choosing B to be the single-state DFA for the empty language, the lower bound of NL-hardness follows from Theorem 1.

It remains to prove containment in NL. Let A be a $\diamond$-DFA and B be a DFA with input alphabet Σ. We have to decide whether $\sigma(L(A)) = L(B)$ holds for all $\diamond$-substitutions σ with $\sigma(\diamond) \subseteq \Sigma$. In order to decide this question, we only need to consider the two $\diamond$-substitutions $\sigma_\emptyset \colon \diamond \mapsto \emptyset$ and $\sigma_\Sigma \colon \diamond \mapsto \Sigma$. Our algorithm works as follows. First we decide whether $\sigma_\emptyset(L(A)) = L(B)$. If this is not true, then obviously the answer to the equivalence problem at hand is "no." Otherwise we check $\sigma_\Sigma(L(A)) \subseteq L(B)$. Again, if this is not true, then we can safely answer "no." Now assume we have $\sigma_\emptyset(L(A)) = L(B)$ and $\sigma_\Sigma(L(A)) \subseteq L(B)$. In this case the answer is "yes," which can be seen as follows. We know that for all $\diamond$-substitutions σ with $\sigma(\diamond) \subseteq \Sigma$ we have

$$L(B) = \sigma_\emptyset(L(A)) \subseteq \sigma(L(A)) \subseteq \sigma_\Sigma(L(A)) \subseteq L(B),$$

hence $\sigma(L(A)) = L(B)$—here we use the fact that for all $\diamond$-substitutions σ_1 and σ_2, with $\sigma_1(\diamond) \subseteq \sigma_2(\diamond)$, and all languages L we have $\sigma_1(L) \subseteq \sigma_2(L)$.

The fact that the described algorithm can be implemented in NL can be seen as follows. The test whether $\sigma_\emptyset(L(A)) = L(B)$ boils down to testing equivalence of the two DFAs B and $A_\emptyset$, where $A_\emptyset$ is obtained from A by ignoring the $\diamond$-transitions, and equivalence of DFAs can be decided in NL. Moreover also $\sigma_\Sigma(L(A)) \subseteq L(B)$ can be decided in NL by checking $\sigma_\Sigma(L(A)) \cap \overline{L(B)} \neq \emptyset$—here $\overline{L(B)}$ is the set $\Sigma^* \setminus L(B)$: we can transform A into an NFA for $\sigma_\Sigma(L(A))$ by re-labeling the $\diamond$-transitions with the letters from Σ, and we transform the DFA B into a DFA for the language $\overline{L(B)}$. From these automata an NFA for the language $\sigma_\Sigma(L(A)) \cap \overline{L(B)}$ can be constructed by a logarithmic space bounded Turing machine. So deciding $\sigma_\Sigma(L(A)) \subseteq L(B)$ can be reduced to deciding (non-)emptiness of an NFA language, which can be solved in NL [12]. □

Because testing universality can be reduced to testing equivalence to a DFA for Σ^*, Theorem 6 readily implies the following result.

Theorem 7. *The problem of deciding for a given $\diamond$-DFA A and an alphabet Σ, whether for all $\diamond$-substitutions σ, with $\sigma(\diamond) \subseteq \Sigma$, we have $\sigma(L(A)) = \Sigma^*$, is* NL*-complete.*

Proof. Containment in NL follows from Theorem 6, and NL-hardness from the fact that the universality problem for DFAs is NL-complete. □

4 Definability Problems

In [8] the authors introduce a property of languages called essentially σ-definable. They give the following definition.

Definition 8. *Let $L \subseteq \Sigma^*$ be a language and σ be a $\diamond$-substitution over Σ. We say that L is σ-defined by the language L', where $L' \subseteq (\Sigma \cup \{\diamond\})^*$ is a partial word language, if $L = \sigma(L')$. Moreover, we say that L is essentially σ-defined by L', where $L' \subseteq (\Sigma \cup \{\diamond\})^*$, if $L = \sigma(L')$ and every word in L' contains at least one $\diamond$-symbol.*

One of the main questions is, given a language L and a $\diamond$-substitution σ, whether L essentially σ-definable. For example, the language $L = \{aa, bb\}$ is not essentially σ-definable no matter how the substitution is chosen. It is not possible to choose a single symbol for the substitution, because both words of L consist either of a's or of b's. Substitutions with more than the two symbols a and b are meaningless and if the substitution is $\{a, b\}$ it leads to the words $ab, ba \notin L$ depending at which position the hole is set.

Here we study the following decision problems. The problem DFA-σ-def is to decide for a given DFA A and a $\diamond$-substitution σ, whether language $L(A)$ is essentially σ-definable. The problems DFA-σ-def$_{(\exists\sigma)}$, and DFA-σ-def$_{(\forall\sigma)}$ ask whether $L(A)$ is essentially σ-definable for *some* $\diamond$-substitution σ, or, respectively, for *all* $\diamond$-substitutions σ, with $\sigma(\diamond) \neq \emptyset$.

In [8] it is shown that, given a regular language L and a $\diamond$-substitution σ, it is decidable whether L is essentially σ-definable. In particular, it is shown that the language $L(A)$ accepted by a DFA $A = (Q, \Sigma, q_0, F, \delta)$ is essentially σ-definable if and only if $L(A) = \bigcup_{q \in Q} R_q$, where

$$R_q = \{\, x \in \Sigma^* \mid \delta(q_0, x) = q \,\} \cdot \Sigma' \cdot \{\, y \in \Sigma^* \mid \delta(q', y) \in F, \text{ for all } q' \in \delta(q, \Sigma) \,\}$$

with $\Sigma' = \sigma(\diamond)$. Testing equivalence of a regular language and the union of regular languages can be very hard. The question is whether this effort is required.

Theorem 9. *The problem DFA-σ-def is* PSPACE*-hard.*

Proof. The main idea of the proof is that we give a reduction from the union universality problem for DFAs. Let the DFAs $A_1, A_2, \ldots, A_n$ over input alphabet Σ form an instance of that problem. For $1 \leq i \leq n$ we denote the ith DFA by $A_i = (Q_i, \Sigma, \delta_i, q_{i,0}, F_i)$. We construct a DFA $A = (Q, \Sigma \cup \{a, b\}, \delta, q_0, F)$, where $a, b \notin \Sigma$ are new input symbols, that uses the DFAs $A_1, A_2, \ldots, A_n$. For $1 \leq i \leq n$, automaton A accepts words from $L(A_i)$ that are preceded by the word $b^{i-1}ab^{n-i}$, and it accepts all words from Σ^* preceded by b^n. So A is defined as shown in Figure 2.

Given this automaton A and the $\diamond$-substitution $\sigma\colon \diamond \mapsto \{a, b\}$, the problem is, whether $L(A)$ essentially σ-definable. It is clear that if there exists an automaton $A_\diamond$ with $L(A) = \sigma(L(A_\diamond))$, then the holes in words from $L(A_\diamond)$ are at the first n positions, because after n symbols only symbols over Σ lead to accepting states.

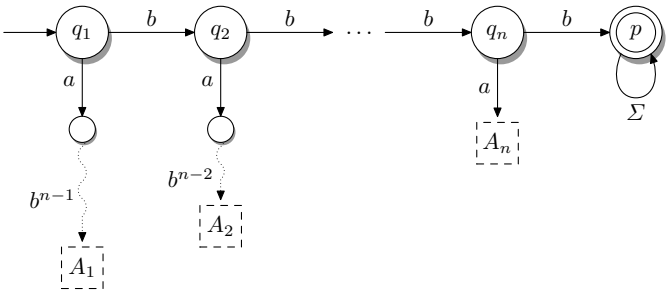

Fig. 2. The DFA A for the instance of the σ-definability problem constructed from the DFAs $A_1, A_2, \ldots, A_n$. All undefined transitions lead to a non-accepting sink state, which is not shown.

If $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$ we can define the automaton $A'_\diamond$ where each word $w \in L(A'_\diamond)$ has a hole and $L(A) = \sigma(L(A'_\diamond))$. Automaton $A'_\diamond$ consists of the automata $A_1, A_2, \ldots, A_n$. Each word from language $L(A_i)$ is preceded by the word $b^{i-1}\diamond b^{n-i}$. The automaton $A'_\diamond$ can be obtained from the automaton A from Figure 2 by removing state p and associated transitions, and by changing the a-transitions in states p_i, for $1 \leq i \leq n$, to $\diamond$-transitions.

Let us first check whether each word w of $L(A)$ is in $\sigma(L(A'_\diamond))$. If w is of the form $b^{i-1}ab^{n-i}v$, with $v \in L(A_i)$, then the word $b^{i-1}\diamond b^{n-i}v$ is in $L(A'_\diamond)$. If the hole is substituted by an a then we obtain the word w. For the words $b^n v \in L(A)$, with $v \in \Sigma^*$ it has to hold that every v is accepted by at least one of the automata $A_1, A_2, \ldots, A_n$. Assume it is accepted by automaton A_i then there is the word $b^{i-1}\diamond b^{n-i}v$ in $L(A'_\diamond)$ and the hole can be replaced by b. Now we check whether each word of $\sigma(L(A'_\diamond))$ is in $L(A)$. If we take some word $w = b^{i-1}\diamond b^{n-i}v \in L(A'_\diamond)$, with $v \in L(A_i)$, and replace the hole by b then it is the word $b^n v$ that is accepted by A by first using the path b^n and then for processing v in state p. If we replace the hole by an a, we get the word $b^{i-1}ab^{n-i}v$ that is accepted by A by first using the path $b^{i-1}ab^{n-i}$ and by processing v with the automaton A_i.

It follows that if $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$ then $L(A)$ is essentially σ-definable. It remains to show that if $\bigcup_{i=1}^{n} L(A_i) \neq \Sigma^*$ then $L(A)$ is not essentially σ-definable. Consider a word $v \in \Sigma^*$ with $v \notin L(A_i)$, for $1 \leq i \leq n$. Then it holds that $b^{i-1}ab^{n-i}v \notin L(A)$, for all $1 \leq i \leq n$, but the word $b^n v$ is in $L(A)$. Assuming $L(A)$ is essentially σ-definable then there has to be a word w with at least one hole so that $b^n v \in \sigma(w)$. The hole has to be placed at the first n positions because the substitution is over $\{a, b\}$. Consider there is a hole at position i then substituting the hole to a, the word $w' = b^{i-1}ab^{n-i}v$ is also in the language $\sigma(L(A'_\diamond))$. But w' is not in $L(A)$, so it follows that if $\bigcup_{i=1}^{n} L(A_i) \neq \Sigma^*$ then $L(A)$ is not essentially σ-definable. □

For the "for all"- and the "existential"-variants we find the following hardness result.

Theorem 10. *The problems DFA-σ-def$_{(\exists\sigma)}$ and DFA-σ-def$_{(\forall\sigma)}$ are* PSPACE*-hard.*

Proof. First we consider the DFA-σ-def$_{(\exists\sigma)}$ problem. In this proof the construction of the former proof is used. Let the DFAs $A_1, A_2, \ldots, A_n$ over the input alphabet Σ form an instance of the universality problem. Now we use a homomorphism h with $h(c) = cc$, for all $c \in \Sigma$, and obtain the languages $h(L(A_1)), h(L(A_2)), \ldots, h(L(A_n))$ that are accepted by DFAs $A'_1, A'_2, \ldots, A'_n$. Obviously $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$ if and only if $\bigcup_{i=1}^{n} L(A'_i) = h(\Sigma)^*$. Let A_0 be a DFA that accepts the regular language $L(A_0) = (\Sigma^2)^* \setminus h(\Sigma)^*$, i.e., it is the set of all words of even length from Σ^* that cannot be obtained by the homomorphism h. This language can be described by

$$L(A) = h(\Sigma)^* \cdot \{\, cc' \mid c, c' \in \Sigma,\, c \neq c' \,\} \cdot h(\Sigma)^*.$$

We construct the DFA A'' that uses the DFAs $A'_1, A'_2, \ldots, A'_n$ similar as in the last proof. A'' accepts words of A_i that are preceded by $b^{i-1}ab^{n-i}$ and words from $h(\Sigma)^*$ that are preceded by b^n.

Then we construct a new automaton A', depicted in Figure 3, so that

$$L(A') = \{\, bw \mid w \in h(L(A)) \,\} \cup \{\, aaw, abw \mid w \in L(A_0) \,\},$$

where $L(A)$ is the language accepted by the DFA A from the proof of Theorem 9.

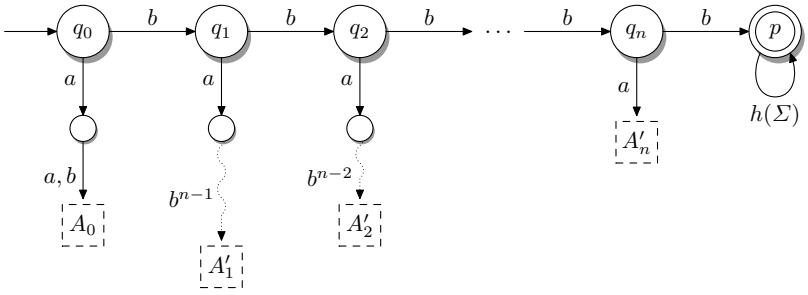

Fig. 3. The DFA A' for the language $\{\, bw \mid w \in h(L(A)) \,\} \cup \{\, aaw, abw \mid w \in L(A_0) \,\}$, where $L(A)$ is the language accepted by the DFA A from the proof of Theorem 9, and A_0 accepts the language $(\Sigma^2)^* \setminus h(\Sigma)^*$.

We show that there exists a $\diamond$-substitution σ for which $L(A')$ is essentially σ-definable if and only if $\bigcup_{i=1}^{n} L(A'_i) = h(\Sigma)^*$. In particular, in the following we show that the only possible $\diamond$-substitution is the substitution $\diamond \mapsto \{a, b\}$.

The $\diamond$-substitution can not be chosen to be a single symbol in $\Sigma \cup \{a, b\}$, because for every letter there exists a word in $L(A')$ that does not contain this letter. Clearly the $\diamond$-substitution can not be chosen by a mixture of $\{a, b\}$ and Σ, because it is required that if a word starts with an a the next symbol has to be an a or a b and then there is a suffix over Σ. If the word starts with symbol b then the next n symbols are from $\{a, b\}$ and the suffix is over Σ.

The only remaining possibilities for $\sigma(\diamond)$ are to choose $\{a, b\}$ or some subset of the form $\{c_1, c_2, \ldots, c_\ell\}$ of Σ, with $\ell \geq 2$. Assume we use the $\diamond$-substitution that maps $\diamond$ to $\{c_1, c_2, \ldots, c_\ell\}$. Because $b^{n+1}c_1c_1 \in L(A')$, the corresponding partial word language must contain the word $b^{n+1}c_1\diamond$ or the word $b^{n+1}\diamond c_1$. In

both cases, substitution $\diamond$ by the letter c_2 results in a word that does not belong to $L(A')$.

The only possibility is to choose the $\diamond$-substitution $\sigma\colon \diamond \mapsto \{a, b\}$. Here it turns out that $L(A')$ is essentially $\diamond$-definable if and only $\bigcup_{i=1}^{n} L(A'_i) = h(\Sigma)^*$. First notice that if $L(A')$ is essentially σ-defined by the partial word language $L_\diamond$, then there is no word $\diamond zw \in L_\diamond$ with $z \in \{a, b\}^*$, and $w \in (\Sigma^2)^*$: there are two possibilities for the length of z, namely either $|z| = 1$ or $|z| = n$. First, if $|z| = 1$ then a word bzw with $z \in \{a, b\}$ and $w \in (\Sigma^2)^*$ has to be in $L(A')$, but this is not the case. If $|z| = n$ then the word azw, with $z \in \{a, b\}^n$ and $w \in (\Sigma^2)^*$, has to be in the language $L(A')$, which is also not the case. So the hole symbol can only appear on positions $2, \ldots, n + 1$. Hence, the only possibility to obtain words from $a \cdot \{a, b\} \cdot L(A_0)$ is to put the $\diamond$ at the second position. This means that $L_\diamond = a\diamond \cdot L(A_0) \cup L'_\diamond$, for some appropriate language $L'_\diamond$, with $\sigma(L'_\diamond) = b \cdot h(L(A))$, where $L(A)$ is the language accepted by the DFA A from the proof of Theorem 9. By a similar argumentation as in that proof we can see that such a language $L'_\diamond$ (where every word has a hole) exists if and only $\bigcup_{i=1}^{n} L(A'_i) = h(\Sigma)^*$.

Next we consider the DFA-σ-def$_{(\forall\sigma)}$ problem. We reuse the construction from the proof of Theorem 9, but we assume the input DFAs $A_1, A_2, \ldots, A_n$ to have a binary input alphabet $\Sigma = \{c, d\}$—with this restriction, the union universality problem stays PSPACE-complete. We start with the automaton $A = (Q, \Sigma \cup \{a, b\}, \delta, q_1, F)$ pictured in Figure 2 with input alphabet $\{a, b, c, d\}$, which is constructed from the DFAs $A_1, A_2, \ldots, A_n$. Let us denote the elements of $2^\Sigma \setminus \{\emptyset, \{a, b\}\}$ by $m_1, m_2, \ldots, m_{14}$. Now we modify automaton A to obtain a new DFA $A' = (Q', \Sigma, \delta', s_1, F)$ as follows. We add the states $s_1, s_1, \ldots, s_{14}$. For $1 \leq i \leq 13$ we add transitions from s_i to s_{i+1} on all symbols from m_i. Further we add transitions from s_{14} to the initial state q_1 of the DFA A on all symbols from m_{14}.

Now we claim that $L(A')$ is essentially σ-definable for each possible $\diamond$-substitution σ over Σ if and only if $\bigcup_{i=1}^{n} L(A_i) = \{c, d\}^*$. The main idea is that for every set $m_i \neq \{a, b\}$ there is an edge in the front of the automaton A'. So for every $\diamond$-substitution $\sigma\colon \diamond \mapsto m_i$, with $1 \leq i \leq 14$, language $L(A')$ is essentially σ-definable: an appropriate partial word automaton $A'_\diamond$ can be obtained by replacing the corresponding transitions from s_i to s_{i+1} by a $\diamond$-transition. Concerning the remaining $\diamond$-substitution $\sigma\colon \diamond \mapsto \{a, b\}$, we have seen in the proof of Theorem 9, that $L(A)$ is essentially σ-definable if and only if $\bigcup_{i=1}^{n} L(A_i) = \{c, d\}^*$. This means that if this union is equal to $\{c, d\}^*$, then also the language $L(A')$ is essentially σ-definable for the substitution $\diamond \mapsto \{a, b\}$, and hence, for all appropriate $\diamond$-substitutions σ. It remains to show that this is the only way for fulfilling this property.

Let $\bigcup_{i=1}^{n} \neq \{c, d\}^*$, and assume to the contrary that there exists some automaton $A'_\diamond$ so that $L(A') = \sigma(L(A'_\diamond))$, for the $\diamond$-substitution $\sigma\colon \diamond \mapsto \{a, b\}$. Assume there is a word $u\diamond v \in L(A'_\diamond)$, with $|u| \leq 13$, i.e., where the hole is set at the position i, with $1 \leq i \leq 14$. If $\{a, b\} \not\subseteq m_i$ then $\sigma(u\diamond v)$ contains a word that does not belong to $L(A')$. So the only possibility is to set a hole at a position j where $\{a, b\} \subseteq m_j$. There are three such positions $1 \leq j_0 < j_1 < j_2 \leq 14$, which w.l.o.g.

correspond to the following subsets of Σ: $m_{j_0} = \{a, b, c\}$, $m_{j_1} = \{a, b, d\}$ and $m_{j_2} = \{a, b, c, d\}$. However, by using $\diamond$ symbols on these positions we still cannot obtain all words from $L(A')$ because words of the form $z_0cz_1dz_2cz_3w' \in \mathsf{L}(A')$, with $|z_0| = j_0 - 1$, $|z_1| = j_1 - 1$, $|z_2| = j_2 - 1$, and $w' \in L(A)$, cannot be obtained by applying the $\diamond$-substitution $\sigma\colon \diamond \mapsto \{a, b\}$. Such words can only be obtained if $\diamond$ appears only after position 14, i.e., in a position corresponding to the suffix $w' \in L(A')$. Now a similar argumentation as in the proof of Theorem 9 shows that there is a word in $L(A')$ that cannot be obtained by the $\diamond$-substitution σ. Hence $L(A')$ is not essentially σ-definable for the $\diamond$-substitution $\sigma\colon \diamond \mapsto \{a, b\}$. □

It can be easily seen that each language over the unary alphabet $\{a\}$ is essentially σ-definable if the substitution is $\{a\}$. For a fixed alphabet $|\Sigma| > 1$ the decision whether a regular language is essentially σ-definable is in PSPACE.

Theorem 11 (Essential σ-Definability). *Let Σ be some fixed alphabet with $|\Sigma| \geq 4$. The problems DFA-σ-def, DFA-σ-def$_{(\exists\sigma)}$, and DFA-σ-def$_{(\forall\sigma)}$, when restricted to DFAs with input alphabet Σ, are* PSPACE*-complete.*

Proof. Theorems 9 and 10 provide the PSPACE lower bounds. It remains to prove containment of the problems in PSPACE. We start with the DFA-σ-def problem. Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ and a $\diamond$-substitution $\sigma\colon \diamond \mapsto \Sigma'$, with $\Sigma' = \{a_1, a_2, \ldots, a_\ell\}$, the test can be done by using the sets

$$R_q = \{\, x \in \Sigma^* \mid \delta(q_0, x) = q \,\} \cdot \Sigma' \cdot \{\, y \in \Sigma^* \mid \delta(q', y) \in F \text{ for all } q' \in \delta(q, \Sigma) \,\}$$

and test whether $L(A) = \bigcup_{q \in Q} R_q$.

For every $q \in Q$, an NFA $A'_q = (Q', \Sigma, \delta', q_0, F')$ that accepts R_q can be constructed by using states $Q' = Q \cup Q^\ell$, final states $F' = F^\ell$, and the transition function δ' which is defined as follows. If $q', q'' \in Q$, and $a \in \Sigma$, such that $\delta(q', a) = q''$, then $q'' \in \delta'(q', a)$. Moreover, for symbols $a \in \Sigma'$, state q also satisfies $(q_1, q_2, \ldots, q_\ell) \in \delta'(q, a)$, where $q_i = \delta(q, a_i)$, for $1 \leq i \leq \ell$. Finally, the transitions in states $(q_1, q_2, \ldots, q_\ell) \in Q^\ell$ are such that for all $a \in \Sigma$ we have $\delta'((q_1, q_2, \ldots, q_\ell), a) = \{(\delta(q_1, a_1), \delta(q_2, a_2), \ldots, \delta(q_\ell, a_\ell)\}$.

If $|Q| = n$ then A'_q has $O(n^\ell)$ many states. There are n different sets R_q, so an NFA A' accepting $\bigcup_{q \in Q} R_q$ with $O(n^{\ell+1})$ many states can be constructed by a polynomial space bounded Turing machine. Now deciding whether $L(A)$ is essentially σ-definable boils down to testing equivalence of A and A', which can be done in PSPACE.

The problems DFA-σ-def$_{(\exists\sigma)}$ and DFA-σ-def$_{(\forall\sigma)}$ can be solved in PSPACE by iterating the above algorithm for all possible $\diamond$-substitutions over Σ. □

5 Minimization Problems

We now study minimization problems for partial word automata. Since we already know that the universality problem for such automata is PSPACE-complete, we can deduce that both the $\diamond$-DFA-to-DFA and the $\diamond$-DFA-to-$\diamond$-DFA minimization problems are PSPACE-hard. However, we can also show that the

minimization problem for $\diamond$-DFAs is PSPACE-complete, even when starting from a DFA. Besides the problem variants where the $\diamond$-substitution is given in the input, we also consider the corresponding "existential"-variants of the problems. At the end of this section we also discuss possible "for all"-variants.

We use a similar notation for our problems as in Section 3. For automata classes $\mathcal{X}$ and $\mathcal{Y}$ we consider the $\mathcal{X}$-to-$\mathcal{Y}$ and $\mathcal{X}$-to-$\mathcal{Y}_{(\exists\sigma)}$ *minimization problems*. For example, the minimization problem NFA-to-$\diamond$-DFA$_{(\exists\sigma)}$ is to decide for a given NFA A and an integer n, which is given in unary notation, whether there exist an n-state $\diamond$-DFA B and a $\diamond$-substitution σ, such that $L(A) = \sigma(L(B))$.

Let us begin with the PSPACE lower bound for the DFA-to-$\diamond$-DFA and DFA-to-$\diamond$-DFA$_{(\exists\sigma)}$ minimization problems.

Theorem 12. *The DFA-to-$\diamond$-DFA and DFA-to-$\diamond$-DFA$_{(\exists\sigma)}$ minimization problems are* PSPACE*-hard.*

Proof. We give a reduction from the union universality problem for DFAs. Let the DFAs $A_1, A_2, \ldots, A_n$ over a common input alphabet Σ form an instance of that problem. For $1 \leq i \leq n$ we denote the ith DFA by $A_i = (Q_i, \Sigma, \delta_i, q_{i,1}, F_i)$, with state set $Q_i = \{q_{i,1}, q_{i,2}, \ldots, q_{i,t_i}\}$. We may assume that all states in these automata are reachable, so that for all integers i and j, with $1 \leq i \leq n$ and $1 \leq j \leq t_i$, we can fix some word $w_{i,j} \in \Sigma^*$ such that $\delta_i(q_{i,1}, w_{i,j}) = q_{i,j}$.

Instead of directly describing the DFA for the instance of the minimization problem, we describe an equivalent NFA A, from which the DFA for the problem instance can easily be constructed. We construct the nondeterministic automaton $A = (Q, \Sigma', \delta, p_0, \{q_f\})$ shown in Figure 4. The input alphabet of A' is

$$\begin{aligned}\Sigma' = \Sigma \cup \{\, b_{i,j} \mid 1 \leq i \leq n,\, 1 \leq j \leq t_i \,\} \cup \{c, e, \#, \$\} \\ \cup \{\, d_i \mid 1 \leq i \leq n+1 \,\} \cup \{\, d'_i \mid 1 \leq i \leq n+2 \,\}.\end{aligned}$$

The transitions of the input DFAs $A_1, A_2, \ldots, A_n$ on alphabet symbols $a \in \Sigma$ are preserved in A, and the additional transitions of A can be read from Figure 4 and its caption. The only final state in A is the state q_f, in particular the final states of the input DFAs are non-final in A. The number of states in A is $k = n + 8 + \sum_{i=2}^{n+1} i + \sum_{i=1}^{n} t_i$. Further notice that for the $\diamond$-substitution $\sigma\colon \diamond \mapsto \{\#\}$ the NFA A can be transformed into a $(k+1)$-state $\diamond$-DFA $A_\diamond$ such that $\sigma(L(A_\diamond)) = L(A)$ as follows: for each state p_i, with $1 \leq i \leq n+1$ we re-label one of its two outgoing #-transitions by $\diamond$, and we add a non-accepting sink state, which is chosen as the target of all undefined transitions.

Now let us see why a DFA for $L(A)$ can easily be constructed. Let A' be the power-set automaton of A. One can see from Figure 4 that only singleton sets or the empty set can be reached in A' when reading a word that starts with a d_i symbol. The only possibility to reach a state in the automaton A', that consists of more than one state of A, is to read a word starting with #. However, all words that start with # and eventually lead to an accepting state of A' have to be of the form $\#^{n+2}w$ for some suffix w that does not contain a # symbol. The prefix $\#^{n+2}$ takes A' to state $\{s_1, s_2, \ldots, s_{n+2}\}$. From there, reading a d'_i symbol leads to the singleton $\{q_f\}$, while reading \$ leads to state

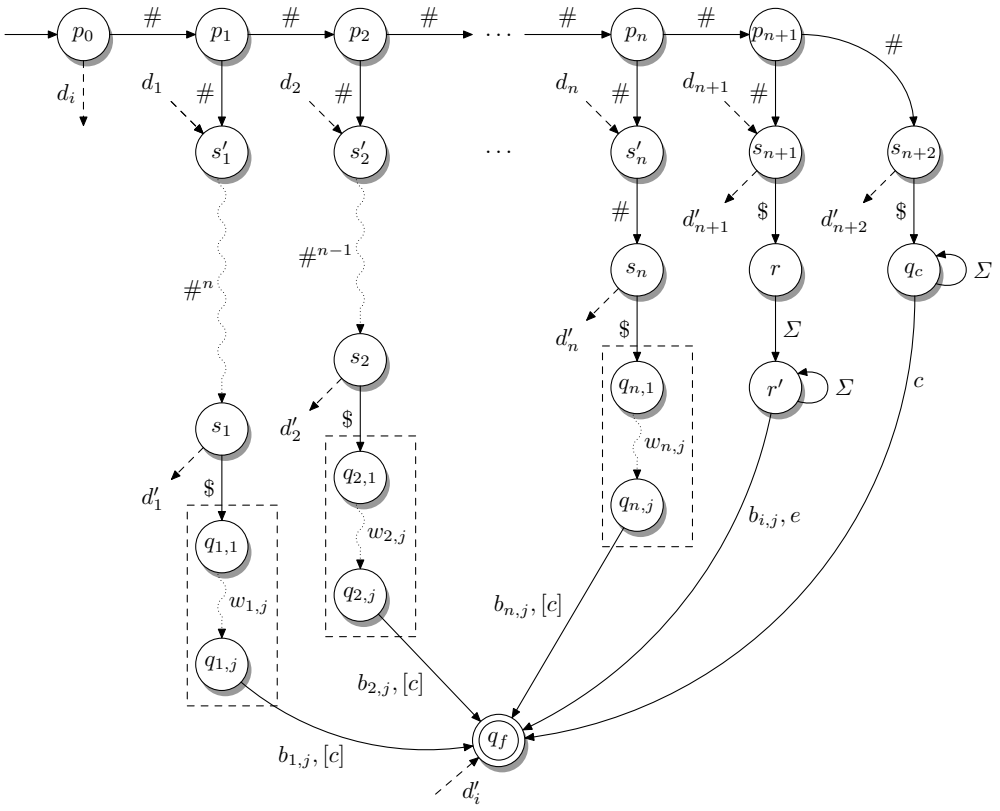

Fig. 4. The NFA A constructed from the DFAs $A_1, A_2, \ldots, A_n$. The dashed arrows denote the following transitions: for $1 \leq i \leq n+1$, the initial state p_0 goes to state s_i on reading d_i; moreover, for $1 \leq i \leq n+2$, state s_i goes to state q_f on reading d'_i. The states s'_i and s_i, for $1 \leq i \leq n$, are connected by a path of length $n+1-i$ consisting of #-transitions—this path contains $n+2-i$ states including s'_i and s_i. The dashed boxes represent the input DFAs. The bracketed $[c]$ on the transitions from states $q_{i,j}$ has the following meaning, for all i and j with $1 \leq i \leq n$ and $1 \leq j \leq t_i$: if $q_{i,j} \in F_i$ then there is a c-transition from $q_{i,j}$ to q_f, otherwise no c-transition is defined in $q_{i,j}$. Further, every state $q_{i,j}$ goes to state q_f on input $b_{i,j}$, and the special state r' has transitions on $b_{i,j}$ to state q_f for all i and j, with $1 \leq i \leq n$ and $1 \leq j \leq t_i$.

$\{q_{1,1}, q_{2,1}, \ldots q_{n,1}, r, q_c\}$. From this set we can either read c or some $b_{i,1}$ symbol to reach the singleton $\{q_f\}$, or read some input symbol $a \in \Sigma$, which leads to a set of the form $R \cup \{r', q_c\}$, where R contains only states from the input DFAs $A_1, A_2, \ldots, A_n$. From this state, reading arbitrary words over the alphabet Σ always leads to states that contain the elements r' and q_c, as well as some states of the input DFAs. Reading c or a $b_{i,j}$ symbol always leads to state $\{q_f\}$, and all other symbols lead to the empty set. Hence all reachable states of the form $R \cup \{r', q_c\}$ are pairwise equivalent—notice that all of them are non-accepting. This shows that the minimal DFA A'' for the language $L(A)$, which serves as the automaton for the instance of the DFA-to-$\diamond$-DFA$_{(\exists\sigma)}$ and DFA-to-$\diamond$-DFA the minimization problems, can be constructed from A in polynomial time. The integer for the instance of the minimization problem is $k = |Q|$, i.e., the number of states of the NFA A, and the $\diamond$-substitution for the instance of the DFA-to-$\diamond$-DFA problem is the substitution $\sigma\colon \diamond \mapsto \{\#\}$.

It remains to prove the correctness of the reduction. We first show that if $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$ then there is a k-state $\diamond$-DFA B such that $\sigma(L(B)) = L(A)$, where σ is the substitution mapping $\diamond$ to $\{\#\}$. For the converse we show that $\bigcup_{i=1}^{n} L(A_i) \neq \Sigma^*$ implies that there is *no* $\diamond$-DFA B and *no* $\diamond$-substitution σ that satisfy $\sigma(L(B)) = L(A)$.

Assume that $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$ holds. We first construct a $(k-1)$-state NFA B' by deleting state q_c together with the associated transitions. This NFA satisfies $L(B') = L(A)$ which can be seen as follows. Clearly, all words accepted by B' are also accepted by A. Assume that there is some word $w \in L(A)$ that is not accepted by B'. Then all accepting computations of A on w must lead through the deleted state q_c. Then the word w can be written as $w = \#^{n+2}\$w'c$, for $w' \in \Sigma^*$. Since $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$ we have $w' \in L(A_i)$ for some i with $1 \leq i \leq n$. Now we can see that $w = \#^{n+2}\$w'c$ is also accepted by B' because after reading $\#^{n+1}\$$, the automaton B can reach state $q_{i,1}$, i.e., the initial state of DFA A_i. On reading w' from state $q_{i,1}$ a state $q_{i,j} \in F_i$ is reached, and from this state a transition on input c leads to the accepting state q_f. This shows that it must be $L(B') = L(A)$. Now we can transform B' into a k-state $\diamond$-DFA B by re-labeling for each state p_i, with $1 \leq i \leq n+1$, one of its two outgoing $\#$-transitions by $\diamond$, and by introducing a non-accepting sink state as the target of all undefined transitions. By choosing the $\diamond$-substitution σ with $\sigma(\diamond) = \{\#\}$, this $\diamond$-DFA B satisfies $\sigma(L(B)) = L(A)$.

It remains to prove that $\bigcup_{i=1}^{n} L(A_i) \neq \Sigma^*$ implies that there is no $\diamond$-substitution σ and no k-state $\diamond$-DFA B such that $\sigma(L(B)) = L(A)$. Because $L(A)$ is non-empty and prefix-free, every such $\diamond$-DFA B must have a non-accepting sink state. Moreover, when given a k-state $\diamond$-DFA B and a $\diamond$-substitution σ such that $\sigma(L(B)) = L(A)$ then we can construct a $(k-1)$-state NFA B' by replacing the $\diamond$-transitions with transitions on the symbols from $\sigma(\diamond)$, and deleting the non-accepting sink state. Therefore, in order prove the above statement, it is sufficient to show that every NFA for the language $L(A)$ needs at least k states, because then every $\diamond$-DFA B needs at least $k+1$ states. Here we use a fooling set technique, see [3]. Let $w \in \Sigma^*$ with $w \notin \bigcup_{i=1}^{n} L(A_i)$, and choose some symbol $a \in \Sigma$. Then the following set S is fooling set for $L(A)$:

$$
\begin{aligned}
S = \{\, & (d_{n+1}\$, ae),\ (d_{n+1}\$a, e),\ (d_{n+1}\$ae, \lambda),\ (\#^{n+2}\$, wc) \,\} \\
& \cup \{\, (\#^i, \#^{n+2-i}d'_{n+2}) \mid 0 \leq i \leq n+2 \,\} \\
& \cup \{\, (d_i\#^j, \#^{n+1-i-j}d'_i) \mid 1 \leq i \leq n+1,\ 0 \leq j \leq n+1-i \,\} \\
& \cup \{\, (d_i\#^{n+1-i}\$w_{i,j}, b_{i,j}) \mid 1 \leq i \leq n,\ 1 \leq j \leq t_i \,\}
\end{aligned}
$$

Notice that $|S| = n + 7 + \sum_{i=1}^{n+1}(n+2-i) + \sum_{i=1}^{n} t_i = k$, and that for every element $(u, v) \in S$ we have $uv \in L(A)$. It remains to verify that for all $(u, v), (u', v') \in S$, with $(u, v) \neq (u', v')$, at least one of the words uv' and $u'v$ does not belong to $L(A)$. Most cases are not hard to check, and therefore are discussed in the Appendix. Here we only deal with the case where $(u, v) = (\#^{n+2}\$, wc)$ and $(u', v') = (d_i\#^{n+1-i}\$w_{i,j}, b_{i,j})$, for $1 \leq i \leq n$ and $1 \leq j \leq t_i$. Assume that $uv' = \#^{n+2}\$b_{i,j} \in L(A)$—otherwise we are done. Because the prefix $\#^{n+2}\$$ leads to the states $\{q_{1,1}, q_{2,1}, \ldots, q_{n,1}, r, q_c\}$, and no

$b_{i,j}$-transitions are defined in the states r and c, we may conclude that $j = 1$, i.e., that $b_{i,j} = b_{i,1}$ belongs the initial $q_{i,1}$ state of the DFA A_i. Now we can see that the word $u'v$ is not accepted by A: reading the prefix $u' = d_i\#^{n+1-i}\$w_{i,1}$ leads to state $q_{i,1}$. From there the word w leads to some state $q_{i,j'}$ of A_i, with $q_{i,j'} \notin F_i$, because we know that $w \notin L(A_i)$. Hence no c-transition is defined in that state, so the word $u'v = d_i\#^{n+1-i}\$w_{i,1}wc$ does not belong to $L(A)$. This concludes our proof. □

With small modifications, the proof of Theorem 12 also shows PSPACE-hardness of the corresponding minimization problems, where the input automaton is a $\diamond$-DFA, $\diamond$-NFA, or a classical NFA, and the target automaton may as well be an NFA, or $\diamond$-NFA. Moreover, the fact that the universality problem for partial word automata is PSPACE-complete implies PSPACE-hardness of the minimization problems where the input is a partial word automaton, and the target automaton is a DFA. Altogether, we know that the minimization problems $\mathcal{X}$-to-$\mathcal{Y}$ and $\mathcal{X}$-to-$\mathcal{Y}_{(\exists\sigma)}$ are PSPACE-hard if the input, or the target automaton is a partial word automaton.

Our main result of this section reads as follows.

Theorem 13 (Minimization). *The $\mathcal{X}$-to-$\mathcal{Y}$, and $\mathcal{X}$-to-$\mathcal{Y}_{(\exists\sigma)}$ minimization problems are* PSPACE*-complete, if $\mathcal{X}, \mathcal{Y} \in \{\diamond\text{-DFA}, \diamond\text{-NFA}, \text{DFA}, \text{NFA}\}$, with $\{\mathcal{X}, \mathcal{Y}\} \cap \{\diamond\text{-DFA}, \diamond\text{-NFA}\} \neq \emptyset$.*

Proof. It remains to prove containment in PSPACE, since PSPACE-hardness is already shown in Theorem 12. When given an instance of one of the problems, a nondeterministic polynomial space bounded Turing machine may guess a k-state automaton B of the appropriate type, and if necessary, also a $\diamond$-substitution σ, and write these on its working tape—because the integer k is given in unary notation, we have enough space. Then the machine simply has to verify equivalence (w.r.t. the $\diamond$-substitution) of the input automaton and automaton B, which can be done in PSPACE by Theorem 5. Since NPSPACE = PSPACE, this proves containment of our minimization problems in PSPACE. □

In the remainder of this section we consider two different "for all"-variants of the minimization problem for partial word automata. The input to both problems is an automaton A and an integer k. The question for the first problem is, whether *for all* appropriate $\diamond$-substitutions σ *there exists* some k-state automaton B that is equivalent to A (w.r.t. σ). In the second question the order of the quantifiers is reversed: decide, whether *there exists* some k-state automaton B that is equivalent to A (w.r.t. σ), *for all* appropriate $\diamond$-substitutions σ. The following theorem shows PSPACE-completeness for the problems where the input and target automata are partial word automata.

Theorem 14. *Given a $\diamond$-DFA A with input alphabet $\Sigma_\diamond$, and an integer k, the following problems are* PSPACE*-complete.*

1. *Does for all $\diamond$-substitutions σ over Σ, with $\sigma(\diamond) \neq \emptyset$, exist a k-state $\diamond$-DFA B, such that $\sigma(L(A)) = \sigma(L(B))$?*

2. *Does there exist a k-state $\diamond$-DFA B, such that for all $\diamond$-substitutions σ over Σ with $\sigma(\diamond) \neq \emptyset$, we have $\sigma(L(A)) = \sigma(L(B))$?*

This statement remains true, if $\diamond$-NFAs are used instead of $\diamond$-DFAs.

Proof. For PSPACE-hardness we give use nearly the same reduction as in the proof for Theorem 4. There, the $\diamond$-DFA A, with input alphabet $\Sigma_\diamond = \{a, b, \diamond\}$, depicted in Figure 1 is constructed from the DFAs $A_1, A_2, \ldots, A_n$, which form an instance of the union universality problem. We use this automaton A and the integer $k = 1$ as input to our minimization problems. We have seen in the proof of Theorem 4 that all non-empty $\diamond$-substitutions σ over $\Sigma = \{a, b\}$ satisfy $\sigma(L(A)) = \Sigma^*$ if and only if $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$. Hence, if $\bigcup_{i=1}^{n} L(A_i) = \Sigma^*$ then the single-state $\diamond$-DFA B for the language $\Sigma_\diamond^*$ satisfies $\sigma(L(A)) = \sigma(L(B))$ for all appropriate $\diamond$-substitutions σ. On the other hand, if there is a word $w \in \Sigma^*$ with $w \notin \bigcup_{i=1}^{n} L(A_i)$ then we have seen that the word $(ab)^{n+1}w$ does not belong to $\sigma(L(A))$, no matter which σ is chosen. However, one can see from Figure 1 that the words a and ab always belong to the language $\sigma(L(A))$. Hence no single-state NFA B' can accept the language $\sigma(L(A))$, and therefore *no* single-state $\diamond$-DFA B can satisfy $\sigma(L(A)) = \sigma(L(B))$, no matter which $\diamond$-substitution σ is chosen.

It remains to prove containment in PSPACE. Let A and k be given. To solve the first question, we consider each non-empty $\diamond$-substitution σ over Σ, one after another, and each time use the PSPACE algorithm for the $\diamond$-DFA-to-$\diamond$-DFA minimization problem from Theorem 13 to decide whether an appropriate k-state $\diamond$-DFA B with $\sigma(L(A)) = \Sigma(L(B))$ exists. To solve the second question, we guess a k-state $\diamond$-DFA B and use the PSPACE algorithm from Theorem 5 for the $\diamond$-DFA-eq-$\diamond$-DFA$_{(\forall\sigma)}$ problem to decide whether $\sigma(L(A)) = \Sigma(L(B))$ holds.

One can check that the reduction for the lower bound, and the algorithm for the upper bound also applies to $\diamond$-NFAs instead of $\diamond$-DFAs. □

A similar PSPACE-completeness result as in Theorem 14 can also be obtained if the input or the target automaton is a classical NFA instead of a partial word automaton: for proving PSPACE-hardness, the automaton A from Figure 1 can be transformed into an NFA A' by replacing the $\diamond$-transitions by transitions on symbol a, and the PSPACE upper bound can be shown by a similar algorithm as in the proof of Theorem 14. The PSPACE upper bound of course also holds, if the input automaton is a DFA. However the proof of PSPACE-hardness (or the proof of an upper bound below PSPACE) for the corresponding minimization problems is left for further research.

6 Problems on Finite Languages

In this section we consider the complexity of partial word automata problems restricted to finite languages. Compared to our previous investigations here the complexity of most of these problems drops significantly from PSPACE-completeness down to coNP-completeness or -hardness and containment in Σ_2^{P},

depending on the problem under consideration. This complexity drop nicely fits to the known results on ordinary finite automata accepting finite languages.

We start with the emptiness problem for partial word automata accepting finite languages. Here the situation remains the same as in the general case. The proof is identical to the proof of Theorem 1. For $\mathcal{X} \in \{\text{DFA}, \text{NFA}, \diamond\text{-DFA}, \diamond\text{-NFA}\}$ let $\mathcal{X}_{fin}$ refer to the class of automata that accept only finite languages.

Theorem 15 (Emptiness). *For $\mathcal{X} \in \{\diamond\text{-}DFA_{fin}, \diamond\text{-}NFA_{fin}\}$ the emptiness problems $\mathcal{X}$-eq-$\emptyset$, $\mathcal{X}$-eq-$\emptyset_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\emptyset_{(\forall\sigma)}$ are* NL*-complete.* □

Now let us come to the equivalent of the universality problem, the bounded-universality problem, which asks for equivalence to the language $\Sigma^{\leq\ell}$. As the reader may have noticed, most hardness proofs in the previous sections rely on the union universality problem for DFAs. For finite languages we first show that the union universality problem adapted to finite languages is coNP-complete. Thus, we define the *union bounded-universality problem* as follows: given finite automata $A_1, A_2, \ldots A_n$ with input alphabet Σ and an integer ℓ coded in unary, decide whether $\bigcup_{i=1}^{n} L(A_i) = \Sigma^{\leq\ell}$, where $\Sigma^{\leq\ell}$ is a short-hand notation for the set $\{\, w \in \Sigma^* \mid |w| \leq \ell \,\}$. For this problem we have the following result:

Theorem 16. *The union bounded-universality problem for NFAs and DFAs is* coNP*-complete, even for automata with a binary input alphabet.*

Proof. Let $A_1, A_2, \ldots, A_n$ be the finite automata with input alphabet Σ and ℓ the bound for the bounded-universality problem. The coNP upper bound on the union bounded-universality problem is easily seen as follows. First construct an NFA B such that $L(B) = \bigcup_{i=1}^{n} L(A_i)$—this can be done in polynomial time. Then by the standard intersection construction for NFAs we build an NFA C with $L(C) = L(B) \cap \Sigma^{\leq\ell}$. Finally, one checks equivalence of $L(C)$ with $\Sigma^{\leq\ell}$, which can be done by an oracle for equivalence for finite languages. Since at least one automaton involved is an NFA, this problem is known to be coNP-complete [15]. Thus, the union bounded-universality problem is contained within coNP.

For the hardness, we alter the construction of [15] on the NP-complete inequivalence problem for regular expressions over a binary alphabet with the operations union and concatenation. There, it is shown how to construct a regular expression r over the alphabet Σ and an integer ℓ for a polynomial time bounded Turing machine M on input x such that $x \in L(M)$ if and only if $L(r) \neq \Sigma^{\leq\ell}$. Let $p(n)$ be the running time of M. In fact, the regular expression encodes the invalid computations of the Turing machine M on input x. The set of valid computations of M on x consists of words of the form $\#ID_1\#ID_2\#\ldots\#ID_{p(n)}\#$, where ID_1 is the initial configuration of M on x, $ID_{p(n)}$ the accepting configuration, and ID_{j+1} is a successor of ID_j, for $1 \leq j < p(n) - 1$. Note that there is no need that any configuration is longer than $p(n)$. To construct the regular expression one has to describe the words that fail to be a valid computation. There are several cases that itself split into several subcases: (i) the word "starts wrong," (ii) it "ends wrong," (iii) it is "not conform" with the movement of the Turing machine, or (iv) is "not of the right shape," or (iv) is

"to short." All these words can be described by polynomial size DFAs instead of regular expressions—for more details on the construction we refer to [15]. The tedious details in the construction of these finite automata is left to the interested reader. Let $A_1, A_2, \ldots A_n$ be these automata with input alphabet Σ and $\ell = (p(n)+1) \cdot p(n) + 1$. Then $x \in L(M)$ if and only if $\bigcup_{i=1}^{n} L(A_i) \neq \Sigma^{\leq \ell}$. Thus, we have reduced coNP to the union bounded-universality problem for DFAs. This proves the stated result. □

Now we are ready to study bounded-universality and equivalence problems for partial word automata accepting finite languages.

Theorem 17 (Bounded-Universality). *For $\mathcal{X} \in \{\diamond\text{-}DFA_{fin}, \diamond\text{-}NFA_{fin}\}$ the universality problems $\mathcal{X}$-eq-$\Sigma^{\leq \ell}$ and $\mathcal{X}$-eq-$\Sigma^{\leq \ell}{}_{(\forall\sigma)}$ are* coNP*-complete, and the problem $\mathcal{X}$-eq-$\Sigma^{\leq \ell}{}_{(\exists\sigma)}$ is* coNP*-hard and contained in Σ_2^{P}.*

Proof. For the coNP-hardness we utilize the proof of Theorem 3. The presented reduction has to be slightly modified. First, instead of the union universality problem for DFAs we use the union bounded-universality problem for DFAs. Let n be the number of DFAs and ℓ the length bound for the union bounded-universality problem. Then in the construction of a $\diamond$-DFA, as depicted in Figure 1, the accepting state q_{acc} has to be replaced by a chain of $2n + 2 + \ell$ accepting states that have to be appropriately connected in a level like fashion to the other states in the construction. The remaining proof runs along similar lines and shows coNP-hardness for the mentioned problem instances.

Next, containment of $\mathcal{X}$-eq-$\Sigma^{\leq \ell}$, for $\mathcal{X} \in \{\diamond\text{-DFA}_{fin}, \diamond\text{-NFA}_{fin}\}$, within coNP is seen as follows: let an automaton $A_\diamond$ and a σ-substitution σ be given. Then one constructs an NFA A from $A_\diamond$ by replacing all $\diamond$-transitions by transitions carrying all letters from $\sigma(\diamond)$. Thus, $L(A) = \sigma(L(A_\diamond))$. This can be done in deterministic polynomial time. Then it remains to check equivalence between $L(A)$ and $\Sigma^{\leq \ell}$, which can be done in coNP, because the involved automata accept finite languages only. Thus, the overall algorithm can be implemented on a polynomial time bounded Turing machine, which works in universal mode. Therefore the problem under consideration belongs to coNP.

For $\mathcal{X}$-eq-$\Sigma^{\leq \ell}{}_{(\forall\sigma)}$ and $\mathcal{X}$-eq-$\Sigma^{\leq \ell}{}_{(\exists\sigma)}$, with $\mathcal{X} \in \{\diamond\text{-DFA}_{fin}, \diamond\text{-NFA}_{fin}\}$, the $\diamond$-substitution is not part of the input. Thus, in the former problem, one can universally guess it, while in the latter problem one can existentially guess the $\diamond$-substitution σ, and apply the just explained coNP algorithm to the automaton and the universally or existentially guessed σ. Thus, in the former problem the upper bound becomes coNP, while in the latter problem a Σ_2^{P} upper bound emerges. □

Our results on the different equivalence problems are as follows:

Theorem 18 (Equivalence). *For $\mathcal{X} \in \{\diamond\text{-}DFA_{fin}, \diamond\text{-}NFA_{fin}\}$ and for each $\mathcal{Y} \in \{\diamond\text{-}DFA_{fin}, \diamond\text{-}NFA_{fin}, DFA_{fin}, NFA_{fin}\}$, the equivalence problems $\mathcal{X}$-eq-$\mathcal{Y}$ and $\mathcal{X}$-eq-$\mathcal{Y}_{(\forall\sigma)}$ are* coNP*-complete, and the problem $\mathcal{X}$-eq-$\mathcal{Y}_{(\exists\sigma)}$ is* coNP*-hard and contained in Σ_2^p.*

Proof. For proving coNP-hardness, we reduce the bounded-universality problems from Theorem 17 to the equivalence problems of the present theorem by choosing the automaton B as the automaton accepting $\Sigma^{\leq \ell}$ (or $\Sigma_{\diamond}^{\leq \ell}$, respectively). Containment in coNP and Σ_2^{P}, depending on the problem variant one is interested in, follows with similar arguments as in the proof Theorem 17. □

Next, we consider the computational complexity of the essential σ-definability problem for finite languages, when represented as a DFA.

Theorem 19 (Essential σ-Definability). *Let Σ be some fixed alphabet with $|\Sigma| \geq 4$. The problems DFA$_{fin}$-σ-def, and DFA$_{fin}$-σ-def$_{(\forall\sigma)}$, for DFAs with input alphabet Σ, are* coNP*-complete, and the problem DFA$_{fin}$-σ-def$_{(\exists\sigma)}$, for DFAs with input alphabet Σ, is* coNP*-hard and contained in Σ_2^{P}.*

Proof. For the coNP lower bound we slightly modify the constructions in the proofs to Theorems 9, 10, and 10. Instead of the union universality problem for DFAs we us its counterpart for finite languages, namely the union bounded-universality problem for DFAs (over a binary input alphabet). Then in the constructions of the DFAs, as depicted in the Figures 2 and 3 the accepting state p has to be replaced appropriately. Let ℓ be the length of the bounded-universality problem. In the former construction the state q is replaced by a chain of $\ell+1$ accepting states. In this way the set $\Sigma^{\leq \ell}$ is accepted, when starting from the replaced state q. In the latter construction, the state q is replaced by a subautomaton with $\ell \cdot (|\Sigma|+1)+1$ states that accepts the set $(h(\Sigma))^{\leq \ell}$. Moreover, the automaton A_0 has to be replaced too. Instead of accepting the infinite language $(\Sigma^2)^* \setminus h(\Sigma)^*$ it now must accept the finite language $(\Sigma^2)^{\leq \ell} \setminus (h(\Sigma))^{\leq \ell}$. This automaton is of size at most $\ell \cdot (|\Sigma| + 1) + 2\ell + 1$. Then one can easily see that the whole argumentation presented in these proofs also applies to the bounded-universality case.

For the upper bound we argue as in the proof of Theorem 11. There it is shown that for the DFA-σ-def problem with given DFA A boils down to test equivalence between A and a series of n NFAs of polynomial size that can be effectively constructed in deterministic polynomial time. Since both A and the n involved NFAs accept finite languages only, this task can be done in coNP. A similar argumentation applies to the corresponding "for all"-variant, since one can first universally guess a σ-substitution and then apply the just described algorithm for the DFA-σ-def problem in parallel. Thus, also DFA$_{fin}$-σ-def$_{(\forall\sigma)}$ belongs to coNP. Finally, the upper bound raises to Σ_2^{P} for the DFA$_{fin}$-σ-def$_{(\exists\sigma)}$ problem, since the guessing of the σ-substitution has to be done existentially. □

Finally, we investigate the minimization problem, when finite languages are involved.

Theorem 20 (Minimization). *The problems $\mathcal{X}$-to-$\mathcal{Y}$ and $\mathcal{X}$-to-$\mathcal{Y}_{(\exists\sigma)}$ are hard for* coNP *and contained in Σ_2^{P}, if $\{\mathcal{X}, \mathcal{Y}\} \cap \{\diamond\text{-DFA}_{fin}, \diamond\text{-NFA}_{fin}\} \neq \emptyset$, and $\mathcal{X}, \mathcal{Y} \in \{\diamond\text{-DFA}_{fin}, \diamond\text{-NFA}_{fin}, \text{DFA}_{fin}, \text{NFA}_{fin}\}$*

Proof. Again, we reuse some of the proofs from the general case. We alter the construction presented in the proof of Theorem 12 to show coNP-hardness for the minimization problems DFA_{fin}-to-$\diamond$-DFA_{fin} and DFA_{fin}-to-$\diamond$-$\mathrm{DFA}_{fin(\exists\sigma)}$. Our reduction starts from the union bounded-universality problem for DFAs. Let ℓ be the length bound for the bounded-universality problem. We construct an automaton according to the description given in the proof of Theorem 12. The state r' has to be replaced by a chain of ℓ non-accepting states, while state and q_c is replaced by $\ell+1$ non-accepting states—cf. Figure 4. All these states in the chain have to be appropriately connected. Then the argumentation is given in the proof of Theorem 12 applies in the same way. This proves the coNP lower bound.

For the containment in Σ_2^{P} we argue as in the proof of Theorem 13. When given an instance of one of the problems, a Turing machine may guess existentially a k-state automaton B of the appropriate type, and if necessary, also a $\diamond$-substitution σ, and write these on its working tape—because the integer k is given in unary notation, we have enough time to do that. Then the machine simply has to verify equivalence (w.r.t. the $\diamond$-substitution) of the input automaton and automaton B—both accept finite languages, so this can be done in coNP. Overall, this results in an Σ_2^{P} algorithm. □

In the remainder of this section we briefly recall what is known for the "for all"-variants of the minimization problem for automata accepting finite languages. By the definition of these problem variants it is easily seen, that the first one from Theorem 14 belongs to Σ_3^{P}, while the second variant from Theorem 14 is contained in Σ_2^{P}. We have to leave open the lower bounds for these problems.

References

1. J. Amilhastre, Ph. Janssen, and M.-C. Vilarem. FA minimisation heuristics for a class of finite languages. In O. Boldt and H. Jürgensen, editors, *Proceedings of the 4th International Workshop on Implementing Automata*, number 2214 in LNCS, pages 1–12, Potsdam, Germany, 2001. Springer.
2. E. Balkanski, F. Blanchet-Sadri, M. Kilgore, and B. J. Wyatt. Partial word DFAs. In S. Konstantinidis, editor, *Proceedings of the 18th International Conference on Implementation and Application of Automata*, number 7982 in LNCS, pages 36–47, Halifax, Nova Scotia, Canada, 2013. Springer.
3. J.-C. Birget. Intersection and union of regular languages and state complexity. *Inform. Process. Lett.*, 43:185–190, 1992.
4. H. Björklund and W. Martens. The tractability frontier for NFA minimization. *J. Comput. System Sci.*, 78(1):198–210, 2012.
5. F. Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words*. Discrete Mathematics and Its Applications. Chapman and Hall/CRC, 2007.
6. F. Blanchet-Sadri, K. Goldner, and A. Shackleton. Minimal partial languages and automata. In M. Holzer and M. Kutrib, editors, *Proceedings of the 19th International Conference on Implementation and Application of Automata*, LNCS, Giessen, Germany, July–August 2014. Springer. To appear.
7. S. Cho and D. T. Huynh. The parallel complexity of finite-state automata problems. *Inform. Comput.*, 97:1–22, 1992.
8. J. Dassow, F. Manea, and R. Mercas. Regular languages of partial words. *Information Sciences*, 268:290–304, 2014.

9. M. J. Fischer and M. S. Paterson. String-matching and other products. In R. M. Karp, editor, *Complexity of Computation*, volume 7, pages 113–126. American Mathematical Society, 1974.
10. H. Gruber and M. Holzer. Computational complexity of NFA minimization for finite and unary languages. In *Preproceedings of the* 1*st International Conference on Language and Automata Theory and Applications*, Technical Report 35/07, pages 261–272, Tarragona, Spain, 2007. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili.
11. T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.
12. N. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. System Sci.*, 11:68–85, 1975.
13. D. Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 254–266, 1977.
14. A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the* 13*th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE Society Press, 1972.
15. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th Symposium on Theory of Computing*, pages 1–9, 1973.

Appendix

Proof (of Theorem 12, continued). Here we complete the proof that the set

$$\begin{aligned} S = \{\, &(d_{n+1}\$, ae),\ (d_{n+1}\$a, e),\ (d_{n+1}\$ae, \lambda),\ (\#^{n+2}\$, wc) \,\} \\ &\cup \{\, (\#^i, \#^{n+2-i} d'_{n+2}) \mid 0 \le i \le n+2 \,\} \\ &\cup \{\, (d_i \#^j, \#^{n+1-i-j} d'_i) \mid 1 \le i \le n+1,\ 0 \le j \le n+1-i \,\} \\ &\cup \{\, (d_i \#^{n+1-i} \$ w_{i,j}, b_{i,j}) \mid 1 \le i \le n,\ 1 \le j \le t_i \,\} \end{aligned}$$

is a fooling set for the language $L(A)$, which is used in the proof of Theorem 12 Let (u, v) and (u', v') be two distinct elements from S. We have to show that $uv' \notin L$ or $u'v \notin L$. We distinguish between the following cases for (u, v):

Let $(u, v) = (d_{n+1}\$, ae)$. Then we have $uv' \notin L(A)$, which can be seen as follows. Reading $u = d_{n+1}\$$ takes A to state r. From there, only words from the set $\Sigma^+ \cdot \{\, e, b_{i,j} \mid 1 \le i \le n,\ 1 \le j \le t_i \,\}$ are accepted. But no possible suffix v' is of that form.

Next let $(u, v) = (d_{n+1}\$a, e)$. We do not have to consider the case $(u', v') = (d_{n+1}\$, ae)$ because it is symmetrical to a case discussed before. In the following we omit such symmetrical cases without explicitly stating. If $(u', v') = (d_i \#^{n+1-i} \$ w_{i,j}, b_{i,j})$ for appropriate i and j then the word $u'v$ is not accepted by A because reading u' leads to a state of the input DFA A_i, and this state has no outgoing e-transition. If $(u', v') \neq (d_i \#^{n+1-i} \$ w_{i,j}, b_{i,j})$, then one can see that $uv' \notin L(A)$ because after reading $d_{n+1}\$a$ the only words ending with e or some $b_{i,j}$ symbol are accepted.

The case $(u, v) = (d_{n+1}\$ae, \lambda)$ is easy. We have $uv' \notin L(A)$ because u belongs to $L(A)$, the language $L(A)$ is prefix-free, and all possible suffixes v' are non-empty.

For $(u, v) = (\#^{n+2}\$, wc)$ we have already discussed the case where (u', v') is of the form $(d_i \#^{n+1-i} \$ w_{i,j}, b_{i,j})$. In the remaining cases the second component v' ends on a symbol d'_i, with $1 \le i \le n+2$. Such symbols may not appear after reading the \$ symbol. Hence the words uv' do not belong to $L(A)$.

Let $(u, v) = (\#^i, \#^{n+2-i} d'_{n+2})$. If $(u', v') = (\#^{i'}, \#^{n+2-i'} d'_{n+2})$, with $i \neq i'$, then the word uv' is of the form $uv' = \#^m d_{n+2}$, with $m \neq n+2$, and so does not belong to $L(A)$. If $(u', v') \neq (\#^{i'}, \#^{n+2-i'} d'_{n+2})$ then it remains to consider the cases where u' starts with some $d_{i'}$ symbol, with $1 \le i' \le n+2$. Then we have $u'v \notin L(A)$ because uv it starts with $d_{i'}$ and ends with d'_{n+2}.

Now let $(u, v) = (d_i \#^j, \#^{n+1-i-j} d'_i)$, and $(u', v') = (d_{i'} \#^{j'}, \#^{n+1-i'-j'} d'_{i'})$. If $i \neq i'$, then $uv' \notin L(A)$ because uv' begins with d_i and ends with a non-matching $d'_{i'}$. If $i = i'$, then it is $j \neq j'$. In this case the word uv' does not have the correct number of # symbols. If we have $(u', v') = (d_{i'} \#^{n+1-i'} \$ w_{i',j'}, b_{i',j'})$ then the word $uv' = d_i \#^j b_{i',j'}$ does not belong to $L(A)$ because a $b_{i,j}$ symbol may only appear some time after a \$ symbol, which is missing in uv'.

Finally it remains to consider the case where $(u, v) = (d_i \#^{n+1-i} \$ w_{i,j}, b_{i,j})$ and $(u', v') = d_{i'} \#^{n+1-i'} \$ w_{i',j'}, b_{i',j'})$, where $(i, j) \neq (i', j')$. Here the word uv' does not belong to $L(A)$ because after reading the words $u = d_i \$ w_{i,j}$ the NFA A is in state $q_{i,j}$, and in this state it cannot make a transition on $v' = b_{i',j'}$. This concludes the proof. □

Institut für Informatik

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

Recent Reports

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

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