WORD PROBLEMS AND MEMBERSHIP PROBLEMS ON COMPRESSED WORDS

MARKUS LOHREY*

Abstract. We consider a compressed form of the word problem for finitely presented monoids, where the input consists of two compressed representations of words over the generators of a monoid $\mathcal{M}$, and we ask whether these two words represent the same monoid element of $\mathcal{M}$. Words are compressed using straight-line programs, i.e., context-free grammars that generate exactly one word. For several classes of finitely presented monoids we obtain completeness results for complexity classes in the range from P to EXPSPACE. As a by-product of our results on compressed word problems we obtain a fixed deterministic context-free language with a PSPACE-complete compressed membership problem. The existence of such a language was open so far. Finally, we will investigate the complexity of the compressed membership problem for various circuit complexity classes.

Key words. grammar-based compression, word problems for monoids, context-free languages, complexity

AMS subject classifications. 20F10, 68Q17, 68Q42

1. Introduction. During the last decade, the massive increase in the volume of data has motivated the investigation of algorithms on *compressed data*, like for instance compressed strings, trees, or pictures. The general goal is to develop algorithms that directly work on compressed data without prior decompression. Let us mention here the work on compressed pattern matching, see, e.g., [19, 23, 49, 60].

In this paper we investigate two classes of computational problems on compressed data that are of central importance in theoretical computer science since its very beginning: the *word problem* and the *membership problem*.

In its most general form, the word problem asks whether two terms over an algebraic structure represent the same element of the structure. Here, we restrict to the word problem for *finitely presented monoids*, i.e., monoids that are given by a finite set of generators and defining relations. In this case the input consists of two finite words over the set of generators and we ask whether these two words represent the same monoid element. The undecidability results concerning the word problem for finitely presented monoids [47, 56] and finitely presented groups [12, 51] are among the first undecidability results that touched "real mathematics". Moreover, these negative results motivated a still ongoing investigation of decidable subclasses of word problems and their computational complexity. In particular, monoids that can be presented by *terminating and confluent semi-Thue systems* (i.e., string rewriting systems where every word can be rewritten in a finite number of steps to a unique irreducible word), see [11, 33], received a lot of attention: these monoids have decidable word problems, and if the restriction to terminating systems is suitably sharpened, then precise complexity bounds can be deduced [10, 41, 42]. All relevant definitions concerning semi-Thue systems and finitely presented monoids are collected in Section 3.3.

In its compressed variant, the input to the word problem for a (finitely presented) monoid consists of two compressed representations of words over the generators. Here we choose *straight-line programs*, or equivalently context-free grammars that generate exactly one word, for compression — this approach is also known as *grammar-based compression*. See Section 3.4 for a formal definition of straight-line

*Institut für Formale Methoden der Informatik, Universität Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany (lohrey@informatik.uni-stuttgart.de).

programs. Recently, straight-line programs turned out to be a very flexible compressed representation of strings. Several other compressed representations, like for instance Lempel-Ziv factorizations [73], can be efficiently converted into straight-line programs and vice versa [55], which implies that most of our complexity results will also hold for Lempel-Ziv factorizations. An algorithmic application of this efficient transformation to and into straight-line programs is given in [23], where the pattern matching problem for Lempel-Ziv compressed texts is solved efficiently via reduction to straight-line programs. Finally, by using straight-line programs for representing inputs, the compressed word problem becomes equivalent to the well-known circuit equivalence problem (a generalization of the circuit evaluation problem), where we ask whether two circuits over a finitely presented monoid $\mathcal{M}$ (i.e., acyclic directed graphs with leafs labeled by generators of $\mathcal{M}$ and internal nodes labeled by the monoid operation) evaluate to the same element of $\mathcal{M}$. This problem was mainly investigated for finite structures (e.g., finite monoids [7]) and integer circuits [48] so far. From this perspective, the compressed word problem highlights the classical circuit versus formula evaluation problem in the context of finitely presented monoids.

In Section 4–7 we study the complexity of compressed word problems for several subclasses of monoids presented by terminating and confluent semi-Thue systems. For this we will distinguish semi-Thue systems with respect to the syntactical form of the rewriting rules. In Section 4 and 5 we will consider confluent and *2-homogeneous* semi-Thue systems, which are confluent systems where all rules are of the form $w \to \varepsilon$ with w of length 2. For confluent and 2-homogeneous systems that satisfy a further syntactical restriction (which we call N-freeness) we prove that the presented monoid has a polynomial time solvable compressed word problem (Theorem 4.5). For all other confluent and 2-homogeneous systems, the compressed word problem becomes coNP-hard (Corollary 5.9) and is contained in P^{NP} (Theorem 5.1). In Section 6 we show that the complexity of the compressed word problem increases to PSPACE-completeness if we allow also rules of the form $u \to v$, where again u has length 2 but v may have length 0 or 1 (Theorem 6.6) — the resulting semi-Thue systems are called *2-monadic*. The largest class of semi-Thue systems that is considered in this paper consists of confluent and *weight-reducing* systems. It is briefly studied in Section 7, where it is shown that the compressed word problem for a monoid which is presented by a confluent and weight-reducing semi-Thue system is in EXPSPACE and is EXPSPACE-hard for some specific system (Theorem 7.1).

As a by-product of our investigation of compressed word problems we obtain several new results concerning compressed membership problems. Here, the problem is to decide for a fixed language L (e.g., a regular or context-free language), whether a given straight-line compressed word w belongs to L [55]. Using Theorem 6.6 concerning 2-monadic semi-Thue systems, we show that there exists a fixed deterministic context-free (even deterministic linear) language with a PSPACE-complete compressed membership problem (Corollary 6.7), which solves an open problem from [23, 55]. Corollary 6.7 should be compared with a result from [24], stating that given a straight-line compressed word w and a nondeterministic hierarchical automaton A (see [24] for a precise definition) it is PSPACE-complete to decide whether $w \in L(A)$. It is straight-forward to transform a hierarchical automaton in logspace into an equivalent nondeterministic push-down automaton of the same size. Corollary 6.7 improves the result of [24] in two aspects: (i) the context-free language in Corollary 6.7 is deterministic and (ii) it is fixed, i.e., it does not belong to the input. Another result related to Corollary 6.7 was recently shown in [50]: it is PSPACE-complete to decide for two given *non-recursive* context-free grammars G_1 and G_2 whether $L(G_1) \cap L(G_2) \neq \emptyset$. Non-recursive context-free grammars generate finite languages, in particular, a straight-line program is a non-recursive context-free grammar. Thus, in comparison to the result of [50], we sharpen the condition on the grammar G_1 (it has to be a straight-line program) but relax the condition on G_2 (it generates an infinite language). One should also note that for the result of [50] it is crucial that both G_1 and G_2 are part of the input.

Compressed membership problems for context-free languages are also interesting in light of recent attempts to use straight-line programs for the compressed representation of control flow traces of procedural programming languages [36, 72]. At a certain level of abstraction, the set of all valid control flow traces of a procedural programming language is a context-free language.

In Section 8 we will investigate the complexity of the compressed membership problem for various circuit complexity classes. We show that the levels of the logtime hierarchy [63] correspond in a compressed setting to the levels of the polynomial time hierarchy. This is another instance of a general phenomenon that we observe: in the worst case there is an exponential jump with respect to computational complexity when moving from the (uncompressed) word/membership problem to its compressed variant. This exponential jump is well known also from other work on the complexity of succinct problems [21, 66, 68, 69], where Boolean circuits/formulas are used for the succinct representation of graphs. But whereas for these formalisms general upgrading theorems can be shown, which roughly state that completeness of problem for a complexity class C implies completeness of the compressed variant for the exponentially harder version of C, such an upgrading theorem fails for straight-line programs: The compressed membership problem for a language may be of the same complexity as the language itself (Proposition 8.5). Thus, the relationship between a computational problem and its straight-line compressed variant is quite loose. A similar phenomenon was observed in the context of hierarchical graph descriptions [38], which can be seen as graph generating straight-line programs.

An extended abstract of this paper appeared in [43].

2. Related work. One of the most intensively studied problems on compressed strings is the pattern matching problem, see e.g. [19, 23, 49, 60]. In these papers, strings are either compressed using a variant of Lempel-Ziv encoding or straight-line programs. Compressed membership problems for straight-line compressed words were investigated for the first time in [23, 55, 59], see also [62] for a recent survey.

The problem of checking whether for a given input string s and a given integer n there exists a straight-line program of size at most n that generates s is NP-complete [37]. A smallest straight-line program generating a given input string is even hard to approximate up to some constant factor unless P = NP [37]. Practical algorithms for generating a small straight-line program that produces a given input string were proposed and analyzed with respect to their approximation ratios in [16, 37, 61].

Algorithmic problems on compressed data were also investigated for more general structures than only strings. Complexity theoretical investigations of computational problems on compressed graphs can be found in [13, 20, 21, 38, 44, 53, 66, 67, 68, 69]. In [13, 21, 53, 68, 69] (resp. [66]) Boolean circuits (resp. formulas) are used for compression, [20, 67] uses OBBDs, and in [38, 44] graphs are represented via hierarchical graph descriptions. The latter formalism can be seen as an adaptation of the idea of grammar-based compression to the context of graphs. Recently, grammar-based compression was also used in the context of XML in order to obtain succinct representations of large XML documents [14, 45, 46, 71]. Here, context-free tree grammars are used as a compact representation of XML-skeletons.

3. Preliminaries.

3.1. General notations. For a binary relation $\to$ on some set, we denote by $\xrightarrow{+}$ ($\xrightarrow{*}$) the transitive (reflexive and transitive) closure of $\to$. For sets A, B, and C, we write $A = B \uplus C$, if A is the disjoint union of B and C (B or C may be empty). Let Γ be a finite alphabet. The *empty word* over Γ is denoted by ε. Let $s = a_1 a_2 \cdots a_n \in \Gamma^*$ be a word over Γ, where $a_i \in \Gamma$ for $1 \leq i \leq n$. We define $w^{\mathrm{rev}} = a_n a_{n-1} \cdots a_1$. The *alphabet of* s is $\mathrm{alph}(s) = \{a_1, \ldots, a_n\}$. The *length* of s is $|s| = n$. Furthermore for $a \in \Gamma$ we define $|s|_a = |\{i \mid a_i = a\}|$. For $1 \leq i \leq n$ let $s[i] = a_i$ and for $1 \leq i \leq j \leq n$ let $s[i,j] = a_i a_{i+1} \cdots a_j$. If $i > j$ we set $s[i,j] = \varepsilon$. For a subalphabet $\Sigma \subset \Gamma$ we define the projection morphism $\pi_\Sigma : \Gamma^* \to \Sigma^*$ by $\pi_\Sigma(a) = \varepsilon$ if $a \notin \Sigma$ and $\pi_\Sigma(a) = a$ if $a \in \Sigma$. For a language $L \subseteq \Gamma^*$ we define the language $\mathrm{Pref}(L) = \{w \in \Gamma^* \mid w \text{ is a prefix of some } u \in L\}$. An *involution* $\overline{}$ on Γ is a function $\overline{} : \Gamma \to \Gamma$ such that $\overline{\overline{a}} = a$ for all $a \in \Gamma$. It may have fixed points, i.e., $\overline{a} = a$ for some $a \in \Gamma$. The involution $\overline{} : \Gamma \to \Gamma$ can be extended to an involution on Γ^* by setting $\overline{a_1 \cdots a_n} = \overline{a_n} \cdots \overline{a_1}$. By $\overline{\Gamma} = \{\overline{a} \mid a \in \Gamma\}$ we will always denote a disjoint copy of the alphabet Γ. Then we can define an involution $\overline{}$ on $\Delta = \Gamma \cup \overline{\Gamma}$ by setting $\overline{\overline{a}} = a$; this involution will be extended to Δ^* in the above way. A *weight-function* is a homomorphism $f : \Gamma^* \to \mathbb{N}$ from the free monoid Γ^* with concatenation to the natural numbers with addition such that $f(s) = 0$ if and only if $s = \varepsilon$. Given a linear oder $\succ$ on the alphabet Γ, we extend $\succ$ to a linear order on Γ^*, called the *lexicographic extension of* $\succ$, as follows: $u \succ v$ if either v is a prefix of u or there exist factorizations $u = wau'$ and $v = wbv'$ with $a, b \in \Gamma$ and $a \succ b$. For two monoids $\mathcal{M}_1$ and $\mathcal{M}_2$ we write $\mathcal{M}_1 \cong \mathcal{M}_2$ if $\mathcal{M}_1$ and $\mathcal{M}_2$ are isomorphic.

3.2. Complexity theory. We assume that the reader is familiar with standard complexity classes like P, coNP, PSPACE, and EXPSPACE, see, e.g., [52] for more details. All hardness results in this paper refer to logspace reductions unless some stronger form of reductions is mentioned explicitly. Several times we will use the fact that addition and multiplication of integers with $n^{O(1)}$ many bits can be done in space $O(\log(n))$. In Section 8 we will make use of *alternating Turing-machines* with logarithmic running times. An alternating Turing-machine is a nondeterministic Turing-machine, where the set of states is partitioned into existential and universal states [15]. A configuration with a universal (resp. existential) state is accepting if every (resp. some) successor state is accepting. An alternation in a computation of an alternating Turing-machine is a transition from a universal state to an existential state or vice versa. Following [15], we add a random access mechanism to the ordinary Turing-machine model when dealing with sublogarithmic time bounds: There exists a special address tape that contains a binary coded number p. If the machine enters a special query state, then it has random access to the p-th symbol of the input. This mechanism allows a Turing-machine to reach every input position in logarithmic time. If the address tape contains a position, which is larger than the length of the input, then querying the input yields some distinguished special symbol. With *DLOGTIME* (resp. *ALOGTIME*) we denote the class of languages that can be recognized on a deterministic (resp. alternating) Turing-machine in time $O(\log(n))$. It is known that ALOGTIME is equal to uniform NC^1 [3], which is the class of all languages that can be recognized by a uniform family of polynomial-size, logarithmic-depth, fan-in 2 Boolean circuits. Functions computable in uniform NC^1 are defined analogously by allowing circuits with more than one output. For the considerations in this paper, it is not necessary to go into the quite technical details of the precise definition of uniformity. We just remark that Ruzzo's U_{E^*}-uniformity of circuit families [58] would be suitable for all purposes in this paper. A language $L \subseteq \{0,1\}^*$ is NC^1-many-one reducible to a language $K \subseteq \{0,1\}^*$, briefly $L \leq_{\mathrm{NC}^1} K$, if there exists a function f which is computable in uniform NC^1 such that for all $x \in \{0,1\}^*$, $x \in L$ if and only if $f(x) \in K$.

3.3. Semi-Thue systems and finitely presented monoids. Presentations for monoids are the basic concept of this work. In this section we will introduce the necessary definitions. For more details and references see [11, 33].

Let Γ be a finite alphabet. A *semi-Thue system* R over Γ is a finite subset $R \subseteq \Gamma^* \times \Gamma^*$, whose elements are called rules. A rule $(s,t) \in R$ will be also written as $s \to t$. The pair (Γ, R) is called a *monoid presentation*. The sets $\mathrm{dom}(R)$ of all left-hand sides and $\mathrm{ran}(R)$ of all right-hand sides are defined by $\mathrm{dom}(R) = \{s \mid \exists t : (s,t) \in R\}$ and $\mathrm{ran}(R) = \{t \mid \exists s : (s,t) \in R\}$, respectively. We define two binary relations $\to_R$ and $\leftrightarrow_R$ on Γ^* as follows:

- $s \to_R t$ if there exist $u, v \in \Gamma^*$ and $(\ell, r) \in R$ with $s = u\ell v$ and $t = urv$ (the *one-step rewrite relation*)
- $s \leftrightarrow_R t$ if ($s \to_R t$ or $t \to_R s$)

We also write $t \;{}_R\!\!\leftarrow s$ in case $s \to_R t$. Let $\mathrm{RED}(R) = \Gamma^* \cdot \mathrm{dom}(R) \cdot \Gamma^*$ be the set of *reducible words* and $\mathrm{IRR}(R) = \Gamma^* \setminus \mathrm{RED}(R)$ be the set of *irreducible words* (with respect to R). The presentation (Γ, R) is *terminating* if there does not exist an infinite chain $s_1 \to_R s_2 \to_R s_3 \to_R \cdots$ in Γ^*. The presentation (Γ, R) is *confluent* if for all $s, t, u \in \Gamma^*$ with $t \;{}_R\!\!\overset{*}{\leftarrow} s \overset{*}{\to}_R u$ there exists $v \in \Gamma^*$ with $t \overset{*}{\to}_R v \;{}_R\!\!\overset{*}{\leftarrow} u$. It is well-known that (Γ, R) is confluent if and only if (Γ, R) is *Church-Rosser*, i.e., for all $s, t \in \Gamma^*$, if $s \overset{*}{\leftrightarrow}_R t$, then $s \overset{*}{\to}_R u \;{}_R\!\!\overset{*}{\leftarrow} t$ for some $u \in \Gamma^*$. The presentation (Γ, R) is *locally confluent* if for all $s, t, u \in \Gamma^*$ with $t \;{}_R\!\!\leftarrow s \to_R u$ there exists $v \in \Gamma^*$ with $t \overset{*}{\to}_R v \;{}_R\!\!\overset{*}{\leftarrow} u$. By Newman's Lemma, a terminating presentation is confluent if and only if it is locally confluent. Moreover, if (Γ, R) is terminating and confluent, then for every $s \in \Gamma^*$ there exists a unique *normal form* $\mathrm{NF}_R(s) \in \mathrm{IRR}(R)$ such that $s \overset{*}{\to}_R \mathrm{NF}_R(s)$. It is undecidable whether a given presentation is terminating, confluent, or locally confluent, respectively. On the other hand, for a terminating presentation, local confluence (and hence by Newman's Lemma also confluence) can be checked using *critical pairs*, which result from overlapping left-hand sides.

The relation $\overset{*}{\leftrightarrow}_R$ is a congruence relation with respect to the concatenation of words, it is called the *Thue-congruence* generated by (Γ, R). Hence we can define the quotient monoid $\Gamma^*/\overset{*}{\leftrightarrow}_R$, which we denote by $\mathcal{M}(\Gamma, R)$. It is called a *finitely presented monoid*, and we say that $\mathcal{M}(\Gamma, R)$ is the *monoid presented by* (Γ, R).

A decision problem that is of fundamental importance in the theory of monoid presentations is the word problem. Let (Γ, R) be a fixed presentation. The *word problem* for (Γ, R) is the following decision problem:

INPUT: Two words $s, t \in \Gamma^*$.

QUESTION: Does $s \overset{*}{\leftrightarrow}_R t$ hold, i.e., do s and t represent the same element of the monoid $\mathcal{M}(\Gamma, R)$?

Here, the input size is $|s| + |t|$.

If (Γ, R) and (Σ, S) are presentations such that $\mathcal{M}(\Gamma, R) \cong \mathcal{M} \cong \mathcal{M}(\Sigma, S)$, then for every $a \in \Gamma$ there exists a word $w_a \in \Sigma^*$ such that a and w_a represent the same element of $\mathcal{M}$. If we define the homomorphism $h : \Gamma^* \to \Sigma^*$ by $h(a) = w_a$ then for all $s, t \in \Gamma^*$ we have $s \overset{*}{\leftrightarrow}_R t$ if and only if $h(s) \overset{*}{\leftrightarrow}_S h(t)$. Thus, the word problem for (Γ, R) can be reduced to the word problem for (Σ, S) and vice versa. Moreover, this reduction can be realized in deterministic logspace (even in uniform TC^0 [35]). Thus, the decidability and complexity of the word problem do not depend on the chosen presentation. Since we are only interested in decidability and complexity questions for word problems, we may just speak of the word problem for the monoid $\mathcal{M}$.

It is well-known that in case (Γ, R) is a terminating and confluent presentation, then the word problem for $\mathcal{M}(\Gamma, R)$ is decidable: in order to check whether $u \overset{*}{\leftrightarrow}_R v$ it suffices to verify whether $\mathrm{NF}_R(u) = \mathrm{NF}_R(v)$. On the other hand, this algorithm does not yield any upper bound on the computational complexity of the word problem [5]. Complexity results on word problems for restricted classes of finitely presented monoids can be found for instance in [10, 41, 42].

3.4. Grammar based compression. Following [55], a *straight-line program (SLP)* (over the terminal alphabet Γ) is a restricted context-free grammar $G = (V, \Gamma, S, P)$ (where V is the set of nonterminals, Γ is the set of terminals, $S \in V$ is the initial nonterminal, and $P \subseteq V \times (V \cup \Gamma)^*$ is the set of productions) such that:

- for every $X \in V$ there exists exactly one production of the form $(X, \alpha) \in P$ for $\alpha \in (V \cup \Gamma)^*$, and
- there exists a linear order $\prec$ on the set of nonterminals V such that $X \prec Y$ whenever there exists a production of the form $(X, \alpha) \in P$ with $Y \in \mathrm{alph}(\alpha)$.[1]

Clearly, the language generated by the SLP G consists of exactly one word that is denoted by $\mathrm{eval}(G)$. More generally, from every nonterminal $X \in V$ we can generate exactly one word that is denoted by $\mathrm{eval}_G(X)$ (thus $\mathrm{eval}(G) = \mathrm{eval}_G(S)$). We omit the index G if the underlying SLP is clear from the context. We also write $P(G)$ for the set of productions P. The size of G is $|G| = \sum_{(X,\alpha)\in P} |\alpha|$. Note that every SLP can be transformed in polynomial time into an equivalent SLP in *Chomsky normal form*, i.e., all productions have the form (A, a) with $a \in \Gamma$ or (A, BC) with $B, C \in V$.

Example 3.1. *Consider the straight-line program G_7 over the terminal alphabet $\{a, b\}$ that consists of the following productions:*

$$X_i \to X_{i-1}X_{i-2} \text{ for } 3 \leq i \leq 7, \quad X_2 \to a, \quad X_1 \to b.$$

X_7 is the start nonterminal. Then $\mathrm{eval}(G_7) =$ *abaababaabaab, which is the 7-th Fibonacci word. The straight-line program G_7 is in Chomsky normal form and* $|G_7| = 12$.

Sometimes we will also allow exponential expressions of the form A^i for $A \in V$ and $i \in \mathbb{N}$ in the right-hand sides of productions. Here the number i is coded binary. Such an expression can be replaced by a sequence of ordinary productions, where the length of that sequence is bounded polynomially in the length of the binary coding of i. The following lemma collects several simple algorithmic properties of SLPs that will be used several times in this paper.

Lemma 3.2. *The following tasks can be easily solved in polynomial time:*

1. *Given an SLP G, calculate* $|\mathrm{eval}(G)|$ *and* $\mathrm{alph}(\mathrm{eval}(G))$.

[1] The term "straight-line program" is used for such a context-free grammar, because a production $A \to \alpha$ corresponds to a definition $A := \alpha$. Thus, the whole context-free grammar can be interpreted as a linear sequence of instructions, i.e., a straight-line program. Usually one allows in straight-line programs also instructions, where the defined variable appears on the right-hand side (e.g., $x := x+1$). But instructions of this kind can be easily eliminated by introducing additional variables.

2. *Given an SLP G and a number $i \in \{1, \dots, |\mathrm{eval}(G)|\}$, calculate* $\mathrm{eval}(G)[i]$.
3. *Given an SLP G over the terminal alphabet Γ and a homomorphism $\rho : \Gamma^* \to \Sigma^*$, calculate an SLP H such that* $\mathrm{eval}(H) = \rho(\mathrm{eval}(G))$ *(this is in fact possible in logspace).*

The proofs of these statements are straight-forward. The following result from [30, 54] is much harder to obtain:

LEMMA 3.3 ([30, 54]). *For two given SLPs G_1 and G_2, we can decide in polynomial time whether* $\mathrm{eval}(G_1) = \mathrm{eval}(G_2)$.

It is open, whether this problem is P-complete. In this work, we will consider the following generalization: Let (Γ, R) be a fixed monoid presentation. The *compressed word problem* for the monoid $\mathcal{M}(\Gamma, R)$ is the following problem:

INPUT: Two SLPs G_1 and G_2 over the terminal alphabet Γ.

QUESTION: Does $\mathrm{eval}(G_1) \overset{*}{\leftrightarrow}_R \mathrm{eval}(G_2)$ hold?

Here, the input size is $|G_1| + |G_2|$. Analogously to the uncompressed word problem, the complexity of the compressed word problem does not depend on the chosen presentation (for this, Lemma 3.2, statement 3, can be used). For a fixed language $L \subseteq \Gamma^*$ we will also consider the *compressed membership problem* for the language L, which is the following problem:

INPUT: An SLP G over the terminal alphabet Γ.

QUESTION: Does $\mathrm{eval}(G) \in L$ hold?

We can view the compressed word problem also from another perspective. A *circuit $\mathcal{C}$ over* $\mathcal{M}(\Gamma, R)$ is a finite directed acyclic graph with exactly one node of outdegree 0. The nodes of indegree 0 are labeled with elements from Γ. All nodes of indegree greater than zero are labeled with the multiplication of the monoid $\mathcal{M}(\Gamma, R)$. Such a circuit computes in a natural way an element of $\mathcal{M}(\Gamma, R)$. Then, the compressed word problem for $\mathcal{M}(\Gamma, R)$ is equivalent to the question, whether two given circuits over $\mathcal{M}(\Gamma, R)$ compute the same monoid element. This question has been considered in [7] for the case of finite monoids. Clearly, for a finite monoid, the compressed word problem can be solved in polynomial time. In [7], it was shown that for a finite non-solvable monoid the compressed word problem is P-complete, whereas for every finite solvable monoid the compressed word problem belongs to DET (the class of all problems that are NC^1-reducible to the calculation of an integer determinant [17]) and hence to NC^2. Due to the tight correspondence between finite monoids and regular languages, every compressed word problem for a finite monoid is equivalent to the compressed membership problem for a specific regular language and vice versa. Thus, [7] gives a complete classification of the complexity of the compressed membership problem for regular languages. Our work can be seen as a first step of an extension of the work from [7] to finitely presented infinite monoids.

Finally, let us remark that most of our complexity results can be transfered to other compression schemes, like for instance Lempel-Ziv 77, briefly LZ77 [73]. If w is a string and G is an SLP of size n with $\mathrm{eval}(G) = w$, then $\mathrm{LZ}(w)$ (the LZ77-compressed representation of w) has size $O(n)$ and can be constructed in polynomial time [55, 61]. On the other hand, if n is the size of $\mathrm{LZ}(w)$, then we can construct in polynomial time an SLP of size $O(n^2 \cdot \log(n))$ that generates w [55]. Thus, if we allow polynomial time reductions, all our hardness results for complexity classes above P also hold, if we use LZ77 for compression. Since the transformation from an SLP to the LZ77-compressed representation might be P-hard, we cannot transfer directly P-hardness results for straight-line programs to LZ77.

4. Compressed word problems in P. As already mentioned in the previous section, for every finite monoid the compressed word problem is solvable in polynomial time. In this section we will present a class of infinite monoids with polynomial time solvable compressed word problems. This class contains all free groups. We will also prove that the compressed word problem for every free group of rank at least 2 is P-complete.

A presentation (Γ, R) is called 2-homogeneous if for every $(\ell, r) \in R$: $|\ell| = 2$ and $r = \varepsilon$ [9]. In [42] it was shown that for every 2-homogeneous presentation the (uncompressed) word problem is in logspace. Moreover, the uniform variant of the word problem for 2-homogeneous presentations, where the presentation is part of the input, is complete for symmetric logspace [42].

The following result was shown by Book [9]:

PROPOSITION 4.1 ([9]). *For every 2-homogeneous presentation (Γ, R) there exists a 2-homogeneous and confluent presentation (Σ, S) with $\mathcal{M}(\Gamma, R) \cong \mathcal{M}(\Sigma, S)$.*

For the further consideration let us fix a 2-homogeneous presentation (Γ, R). By Proposition 4.1 we may assume that (Γ, R) is confluent. The following lemma is easy to prove.

LEMMA 4.2. *Let $u, v \in \mathrm{IRR}(R)$. Then there exist factorizations $u = u_1u_2$ and $v = v_1v_2$ such that $\mathrm{NF}_R(uv) = u_1v_2$, $|u_2| = |v_1|$, and $u_2v_1 \xrightarrow{*}_R \varepsilon$.*

The following lemma was shown in [42].

LEMMA 4.3 ([42]). *There exists a partition $\Gamma = \Sigma_\ell \uplus \Sigma_r \uplus \Delta$ and an involution $\overline{} : \Delta \to \Delta$ with $\{(a\overline{a}, \varepsilon) \mid a \in \Delta\} \subseteq R \subseteq \{(a\overline{a}, \varepsilon) \mid a \in \Delta\} \cup \{(ab, \epsilon) \mid a \in \Sigma_\ell,\ b \in \Sigma_r\}$.*

We say that (Γ, R) is *N-free*, if there do not exist $a, b \in \Sigma_\ell$ and $c, d \in \Sigma_r$ (where Σ_ℓ and Σ_r result from Lemma 4.3) such that $ac, ad, bc \in \mathrm{dom}(R)$ but $bd \notin \mathrm{dom}(R)$. N-freeness means that the bipartite graph $(\Sigma_\ell \cup \Sigma_r, \{(a, b) \in \Sigma_\ell \times \Sigma_r \mid (ab, \varepsilon) \in R\})$ does not contain an N-shaped induced subgraph, i.e., it is a disjoint union of complete bipartite graphs.

EXAMPLE 4.4. *Let $\Gamma = \{a, \overline{a}, b, \overline{b}\}$ and $R = \{(a\overline{a}, \varepsilon), (\overline{a}a, \varepsilon), (b\overline{b}, \varepsilon), (\overline{b}b, \varepsilon)\}$. Then (Γ, R) is 2-homogeneous, confluent, and N-free. In fact, we have $\Delta = \Gamma$ and $\Sigma_\ell = \Sigma_r = \emptyset$. The monoid $\mathcal{M}(\Gamma, R)$ is the free group of rank 2, see also the paragraph before Theorem 4.9. If $\Gamma = \{a, b, c, d\}$ and $R = \{(ac, \varepsilon), (ad, \varepsilon), (bc, \varepsilon)\}$, then (Γ, R) is 2-homogeneous and confluent but not N-free. In fact, (Γ, R) is contained in every 2-homogeneous and confluent presentation, which is not N-free.*

THEOREM 4.5. *If (Γ, R) is 2-homogeneous, confluent, and N-free, then the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in P.*

In the next section we will see that Theorem 4.5 cannot be extended to non-N-free presentations unless P = NP.

The proof of Theorem 4.5 will be presented after introducing *composition systems* [23] — a generalization of straight-line programs. A composition system $G = (V, \Gamma, S, P)$ is defined analogously to an SLP, but in addition to productions of the form $A \to \alpha$ ($A \in V$, $\alpha \in (V \cup \Gamma)^*$) it may also contain productions of the form $A \to B[i, j]$ for $B \in V$ and $i, j \in \mathbb{N}$. For such a production we define $\mathrm{eval}_G(A) = \mathrm{eval}_G(B)[i, j]$, i.e., we select from $\mathrm{eval}_G(B)$ the subword from position i to position j.[2] We also allow more general rules like for instance $A \to B[i, j]C[k, \ell]$, of course this does not lead to higher compression rates. As for SLPs we define $\mathrm{eval}(G) = \mathrm{eval}_G(S)$. The following result from [23] generalizes Lemma 3.3:

[2]In [23], a slightly more restricted formalism, where only productions of the form $A \to B[j, |\mathrm{eval}_G(B)|]C[1, i]$ are allowed, was introduced. But this definition is easily seen to be equivalent to our formalism.

LEMMA 4.6 ([23]). *For two given composition systems* G_1 *and* G_2*, we can decide in polynomial time whether* $\mathrm{eval}(G_1) = \mathrm{eval}(G_2)$.

The following result was shown in [29, Chap. 8]:

LEMMA 4.7 ([29]). *A given composition system can be transformed in polynomial time into an SLP that generates the same word.*

Lemma 4.7 leads to an alternative proof of Lemma 4.6: transform the two given composition systems in polynomial time into equivalent SLPs and apply Lemma 3.3. Moreover, Lemma 4.7 implies that all statements from Lemma 3.2 also hold for composition systems.

LEMMA 4.8. *Assume that* (Γ, R) *is 2-homogeneous, confluent, and* N*-free. Then the following problem belongs to P:*

INPUT: Composition systems G_1 *and* G_2 *with* $\mathrm{eval}(G_1), \mathrm{eval}(G_2) \in \mathrm{IRR}(R)$ *and* $|\mathrm{eval}(G_1)| = |\mathrm{eval}(G_2)|$.

QUESTION: Does $\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xrightarrow{*}_R \varepsilon$ *hold?*

Proof. Let $\Gamma = \Sigma_\ell \uplus \Sigma_r \uplus \Delta$ be the partition resulting from Lemma 4.3 and $\overline{} : \Delta \to \Delta$ be the corresponding involution on Δ. Note that $\mathrm{eval}(G_1), \mathrm{eval}(G_2) \in \mathrm{IRR}(R)$ and $\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xrightarrow{*}_R \varepsilon$ implies that $\mathrm{eval}(G_1) \in (\Sigma_\ell \cup \Delta)^*$ and $\mathrm{eval}(G_2) \subseteq (\Sigma_r \cup \Delta)^*$. Thus, we first check in polynomial time whether this is true; if not we can reject. Next, from G_2 we can easily construct (by reversing productions) a composition system G_2' with $\mathrm{eval}(G_2') = \mathrm{eval}(G_2)^{\mathrm{rev}}$. Since (Γ, R) is N-free, we can find partitions $\Sigma_\ell = \biguplus_{i=1}^k \Sigma_{\ell,i}$ and $\Sigma_r = \biguplus_{i=1}^k \Sigma_{r,i}$ such that

$$R = \{(a\overline{a}, \varepsilon) \mid a \in \Delta\} \cup \bigcup_{i=1}^{k} \{(ab, \varepsilon) \mid a \in \Sigma_{\ell,i}, b \in \Sigma_{r,i}\}.$$

Let us take new symbols $a_1, \ldots, a_k$ and define homomorphisms ρ_1 and ρ_2 by $\rho_1(a) = \rho_2(a) = a_i$ for all $a \in \Sigma_{\ell,i} \cup \Sigma_{r,i}$, $1 \le i \le k$, $\rho_1(a) = a$ for all $a \in \Delta$ and $\rho_2(a) = \overline{a}$ for all $a \in \Delta$. From G_1 and G_2' we can construct in polynomial time composition systems H_1, H_2 such that $\mathrm{eval}(H_1) = \rho_1(\mathrm{eval}(G_1))$ and $\mathrm{eval}(H_2) = \rho_2(\mathrm{eval}(G_2'))$. By construction, we have $\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xrightarrow{*}_R \varepsilon$ if and only if $\mathrm{eval}(H_1) = \mathrm{eval}(H_2)$. The latter identity can be verified in polynomial time by Lemma 4.6. ◻

Proof of Theorem 4.5. Let (Γ, R) be a fixed 2-homogeneous, confluent, and N-free presentation. Given SLPs G_1 and G_2 over the terminal alphabet Γ, we have to verify in polynomial time, whether $\mathrm{NF}_R(\mathrm{eval}(G_1)) = \mathrm{NF}_R(\mathrm{eval}(G_2))$. By Lemma 4.6, it suffices to prove that given an SLP G in Chomsky normal form over the terminal alphabet Γ, we can construct in polynomial time a composition system H such that $\mathrm{eval}(H) = \mathrm{NF}_R(\mathrm{eval}(G))$. We construct H inductively by adding more and more rules. Initially, we put into $P(H)$ all productions from $P(G)$ of the form $A \to a$ with $a \in \Gamma$. Now assume that $A \to BC$ belongs to $P(G)$ and that H already contains enough productions such that $\mathrm{eval}_H(B) = \mathrm{NF}_R(\mathrm{eval}_G(B))$ and $\mathrm{eval}_H(C) = \mathrm{NF}_R(\mathrm{eval}_G(C))$. We first calculate the largest i such that

$$\mathrm{eval}_H(B) = u_1u_2, \quad \mathrm{eval}_H(C) = v_1v_2, \quad |u_2| = |v_1| = i, \quad u_2v_1 \xrightarrow{*}_R \varepsilon. \tag{4.1}$$

Lemma 4.2 implies that $\mathrm{NF}_R(\mathrm{eval}_G(A)) = u_1v_2$. For a given $i \in \mathbb{N}$, we can check condition (4.1) in polynomial time by Lemma 4.8. Since i is bounded exponentially in the input size, the largest i satisfying (4.1) can be calculated in polynomial time by doing a binary search. For this largest i we add to the current H the production

$A \to B[1, |\mathrm{eval}_H(B)|-i]C[i+1, |\mathrm{eval}_H(C)|]$. This concludes the proof of Theorem 4.5. $\square$

For a finite alphabet Γ, the *free group* $F(\Gamma)$ *generated by* Γ is defined as

$$F(\Gamma) = \mathcal{M}(\Gamma \cup \overline{\Gamma}, \{(c\overline{c}, \varepsilon) \mid c \in \Gamma \cup \overline{\Gamma}\}). \tag{4.2}$$

Recall from Section 3.1 that we define an involution on $\Gamma \cup \overline{\Gamma}$ by setting $\overline{\overline{a}} = a$. Clearly, $F(\Gamma)$ is indeed a group. In case $|\Gamma| = n$ we also write F_n instead of $F(\Gamma)$ and call it the *free group of rank* n. It is known that the (uncompressed) word problem for a free group is in logspace [40]. Moreover, the word problem for F_2 hard for uniform NC^1 [57]. By Theorem 4.5, the compressed word problem for every free group F_n is in P (we have $\Sigma_\ell = \Sigma_r = \emptyset$ for the presentation in (4.2)). This upper bound is also sharp:

THEOREM 4.9. *The compressed word problem for F_2 is P-complete.*

Proof. It suffices to prove P-hardness, which will be done by a reduction from the monotone circuit value problem, i.e., the problem whether a Boolean circuit consisting of AND and OR gates evaluates to TRUE [25]. We will use the following result, which is proved in [57]: Let $\Gamma = \{a, b\}$ and $x, y \in (\Gamma \cup \overline{\Gamma})^*$ such that $|x| = |y| = k$ and $|x|_a - |x|_{\bar{a}} = |y|_a - |y|_{\bar{a}} = 0$. Then, if we interpret x and y as elements from F_2, the following holds, where 1 denoted the neutral element of F_2:

$$\begin{aligned} (x = 1) \vee (y = 1) \quad &\Leftrightarrow \quad \bar{a}^{3k} x a^{3k} y \bar{a}^{3k} \overline{x} a^{3k} \overline{y} = 1 \\ (x = 1) \wedge (y = 1) \quad &\Leftrightarrow \quad \bar{a}^{3k} x a^{3k} y \bar{a}^{3k} x a^{3k} y = 1 \end{aligned}$$

Note that the words on the right of these equivalences have length $16k$ and that the number of a's minus the number $\bar{a}$'s is again 0.

Now let C be a monotone Boolean circuit. W.l.o.g. we can assume that C is layered, i.e., the gates of C are partitioned into n layers and a gate in layer $i > 1$ receives its inputs from layer $i-1$ see, e.g., [28, Problem A.1.6]. Layer 1 contains the input gates and layer n contains the unique output gate. We now construct an SLP $G(C)$ as follows. For every gate z of C, G contains two nonterminals A_z and $A_{\bar{z}}$. The nonterminal A_z will evaluate to a string that represents the 1 of F_2 if and only if gate z of the circuit evaluates to TRUE. The nonterminal $A_{\bar{z}}$ evaluates to the inverse of $\mathrm{eval}_{G(C)}(A_z)$ in F_2. Moreover, we will have $|\mathrm{eval}_{G(C)}(A_z)| = |\mathrm{eval}_{G(C)}(A_{\bar{z}})| = 2 \cdot 16^{i-1}$ if z is located in the i-th layer of the circuit ($1 \le i \le n$).

For every input gate x in layer 1 we introduce the productions

$$A_x \to \begin{cases} a\bar{a} & \text{if input gate } x \text{ is TRUE} \\ b^2 & \text{if input gate } x \text{ is FALSE} \end{cases}$$

$$A_{\bar{x}} \to \begin{cases} a\bar{a} & \text{if input gate } x \text{ is TRUE} \\ \bar{b}^2 & \text{if input gate } x \text{ is FALSE} \end{cases}$$

If z is an OR gate in the ith layer ($i \ge 2$) with input gates x and y from the $(i-1)$-th layer, then the productions for A_z and $A_{\bar{z}}$ are

$$\begin{aligned} &A_z \to \bar{a}^{6 \cdot 16^{i-2}} A_x a^{6 \cdot 16^{i-2}} A_y \bar{a}^{6 \cdot 16^{i-2}} A_{\bar{x}} a^{6 \cdot 16^{i-2}} A_{\bar{y}} \text{ and} \\ &A_{\bar{z}} \to A_y \bar{a}^{6 \cdot 16^{i-2}} A_x a^{6 \cdot 16^{i-2}} A_{\bar{y}} \bar{a}^{6 \cdot 16^{i-2}} A_{\bar{x}} a^{6 \cdot 16^{i-2}}. \end{aligned}$$

Note that the binary codings of the exponents $6 \cdot 16^{i-2}$ have polynomial length and hence each of the above productions can be replaced by a sequence of ordinary productions. Moreover, if $|\mathrm{eval}(A_u)| = 2 \cdot 16^{i-2}$ for $u \in \{x, \bar{x}, y, \bar{y}\}$ (which is true if x and

y are located in the first layer, i.e., $i = 2$), then $|\mathrm{eval}(A_z)| = |\mathrm{eval}(A_{\bar{z}})| = 2 \cdot 16^{i-1}$. If z is an AND gate in the ith layer ($i \geq 2$) with input gates x and y, then the productions for A_z and $A_{\bar{z}}$ are

$$A_z \to \bar{a}^{6\cdot 16^{i-2}} A_x a^{6\cdot 16^{i-2}} A_y \bar{a}^{6\cdot 16^{i-2}} A_x a^{6\cdot 16^{i-2}} A_y \text{ and}$$
$$A_{\bar{z}} \to A_{\bar{y}} \bar{a}^{6\cdot 16^{i-2}} A_{\bar{x}} a^{6\cdot 16^{i-2}} A_{\bar{y}} \bar{a}^{6\cdot 16^{i-2}} A_{\bar{x}} a^{6\cdot 16^{i-2}}.$$

Once again, these productions can be replaced by a sequence of ordinary productions. Let o be the unique output gate of the circuit C. Then, by the result from [57], the circuit C evaluates to TRUE if and only if $\mathrm{eval}_{G(C)}(A_o) = 1$ in F_2. ☐

5. Compressed word problems between P and PSPACE. In this section we will consider 2-homogeneous and confluent presentations that are not necessarily N-free. Recall that P^{NP} is the class of all languages that can be accepted by a deterministic polynomial time machine that has additional access to an NP-oracle [64]. P^{NP} is also denoted by Δ_2^p in the literature; it is contained in the second level of the polynomial time hierarchy and hence in PSPACE. Several complete problems for P^{NP} can be found in [34].

THEOREM 5.1. *If* (Γ, R) *is 2-homogeneous and confluent (but not necessarily* N*-free), then the compressed word problem for* $\mathcal{M}(\Gamma, R)$ *is in* P^{NP}.

Proof. The key observation is that for a 2-homogeneous and confluent (but not necessarily N-free) presentation (Γ, R) the problem from Lemma 4.8 is in coNP, i.e., the complementary condition is in NP: If $\mathrm{eval}(G_1), \mathrm{eval}(G_2) \in \mathrm{IRR}(R)$ and $|\mathrm{eval}(G_1)| = |\mathrm{eval}(G_2)|$, then

$$\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xrightarrow{*}_R \varepsilon \text{ does not hold}$$

if and only if there exists $1 \leq i \leq |\mathrm{eval}(G_1)|$ with

$$\mathrm{eval}(G_1)[i]\, \mathrm{eval}(G_2)[|\mathrm{eval}(G_2)| - i + 1] \notin \mathrm{dom}(R).$$

For a given i, the latter condition can be checked in polynomial time. Now the decision procedure from the proof of Theorem 4.5 in Section 4 gives us a $\mathrm{P}^{\mathrm{coNP}}$-, i.e., P^{NP}-algorithm in the present situation. ☐

By the next result, coNP-hardness can be obtained for every 2-homogeneous presentations that is not N-free:

THEOREM 5.2. *Let* $\Gamma = \{a, b, c, d\}$ *and* $R = \{(ac, \varepsilon), (ad, \varepsilon), (bc, \varepsilon)\}$. *The compressed word problem for* $\mathcal{M}(\Gamma, R)$ *is coNP-hard.*

Proof. The following problem is the complementary problem to SUBSETSUM [22, Problem SP13] and hence coNP-complete:

INPUT: Binary coded integers $w_1, \ldots, w_n, t \geq 0$

QUESTION: For all $x_1, \ldots, x_n \in \{0, 1\}$, $\sum_{i=1}^n x_i \cdot w_i \neq t$?

Let us fix binary coded integers $w_1, \ldots, w_n, t \geq 0$. Let $\overline{w}_k = (w_1, \ldots, w_k)$, and $\overline{w} = \overline{w}_n$. Let $\overline{1}_k = (1, \ldots, 1)$ be the k-dimensional vector with all entries equal to 1. For vectors $\overline{x} = (x_1, \ldots, x_m)$ and $\overline{y} = (y_1, \ldots, y_m)$ we define $\overline{x} \cdot \overline{y} = x_1 y_1 + \cdots + x_m y_m$. Finally, let $s_k = \overline{1}_k \cdot \overline{w}_k = w_1 + \cdots + w_k$, and $s = s_n = w_1 + \cdots + w_n$.

We construct two SLPs G_1 and G_2 over the terminal alphabets $\{a, b\}$ and $\{c, d\}$, respectively, such that $\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xrightarrow{*}_R \varepsilon$ (i.e., $\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xleftrightarrow{*}_R \varepsilon$ since R

is confluent and hence Church-Rosser) if and only if $\overline{x} \cdot \overline{w} \neq t$ for all $\overline{x} \in \{0,1\}^n$. This will prove the theorem. First let us construct G_1:

$$\begin{aligned} S_1 &\to ba^{s+w_1}b \\ S_{k+1} &\to S_k a^{s-s_k+w_{k+1}} S_k \end{aligned}$$

Let S_n be the start nonterminal of G_1.

Claim:[3] $\mathrm{eval}_{G_1}(S_k) = \left(\prod_{\overline{x} \in \{0,1\}^k \setminus \{\overline{1}_k\}} a^{\overline{x} \cdot \overline{w}_k} b a^{s - \overline{x} \cdot \overline{w}_k} \right) a^{s_k} b$

We prove the claim by induction on k. The case $k = 1$ is clear, since $\mathrm{eval}_{G_1}(S_1) = ba^{s+w_1}b = ba^s a^{s_1} b = a^0 b a^{s-0} a^{s_1} b$. For $k+1 \leq n$ we obtain the following:

$$\left(\prod_{\overline{x} \in \{0,1\}^{k+1} \setminus \{\overline{1}_{k+1}\}} a^{\overline{x} \cdot \overline{w}_{k+1}} b a^{s - \overline{x} \cdot \overline{w}_{k+1}} \right) a^{s_{k+1}} b =$$

$$\underbrace{\left(\prod_{\overline{x} \in \{0,1\}^k} a^{\overline{x} \cdot \overline{w}_k} b a^{s - \overline{x} \cdot \overline{w}_k} \right)}_{\mathrm{eval}(S_k) a^{s-s_k}} \underbrace{\left(\prod_{\overline{x} \in \{0,1\}^k \setminus \{\overline{1}_k\}} a^{\overline{x} \cdot \overline{w}_k + w_{k+1}} b a^{s - \overline{x} \cdot \overline{w}_k - w_{k+1}} \right) a^{w_{k+1}} a^{s_k} b}_{a^{w_{k+1}} \mathrm{eval}(S_k)} =$$

$$\mathrm{eval}_{G_1}(S_k) a^{s-s_k+w_{k+1}} \mathrm{eval}_{G_1}(S_k) = \mathrm{eval}_{G_1}(S_{k+1}),$$

which proves the claim.

For $k = n$ we get

$$\mathrm{eval}(G_1) = \mathrm{eval}_{G_1}(S_n) = \prod_{\overline{x} \in \{0,1\}^n} a^{\overline{x} \cdot \overline{w}} b a^{s - \overline{x} \cdot \overline{w}}.$$

Now let G_2 be an SLP such that $\mathrm{eval}(G_2) = (c^{s-t} d c^t)^{2^n}$, which is easy to construct. Let us give an example before we continue with the proof:

EXAMPLE 5.3. *Assume that* $w_1 = 2$, $w_2 = 5$, $w_3 = 8$, *and* $t = 9$. *Thus,* $s = 2 + 5 + 8 = 15$ *and*

$$\begin{aligned} \mathrm{eval}(G_1) &= ba^{15}a^2ba^{13}a^5ba^{10}a^7ba^8a^8ba^7a^{10}ba^5a^{13}ba^2a^{15}b \\ &= ba^{17}ba^{18}ba^{17}ba^{16}ba^{17}ba^{18}ba^{17}b \\ \mathrm{eval}(G_2) &= (c^6dc^9)^8 = c^6dc^{15}dc^{15}dc^{15}dc^{15}dc^{15}dc^{15}dc^{15}dc^9. \end{aligned}$$

For this example, we have $\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xrightarrow{*}_R \varepsilon$, *because while reducing the word* $\mathrm{eval}(G_1)\mathrm{eval}(G_2)$, *in the middle of the current word the factor bd (which cannot be replaced by* ε*) will never occur.*

Note that $\mathrm{eval}(G_1) \in \{a,b\}^*$, $\mathrm{eval}(G_2) \in \{c,d\}^*$, and $|\mathrm{eval}(G_1)| = |\mathrm{eval}(G_2)| = 2^n \cdot (s+1)$. Thus, since bd is the only factor from $\{ac, ad, bc, bd\}$ that cannot be rewritten to ε, we have $\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xrightarrow{*}_R \varepsilon$ if and only if a b does not meet a d in

[3] The $\prod$-expression on the right-hand side of this production denotes the concatenation of the corresponding words, where the order of concatenation is given by the lexicographic order on the set of vectors $\{0,1\}^k$, where the last position has the highest significance.

the unique R-derivation that starts from $\mathrm{eval}(G_1)\mathrm{eval}(G_2)$. Hence, by construction of G_1 and G_2, we have $\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xrightarrow{*}_R \varepsilon$ if and only if $\overline{x} \cdot \overline{w} \neq t$ for all $\overline{x} \in \{0,1\}^n$. This concludes the proof. ▯

The precise complexity of the compressed word problem for a 2-homogeneous but not N-free presentation remains open, it is located somewhere between coNP and P^{NP}. On the other hand, by the previous proof, it is already coNP-hard to decide whether $\mathrm{eval}(G) \overset{*}{\leftrightarrow}_R \varepsilon$ for a given SLP G, in case R is 2-homogeneous and confluent but not N-free. For this restricted variant of the compressed word problem we can also prove an upper bound of coNP.

THEOREM 5.4. *For every 2-homogeneous and confluent (but not necessarily N-free) presentation (Γ, R), the following problem belongs to coNP:*

INPUT: An SLP over the terminal alphabet Γ.

QUESTION: Does $\mathrm{eval}(G) \overset{*}{\leftrightarrow}_R \varepsilon$ (i.e., $\mathrm{eval}(G) \xrightarrow{*}_R \varepsilon$) hold?

For the proof of Theorem 5.4 we first introduce a few notations.

Let us fix a 2-homogeneous and confluent (but not necessarily N-free) presentation (Γ, R) for the rest of this section. Recall that by Lemma 4.3, there exist a partition $\Gamma = \Sigma_\ell \uplus \Sigma_r \uplus \Delta$ and an involution $\overline{} : \Delta \to \Delta$ such that $\{(a\overline{a}, \varepsilon) \mid a \in \Delta\} \subseteq R \subseteq \{(a\overline{a}, \varepsilon) \mid a \in \Delta\} \cup \{(ab, \epsilon) \mid a \in \Sigma_\ell,\ b \in \Sigma_r\}$. Let $S = \{(a\overline{a}, \varepsilon) \mid a \in \Delta\} \subseteq R$, which is also 2-homogeneous and confluent, but in addition N-free. Thus, by Theorem 4.5, the compressed membership problem for the language $\{w \in \Delta^* \mid w \xrightarrow{*}_S \varepsilon\}$ can be solved in polynomial time.

Let us take two bracket symbols "$\langle$" and "$\rangle$" and define the morphism $\rho : \Gamma^* \to \{\langle, \rangle\}^*$ by $\rho(a) = \langle$ for $a \in \Sigma_\ell$, $\rho(b) = \rangle$ for $b \in \Sigma_r$, and $\rho(c) = \varepsilon$ for $c \in \Delta$. Let D_1 be the set of all well-bracketed words over $\{\langle, \rangle\}$, i.e., D_1 is the Dyck language over one bracket pair. If P is the semi-Thue system that contains the single rule $\langle\,\rangle \to \varepsilon$, then $D_1 = \{w \in \{\langle, \rangle\}^* \mid w \xrightarrow{*}_P \varepsilon\}$. Since P is 2-homogeneous, confluent, and N-free, Theorem 4.5 implies that the compressed membership problem for D_1 can be solved in polynomial time.

Now assume that $w \in \Gamma^*$ is a word such that $\rho(w) \in D_1$. We say that two positions $i, j \in \{1, \ldots, |w|\}$ are *corresponding brackets*, briefly $\mathrm{match}(i) = j$, if $w[i] \in \Sigma_\ell$, $w[j] \in \Sigma_r$, $i < j$, $\rho(w[i,j]) \in D_1$, and $\rho(w[i,k]) \notin D_1$ for all k with $i < k < j$.

EXAMPLE 5.5. *Let $\Gamma = \{a, b, c, d, x, y, \overline{x}, \overline{y}\}$ and*

$$R = \{(ac, \varepsilon), (ad, \varepsilon), (bc, \varepsilon), (x\overline{x}, \varepsilon), (\overline{x}x, \varepsilon), (y\overline{y}, \varepsilon), (\overline{y}y, \varepsilon)\}.$$

Thus, $\Delta = \{x, y, \overline{x}, \overline{y}\}$, $\Sigma_\ell = \{a, b\}$ and $\Sigma_r = \{c, d\}$. Consider the word $w = x\,a\,\overline{y}\,y\,c\,y\,\overline{x}\,b\,a\,x\,\overline{x}\,d\,x\,c\,\overline{y}\,\overline{x}$. Then $\rho(w) = \langle\,\rangle\,\langle\,\langle\,\rangle\,\rangle \in D_1$ and for instance $\mathit{match}(8) = 14$.

LEMMA 5.6. *The following problem can be solved in polynomial time:*

INPUT: An SLP G over the terminal alphabet Γ such that $\rho(\mathrm{eval}(G)) \in D_1$ and a position $1 \leq i \leq |\mathrm{eval}(G)|$ such that $\mathrm{eval}(G)[i] \in \Sigma_\ell$.

OUTPUT: The unique position $j = \mathit{match}(i)$ in $\mathrm{eval}(G)$.

Proof. In a first step we will reduce the problem to the alphabet $\{\langle, \rangle\}$. Let $1 \leq i \leq |\mathrm{eval}(G)|$ such that $\mathrm{eval}(G)[i] \in \Sigma_\ell$. First we calculate in polynomial time the unique number k such that i is the position of the k-th symbol from $\Sigma_\ell \cup \Sigma_r$ in $\mathrm{eval}(G)$. Formally, $k = |\pi_{\Sigma_\ell \cup \Sigma_r}(\mathrm{eval}(G)[1,i])|$ (by Lemma 4.7 we can calculate in polynomial time an SLP H that generates $\mathrm{eval}(G)[1,i]$; then $|\pi_{\Sigma_\ell \cup \Sigma_r}(\mathrm{eval}(H))|$ can be calculated in polynomial time using Lemma 3.2). Now assume for a moment that we can calculate the position ℓ of the bracket "$\rangle$" that corresponds to the bracket "$\langle$" at

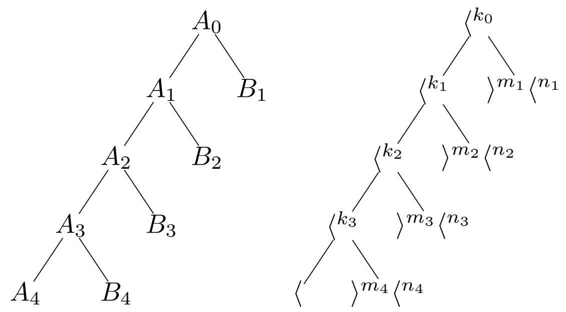


FIG. 5.1.

position k in $\rho(\mathrm{eval}(G))$ — we call this position also $\mathrm{match}(k)$. Then we just calculate the position j of the ℓ-th symbol from $\Sigma_\ell \cup \Sigma_r$ in $\mathrm{eval}(G)$ and we have $\mathrm{match}(i) = j$ in $\mathrm{eval}(G)$. Formally, j is the unique number with $\ell = |\pi_{\Sigma_\ell\cup\Sigma_r}(\mathrm{eval}(G)[1,j])|$ and $\mathrm{eval}(G)[j] \in \Sigma_\ell \cup \Sigma_r$. In order to calculate j from ℓ in polynomial time, we calculate bottom-up $|\pi_{\Sigma_\ell\cup\Sigma_r}(\mathrm{eval}_G(A))|$ as well as $|\mathrm{eval}_G(A)|$ for every nonterminal A of the SLP G. Then we can walk top-down in G in order to calculate j from ℓ.

Thus, we may assume that G is an SLP with $\mathrm{eval}(G) \in D_1$ and $1 \le i \le |\mathrm{eval}(G)|$ is a position with $\mathrm{eval}(G)[i] = \langle$. We compute in polynomial time the unique matching position $\mathrm{match}(i)$ in $\mathrm{eval}(G)$. Consider the language

$$K = \mathrm{Pref}(D_1) \setminus [(D_1 \setminus \{\varepsilon\}) \cdot \mathrm{Pref}(D_1)],$$

i.e., K is the set of all prefixes of words from D_1 that do not contain a non-empty prefix from D_1. Then

$$\mathrm{match}(i) = \max\{j > i \mid \mathrm{eval}(G)[i,j] \in K\} + 1.$$

Since K is prefix-closed, it suffices to show that the compressed membership problem for the language K is solvable in polynomial time, because then we can find the largest $j > i$ with $\mathrm{eval}(G)[i,j] \in K$ using a binary search.

Thus, let H be an SLP in Chomsky normal form. We want to check, whether $\mathrm{eval}(H) \in K = \mathrm{Pref}(D_1) \setminus [(D_1 \setminus \{\varepsilon\}) \cdot \mathrm{Pref}(D_1)]$. Recall that the semi-Thue system P consists of the single rule $\langle\rangle \to \varepsilon$. Clearly, for a word $w \in \{\langle,\rangle\}^*$, we have $\mathrm{NF}_P(w) \in \rangle^*\langle^*$. We represent a word $\rangle^n\langle^m$ by the binary coding of n and m. Using this representation, we can calculate bottom-up in polynomial time for every nonterminal A of H the normal form $\mathrm{NF}_P(\mathrm{eval}_H(A)) \in \rangle^*\langle^*$. Clearly, for every word $w \in \{\langle,\rangle\}^*$, $w \in \mathrm{Pref}(D_1)$ if and only if $\mathrm{NF}_P(w) \in \langle^*$. Using this, we can first check whether $\mathrm{eval}(H) = \mathrm{eval}_H(S) \in \mathrm{Pref}(D_1)$. If this is not the case, we reject. Thus, assume that $\mathrm{eval}(H) \in \mathrm{Pref}(D_1)$. We have to check in polynomial time whether $\mathrm{eval}(H) \in (D_1 \setminus \{\varepsilon\}) \cdot \mathrm{Pref}(D_1)$ or not.

Consider the unique path of nonterminals $S = A_0, A_1, \dots, A_m$ such that $A_m \to \langle$ and for all $i \le 0 < m$, $A_i \to A_{i+1}B_{i+1}$ are productions of H. This path is shown in Figure 5.1 on the left for $m = 4$. Since every prefix of $\mathrm{eval}(H)$ also belongs to $\mathrm{Pref}(D_1)$, we have $\mathrm{NF}_P(\mathrm{eval}_H(A_i)) = \langle^{k_i}$ for some numbers $k_i \ge 0$. Assume that $\mathrm{NF}_P(\mathrm{eval}_H(B_i)) = \rangle^{m_i}\langle^{n_i}$ for $m_i, n_i \ge 0$, see the right tree in Figure 5.1. We claim that $\mathrm{eval}(H) \in (D_1 \setminus \{\varepsilon\}) \cdot \mathrm{Pref}(D_1)$ if and only if there exists $1 \le i \le m$ such that $k_i \le m_i$, which can be checked in polynomial time. If $k_i \le m_i$ for some $1 \le i \le m$, i.e., $\mathrm{NF}_P(\mathrm{eval}_H(B_i)) = \rangle^{k_i}\rangle^{\ell}\langle^{n_i}$ for some ℓ, then there exists a prefix u of $\mathrm{eval}_H(B_i)$ such that $\mathrm{NF}_P(u) = \rangle^{k_i}$ and thus $\mathrm{eval}_H(A_i)u \in D_1$ (we may have $u = \varepsilon$ in case $k_i = 0$, i.e., $\mathrm{eval}_H(A_i) \in D_1$). The word $\mathrm{eval}_H(A_i)u$ is a nonempty prefix of $\mathrm{eval}(H)$ that belongs to D_1. On the other hand, if there exists a nonempty prefix $v \in D_1$ of $\mathrm{eval}(H)$ then let i be maximal such that v is a prefix of $\mathrm{eval}_H(A_i)$. Clearly, $i < m$, since $\mathrm{eval}_H(A_m) = \langle$. Thus, $v = \mathrm{eval}_H(A_{i+1})u$ for some prefix u of $\mathrm{eval}_H(B_{i+1})$. Since $\mathrm{NF}_P(\mathrm{eval}_H(A_{i+1})) = \langle^{k_{i+1}}$ and $\mathrm{eval}_H(A_{i+1})u \in D_1$ it follows that $\mathrm{NF}_P(u) = \rangle^{k_{i+1}}$. Since $\mathrm{NF}_P(\mathrm{eval}_H(B_{i+1})) = \rangle^{m_{i+1}}\langle^{n_{i+1}}$ and u is a prefix of $\mathrm{eval}_H(B_{i+1})$ we obtain $m_{i+1} \geq k_{i+1}$. This proves the lemma. ☐

Let us take again a word $w \in \Gamma^*$ such that $\rho(w) \in D_1$. Then we can factorize w uniquely as $w = s_0 w[i_1, j_1] s_1 \cdots w[i_n, j_n] s_n$, where $n \geq 0$, $\mathrm{match}(i_k) = j_k$ for all $k \in \{1, \ldots, n\}$ and $s_k \in \Delta^*$ for all $k \in \{0, \ldots, n\}$. We define $\mathcal{F}(w) = s_0 s_1 \cdots s_n \in \Delta^*$.

EXAMPLE 5.7. *Take* Γ, R, *and the word* $w \in \Gamma^*$ *from Example 5.5. Then we have* $\mathcal{F}(w) = x\, y\, \overline{x}\, \overline{y}\, \overline{x}$.

LEMMA 5.8. *The following problem can be solved in polynomial time:*

INPUT: An SLP G *over the terminal alphabet* Γ *such that* $\rho(\mathrm{eval}(G)) \in D_1$ *and two positions* i *and* j *such that match*$(i) = j$ *in* $\mathrm{eval}(G)$.

OUTPUT: An SLP that generates $\mathcal{F}(\mathrm{eval}(G)[i+1, j-1])$.

Proof. Let $\Theta = \Delta \cup \{\langle, \rangle\}$ and consider the infinite semi-Thue system

$$T = \{\langle w \rangle \to \varepsilon \mid w \in \Delta^*\}$$

over the alphabet Θ. This system is clearly terminating and confluent (T has no overlapping left-hand sides), hence every word $w \in \Theta^*$ has a unique normal form $\mathrm{NF}_T(w)$. Note that $\mathrm{IRR}(T) = (\Delta^*\rangle\Delta^*)^*(\Delta^*\langle\Delta^*)^*$. Let $\mu : \Gamma \to \Theta$ be the morphism with $\mu(a) = \langle$ for all $a \in \Sigma_\ell$, $\mu(a) = \rangle$ for all $a \in \Sigma_r$, and $\mu(a) = a$ for all $a \in \Delta$.

In a first step we construct in polynomial time an SLP G' such that $\mathrm{eval}(G') = \mu(\mathrm{eval}(G)[i+1, j-1])$. Then $\mathrm{NF}_T(\mathrm{eval}(G')) = \mathcal{F}(\mathrm{eval}(G)[i+1, j-1])$. Hence, it suffices to calculate a composition system H that generates $\mathrm{NF}_T(\mathrm{eval}(G))$ for a given SLP G with $\mathrm{eval}(G) \in \Theta^*$; this composition system can be transformed in polynomial time into an equivalent SLP by Lemma 4.7. We construct H analogously to the proof of Theorem 4.5. Assume that $A \to BC$ is a production from G and assume that H already contains enough rules such that $\mathrm{eval}_H(X) = \mathrm{NF}_T(\mathrm{eval}_G(X)) =: w_X \in (\Delta^*\rangle\Delta^*)^*(\Delta^*\langle\Delta^*)^*$ for $X \in \{B, C\}$. We then calculate the numbers $n_B = |w_B|_\langle$ and $n_C = |w_C|_\rangle$, i.e., the number of opening (closing) brackets in w_B (w_C). Assume that $n_C \geq n_B$, the other case is analogous. Then we calculate the position i_B of the n_B-th opening bracket $\langle$ in w_B as well as the position i_C of the n_C-th closing bracket $\rangle$ in w_C. It follows that $\mathrm{NF}_T(\mathrm{eval}_G(A)) = w_B[1, i_B - 1] w_C[i_C + 1, |w_C|]$. Thus, we add to the current H the production $A \to B[1, i_B - 1] C[i_C + 1, |w_C|]$. ☐

Proof of Theorem 5.4. We use all notations from the previous discussion. The following statement was shown in [42]: For $w \in \Gamma^*$ it holds $w \overset{*}{\leftrightarrow}_R \varepsilon$ if and only if $\rho(w) \in D_1$, $\mathcal{F}(w) \overset{*}{\to}_S \varepsilon$, and for all $i, j \in \{1, \ldots, |w|\}$ with $\mathrm{match}(i) = j$ it holds $w[i]w[j] \in \mathrm{dom}(R)$ and $\mathcal{F}(w[i+1, j-1]) \overset{*}{\to}_S \varepsilon$.

This leads to the following NP-algorithm for testing $\mathrm{eval}(G) \overset{*}{\not\leftrightarrow}_R \varepsilon$ for a given SLP G, and hence to a coNP-algorithm for testing $\mathrm{eval}(G) \overset{*}{\leftrightarrow}_R \varepsilon$:

1. produce an SLP G' such that $\mathrm{eval}(G') = \rho(\mathrm{eval}(G))$ and check whether $\mathrm{eval}(G') \in D_1$. This is possible in polynomial time by Theorem 4.5. If $\mathrm{eval}(G') \notin D_1$ then accept, otherwise continue.
2. Guess a position $1 \leq i \leq |\mathrm{eval}(G)|$ (this is the only nondeterministic step) such that $\mathrm{eval}(G)[i] \in \Sigma_\ell$, calculate in polynomial time (by Lemma 5.6) the unique

position $j = \mathrm{match}(i)$ in $\mathrm{eval}(G)$, and check whether $\mathrm{eval}(G)[i]\mathrm{eval}(G)[j] \notin \mathrm{dom}(R)$. If this is true, then accept, otherwise continue.

3. Calculate in polynomial time (by Lemma 5.8) an SLP G'' that generates $\mathcal{F}(\mathrm{eval}(G)[i+1, j-1])$ and test in polynomial time (by Theorem 4.5) whether $\mathrm{eval}(G'') \overset{*}{\not\to}_S \varepsilon$. If this is true, then accept, otherwise reject.

This concludes the proof of Theorem 5.4. ☐

From Theorem 4.5, 5.2, and 5.4 we obtain the following corollary:

COROLLARY 5.9. *Let (Γ, R) be a 2-homogeneous and confluent presentation. Consider the following computational problem:*

INPUT: A word $s \in \Gamma^*$.
QUESTION: Does $w \overset{*}{\leftrightarrow}_R \varepsilon$ hold?

If (Γ, R) is N-free then this problem can be solved in polynomial time, otherwise this problem is coNP-complete.

6. Compressed word problems in PSPACE. In the previous two sections we have investigated 2-homogeneous systems, where every rule is of the form $ab \to \varepsilon$. In this section we consider a slightly more general class of presentations, where we also allow rules of the form $ab \to c$. We show that this generalization leads to PSPACE-complete compressed word problems. As a corollary we obtain a fixed deterministic context-free language with a PSPACE-complete compressed word problem, which solves an open problem from [23, 55].

Our PSPACE upper bounds rely all on the following simple fact:

PROPOSITION 6.1. *If the membership problem for the language L (the word problem for a finitely presented monoid $\mathcal{M}$) belongs to $\bigcup_{c>0} NSPACE(\log^c(n))$, then the compressed membership problem for L (the compressed word problem for $\mathcal{M}$) belongs to PSPACE.*

Proof. Assume that the language L belongs to $\mathrm{NSPACE}(\log^c(n))$. Let us fix an SLP G. We decide $\mathrm{eval}(G) \in L$ by simulating the $\mathrm{NSPACE}(\log^c(n))$ algorithm for the membership problem for L on words of length $|\mathrm{eval}(G)|$. Note that a number less than $|\mathrm{eval}(G)|$ can be stored in polynomial space and that for a given position $i \in \{1, \ldots, |\mathrm{eval}(G)|\}$ we can calculate $\mathrm{eval}(G)[i]$ in polynomial time. Thus, the simulation gives us a PSPACE-algorithm for the compressed membership problem for L. ☐

A presentation (Γ, R) is *weight-reducing* if there exists a weight-function $f : \Gamma^* \to \mathbb{N}$ such that $f(s) > f(t)$ for all $(s, t) \in R$. Typical examples of weight-reducing presentations are *length-reducing presentations* (i.e., $|s| > |t|$ for all $(s, t) \in R$).

PROPOSITION 6.2. *For every weight-reducing and confluent presentation (Γ, R), the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in PSPACE.*

Proof. In [41] we have shown that for every fixed weight-reducing and confluent presentation (Γ, R), the (uncompressed) word problem for $\mathcal{M}(\Gamma, R)$ is in LOGCFL, which is the logspace closure of the class of context-free languages [65]. The class LOGCFL is known to be contained in $\mathrm{NSPACE}(\log^2(n))$ [39]. Thus, membership in PSPACE follows from Proposition 6.1. ☐

In the rest of this section, we will show that PSPACE-hardness can be shown already for a quite small subclass of weight-reducing and confluent presentations.

A presentation (Γ, R) is called *monadic* if for every $(\ell, r) \in R$: $|\ell| > |r|$ and $|r| \leq 1$. A *2-monadic* presentation is a monadic presentation (Γ, R) such that moreover $|\ell| = 2$ for every $\ell \in \mathrm{dom}(R)$. In the following, we present a construction that reduces the reachability problem for directed forests to the (uncompressed) word problem of a fixed 2-monadic and confluent presentation (Γ, R). Later in this section, we will use this construction in order to prove that the compressed word problem for $\mathcal{M}(\Gamma, R)$ is PSPACE-complete.

Let $\Gamma = \{b_0, b_1, c_0, c_1, c_2, \#, \$, \triangleright, 0\}$ and let R be the 2-monadic semi-Thue system consisting of the following rules:

(1) $b_0 x \to \varepsilon$ for all $x \in \{\$, c_0, c_1, c_2\}$	(2) $b_1 c_0 \to \varepsilon$
(3) $b_1 \$ \to \triangleright$	(4) $\triangleright c_i \to \triangleright$ for all $i \in \{0, 1, 2\}$
(5) $\triangleright \$ \to \$$	(6) $\# \$ \to \varepsilon$
(7) $b_1 c_2 \to 0$	
(8) $0x \to 0$ for all $x \in \Gamma$	(9) $x0 \to 0$ for all $x \in \Gamma$

Only the rules involving the absorbing symbol 0 produce overlappings. In the resulting critical pairs, both words can be reduced to 0. Thus, R is confluent.

A *directed forest* is a directed acyclic graph (V, E) (where V is the finite set of nodes and $E \subseteq V \times V$ is the edge relation) such that moreover for every $u \in V$, $|\{v \in V \mid (u, v) \in E\}| \leq 1$, i.e., every node has at most one outgoing edge. Assume now that (V, E) is a directed forest, where $V = \{v_1, \ldots, v_n\}$ and $(v_i, v_j) \in E$ implies $i < j$. Let $v_\alpha \in V$ be a distinguished start node ($1 \leq \alpha \leq n$) and $U \subseteq V$ be a set of final nodes such that every node in U has outdegree 0 (there may be also nodes in $V \setminus U$ without outgoing edges). For $i \leq j$ we define the interval $I_{i,j} = \{v_k \mid i \leq k \leq j\}$. Thus, $I_{1,n} = V$. If $i > j$ we set $I_{i,j} = \emptyset$. We will construct a word $w(v_\alpha, U) \in \Gamma^*$ such that $(v_\alpha, v_i) \in E^*$ for some $v_i \in U$ (i.e., there is a path from the start node to some final node) if and only if $w(v_\alpha, U) \overset{*}{\leftrightarrow}_R 0$, i.e., $w(v_\alpha, U) \overset{*}{\to}_R 0$. For every $i \in \{1, \ldots, n\}$ define the word δ_i as follows:

$$\delta_i = \begin{cases} c_0^{n-j+i+1} & \text{if } (v_i, v_j) \text{ is the unique outgoing edge at node } v_i \\ c_1 & \text{if } v_i \in V \setminus U \text{ and } v_i \text{ has no outgoing edge} \\ c_2 & \text{if } v_i \in U \text{ (and thus has no outgoing edge)} \end{cases}$$

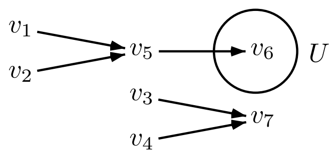
Note that $\triangleright \delta_i \overset{*}{\to}_R \triangleright$ for all $1 \leq i \leq n$ using the rules in (4). For an interval $I_{i,j}$ ($i \leq j$) we define $\sigma[I_{i,j}] = \delta_i \$ \delta_{i+1} \$ \cdots \delta_j \$$. We set $\sigma[\emptyset] = \varepsilon$. Note that $\triangleright \sigma[I_{i,j}] \overset{*}{\to}_R \$ \sigma[I_{i+1,j}]$ if $i \leq j$ using the rules in (4) and (5). Let $\beta = |\sigma[I_{1,\alpha-1}]|$. Finally, define

$$w(v_\alpha, U) = (\# b_1^n)^n b_0^\beta \sigma[I_{1,n}].$$

LEMMA 6.3. *We have $w(v_\alpha, U) \overset{*}{\leftrightarrow}_R 0$ if and only if $(v_\alpha, v_i) \in E^*$ for some $v_i \in U$.*

Before we prove Lemma 6.3 let us first consider an example:

EXAMPLE 6.4. *Let (V, E) be the following directed forest. The set U only contains the node v_6. Let $\alpha = 2$, i.e., v_2 is the start node.*



Then $w(v_\alpha, U) = (\#b_1^7)^7 b_0^5 c_0^4 \$ c_0^5 \$ c_0^4 \$ c_0^5 \$ c_0^7 \$ c_2 \$ c_1 \$$. *We obtain the following derivation:*

$$
\begin{array}{ll}
(\#b_1^7)^7 b_0^5 c_0^4 \$ c_0^5 \$ c_0^4 \$ c_0^5 \$ c_0^7 \$ c_2 \$ c_1 \$ \xrightarrow{*}_R & \textit{(rules in (1))} \\
(\#b_1^7)^7 c_0^5 \$ c_0^4 \$ c_0^5 \$ c_0^7 \$ c_2 \$ c_1 \$ \xrightarrow{*}_R & \textit{(rule (2))} \\
(\#b_1^7)^6 \# b_1^2 \$ c_0^4 \$ c_0^5 \$ c_0^7 \$ c_2 \$ c_1 \$ \to_R & \textit{(rule (3))} \\
(\#b_1^7)^6 \# b_1 \triangleright c_0^4 \$ c_0^5 \$ c_0^7 \$ c_2 \$ c_1 \$ \xrightarrow{*}_R & \textit{(rules in (4))} \\
(\#b_1^7)^6 \# b_1 \triangleright \$ c_0^5 \$ c_0^7 \$ c_2 \$ c_1 \$ \to_R & \textit{(rule (5))} \\
(\#b_1^7)^6 \# b_1 \$ c_0^5 \$ c_0^7 \$ c_2 \$ c_1 \$ \to_R & \textit{(rule (3))} \\
(\#b_1^7)^6 \# \triangleright c_0^5 \$ c_0^7 \$ c_2 \$ c_1 \$ \xrightarrow{*}_R & \textit{(rules in (4))} \\
(\#b_1^7)^6 \# \triangleright \$ c_0^7 \$ c_2 \$ c_1 \$ \to_R & \textit{(rule (5))} \\
(\#b_1^7)^6 \# \$ c_0^7 \$ c_2 \$ c_1 \$ \xrightarrow{*}_R & \textit{(rule (6))} \\
(\#b_1^7)^6 c_0^7 \$ c_2 \$ c_1 \$ \xrightarrow{*}_R & \textit{(rule (2))} \\
(\#b_1^7)^5 \# \$ c_2 \$ c_1 \$ \to_R & \textit{(rule (6))} \\
(\#b_1^7)^5 c_2 \$ c_1 \$ \xrightarrow{*}_R & \textit{(rule (7))} \\
(\#b_1^7)^4 \# b_1^6 0 \$ c_1 \$ \xrightarrow{*}_R & \textit{(rules in (8) and (9))} \\
0 &
\end{array}
$$

Indeed, there exists a path from v_2 *to a node in* U. *If* U *would only consist of the node* v_7 *instead of* v_6, *then* $w(v_\alpha, U) = (\#b_1^7)^7 b_0^5 c_0^4 \$ c_0^5 \$ c_0^4 \$ c_0^5 \$ c_0^7 \$ c_1 \$ c_2 \$$. *In this case we obtain a similar derivation showing* $w(v_\alpha, U) \xrightarrow{*}_R (\#b_1^7)^5 c_1 \$ c_2 \$$. *But the latter word is irreducible. Since* R *is confluent,* $w(v_\alpha, U) \xrightarrow{*}_R 0$ *cannot hold.*

Proof of Lemma 6.3. First note that using the rules in (1) we obtain

$$w(v_\alpha, U) = (\#b_1^n)^n b_0^\beta \sigma[I_{1,\alpha-1}] \sigma[I_{\alpha,n}] \xrightarrow{*}_R (\#b_1^n)^n \sigma[I_{\alpha,n}].$$

Claim: For every $v_i \in V$, if $(v_\alpha, v_i) \in E^*$, then there exists $k \geq n - i + \alpha$ such that $w(v_\alpha, U) \xrightarrow{*}_R (\#b_1^n)^k \sigma[I_{i,n}]$.

We prove this claim by induction over the length of the unique path from v_α to v_i. The case $i = \alpha$ is clear. Thus, assume that $(v_\alpha, v_j) \in E^*$ and $(v_j, v_i) \in E$. Then $j < i$ and by induction we have

$$w(v_\alpha, U) \xrightarrow{*}_R (\#b_1^n)^k \sigma[I_{j,n}] = (\#b_1^n)^{k-1} \# b_1^n c_0^{n-i+j+1} \$ \sigma[I_{j+1,n}],$$

where $k \geq n - j + \alpha$, i.e., $k - 1 \geq n - i + \alpha$. We obtain

$$
\begin{array}{ll}
(\#b_1^n)^{k-1} \# b_1^n c_0^{n-i+j+1} \$ \sigma[I_{j+1,n}] \xrightarrow{*}_R & \text{(rule (2))} \\
(\#b_1^n)^{k-1} \# b_1^{i-j-1} \$ \sigma[I_{j+1,n}] \to_R & \text{(rule (3))} \\
(\#b_1^n)^{k-1} \# b_1^{i-j-2} \triangleright \sigma[I_{j+1,n}] \xrightarrow{*}_R & \text{(rules in (4) and (5))} \\
(\#b_1^n)^{k-1} \# b_1^{i-j-2} \$ \sigma[I_{j+2,n}] \xrightarrow{*}_R & \\
\vdots & \\
(\#b_1^n)^{k-1} \# b_1^0 \$ \sigma[I_{i,n}] = & \\
(\#b_1^n)^{k-1} \# \$ \sigma[I_{i,n}] \to_R & \text{(rule (6))} \\
(\#b_1^n)^{k-1} \sigma[I_{i,n}]. &
\end{array}
$$

This proves the claim.

Thus, if $(v_\alpha, v_i) \in E^*$ for some $v_i \in U$, then by the above claim

$$\begin{aligned} w(v_\alpha, U) \ \xrightarrow{*}_R\ & (\# b_1^n)^k \sigma[I_{i,n}] = \\ & (\# b_1^n)^{k-1} \# b_1^{n-1} b_1 c_2 \$ \sigma[I_{i+1,n}] \to_R \qquad \text{(rule (7))} \\ & (\# b_1^n)^{k-1} \# b_1^{n-1} 0 \$ \sigma[I_{i+1,n}] \xrightarrow{*}_R 0 \qquad \text{(rules in (8) and (9))} \end{aligned}$$

for some $k > 0$. On the other hand, if there does not exist $v_i \in U$ with $(v_\alpha, v_i) \in E^*$, then there exists $v_i \in V \setminus U$ with outdegree 0 and $(v_\alpha, v_i) \in E^*$. Thus,

$$w(v_\alpha, U) \ \xrightarrow{*}_R\ (\# b_1^n)^k \sigma[I_{i,n}] = (\# b_1^n)^k c_1 \$ \sigma[I_{i+1,n}] \in \mathrm{IRR}(R).$$

Since (Γ, R) is confluent, $w(v_\alpha, U) \xrightarrow{*}_R 0$ cannot hold. This proves the Lemma 6.3. ☐

Lemma 6.3 yields the following result that is of independent interest; see Section 3.2 for the definition of NC^1-reductions.

THEOREM 6.5. *There exists a fixed 2-monadic and confluent presentation* (Γ, R) *such that the word problem for* $\mathcal{M}(\Gamma, R)$ *is L-hard under* NC^1*-reductions.*

Proof. By [18], the reachability problem for directed forests that are ordered (i.e., the set of nodes is $\{1, \ldots, n\}$ for some n and $i < j$ whenever there is an edge from i to j) is L-complete under NC^1-reductions. Moreover, one can assume that 1 is the initial node. It remains to show that for such a forest $G = (V, E)$ and $U \subseteq V$ the word $w(1, U)$ can be constructed in NC^1 from G and U. We leave the details to the reader. ☐

The existence of a fixed monadic and confluent presentation with an L-hard word problem was also shown in [6].

Now, let us consider the compressed word problem for 2-monadic and confluent presentations.

THEOREM 6.6. *For every 2-monadic and confluent presentation* (Γ, R)*, the compressed word problem for* $\mathcal{M}(\Gamma, R)$ *is in PSPACE. There exists a fixed 2-monadic and confluent presentation* (Γ, R) *such that the compressed word problem for* $\mathcal{M}(\Gamma, R)$ *is PSPACE-complete.*

Proof. The upper bound follows from Proposition 6.2. For the lower bound we will show that the compressed word problem for the 2-monadic presentation (Γ, R) from the previous discussion is PSPACE-hard. For this we repeat a construction from [41]. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, q_f)$ be a fixed deterministic linear bounded automaton (Q is the set of states, Σ is the tape alphabet, q_0 (resp. q_f) is the initial (resp. final) state, and $\delta : Q \setminus \{q_f\} \times \Sigma \to Q \times \Sigma \times \{\mathrm{left}, \mathrm{right}\}$ is the transition function) such that the question whether a word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ is PSPACE-complete. Such a linear bounded automaton exists, see, e.g., [5]. The one-step transition relation between configurations of $\mathcal{A}$ is denoted by $\Rightarrow_{\mathcal{A}}$. Let $w \in \Sigma^*$ be an input for $\mathcal{A}$ with $|w| = N$. We may assume that $\mathcal{A}$ operates in phases, where a single phase consists of a sequence of $2 \cdot N$ transitions of the form $q_1 \gamma_1 \xRightarrow{*}_{\mathcal{A}} \gamma_2 q_2 \xRightarrow{*}_{\mathcal{A}} q_3 \gamma_3$, where $\gamma_1, \gamma_2, \gamma_3 \in \Sigma^N$ and $q_1, q_2, q_3 \in Q$. During the transition sequence $q_1 \gamma_1 \xRightarrow{*}_{\mathcal{A}} \gamma_2 q_2$ only right-moves are made, whereas during the sequence $\gamma_2 q_2 \xRightarrow{*}_{\mathcal{A}} q_3 \gamma_3$ only left-moves are made. The automaton $\mathcal{A}$ accepts, if it reaches the final state q_f. Otherwise $\mathcal{A}$ does not terminate. There exists a constant $c > 0$ such that if w is accepted by $\mathcal{A}$, then $\mathcal{A}$, started on w, reaches the final state q_f after at most $2^{c \cdot N}$ phases. Let $\widehat{\Sigma} = \{\widehat{a} \mid a \in \Sigma\}$ be a disjoint copy of Σ and similarly for $\widehat{Q}$. Let $\Delta = \Sigma \cup \widehat{\Sigma} \cup \{\triangleleft, 0, 1, \pounds\}$ and $\Theta = Q \cup \widehat{Q} \cup \Delta$. We simulate $\mathcal{A}$ by the following semi-Thue system S over the alphabet Θ:

$0\widehat{q} \to \widehat{q}\pounds$	for all $q \in Q \setminus \{q_f\}$	$qa \to \widehat{b}p$	if $\delta(q,a) = (p,b,\text{right})$
$1\widehat{q} \to 0q$	for all $q \in Q \setminus \{q_f\}$	$\widehat{a}\,\widehat{q} \to \widehat{p}b$	if $\delta(q,a) = (p,b,\text{left})$
$q\pounds \to 1q$	for all $q \in Q \setminus \{q_f\}$	$q\lhd \to \widehat{q}\lhd$	for all $q \in Q \setminus \{q_f\}$

(Θ, S) is length-preserving and if $\succ$ is any linear order on the alphabet Θ that satisfies $Q \succ 1 \succ 0 \succ \widehat{\Sigma} \succ \widehat{Q}$, then $s \succ t$ for every rule $(s,t) \in S$, i.e., S is lexicographic (recall from Section 3.1 that we identify an order $\succ$ on the alphabet Θ with its lexicographic extension to Θ^*. Let us choose such a linear order on Θ that moreover satisfies $Q \succ \Delta \succ \widehat{Q}$. In [41] we have argued that w is accepted by $\mathcal{A}$ if and only if $1q_0\pounds^{c\cdot N-1}w\lhd \xrightarrow{*}_S v$ for some word v with $\mathrm{alph}(v) \cap \{q_f, \widehat{q}_f\} \neq \emptyset$. We briefly repeat the arguments: First, note that $1q_0\pounds^{c\cdot N-1}w\lhd \xrightarrow{*}_S 1^{c\cdot N}q_0w\lhd$. From the word $1^{c\cdot N}q_0w\lhd$ we can simulate $2^{c\cdot N}$ phases of $\mathcal{A}$. The crucial point is that the prefix from $\{0,1\}^*$ acts as a binary counter: for every $u \in \{0,1\}^i$ $(i < c\cdot N)$ and every $q \in Q$ we have: $u10^{c\cdot N-|u|-1}\widehat{q} \xrightarrow{*}_S u1\widehat{q}\pounds^{c\cdot N-|u|-1} \to_S u0q\pounds^{c\cdot N-|u|-1} \xrightarrow{*}_S u01^{c\cdot N-|u|-1}q$. Thus, if $\mathcal{A}$ accepts w, then we can derive from $1q_0\pounds^{c\cdot N-1}w\lhd$ a word v with $\mathrm{alph}(v)\cap\{q_f,\widehat{q}_f\} \neq \emptyset$. On the other hand, if $\mathcal{A}$ does not accept w and hence does not terminate, then we can derive from $1q_0\pounds^{c\cdot N-1}w\lhd$ a word of the form $\widehat{q}\pounds^{c\cdot N}u\lhd$ for some $u \in \Sigma^N$ and $q \neq q_f$. Since S is confluent (the left-hand sides of S do not overlap) and $\widehat{q}\pounds^{c\cdot N}u\lhd \in \mathrm{IRR}(S)$, we cannot reach a word v with $\mathrm{alph}(v) \cap \{q_f, \widehat{q}_f\} \neq \emptyset$ from $1q_0\pounds^{c\cdot N-1}w\lhd$.

To simplify the following construction, we will next expand all rules from S in the following sense: The rule $q\pounds \to 1q$ for instance is replaced by all rules of the form $xq\pounds \to x1q$ for all $x \in \Delta$, whereas the rule $0\widehat{q} \to \widehat{q}\pounds$ is replaced by all rules of the form $0\widehat{q}x \to \widehat{q}\pounds x$ for all $x \in \Delta$. Let us call the resulting system again S. Then S is still length-preserving and lexicographic and $\mathrm{dom}(S) \subseteq \Delta(Q \cup \widehat{Q})\Delta$. The new system S is no longer confluent, but this is not important for the further arguments. It is only important that

$$\forall v, v_1, v_2 \in \Delta^*(Q \cup \widehat{Q})\Delta^* : (v_1 \,{}_S\!\leftarrow v \to_S v_2) \Rightarrow v_1 = v_2. \tag{6.1}$$

This is easy to see by inspection of the rules. Moreover, w is accepted by $\mathcal{A}$ if and only if $1q_0\pounds^{c\cdot N-1}w\lhd \xrightarrow{*}_S v$ for some word v with $\mathrm{alph}(v) \cap \{q_f, \widehat{q}_f\}$. Let $m = (c+1)N - 1$; thus $m+3$ is the length of words in any derivation starting from $1q_0\pounds^{c\cdot N-1}w\lhd$.

Let us now define the directed graph (V,E), where $V = \bigcup_{i=1}^{m+1} \Delta^i(Q \cup \widehat{Q})\Delta^{m-i+2}$ and $E = \{(v,v') \in V \times V \mid v \to_S v'\}$. This graph is basically the transition graph of the automaton $\mathcal{A}$ on configurations of length N. Since S is lexicographic, (V,E) is acyclic. Moreover, by (6.1), every node from V has at most one outgoing edge. Thus, (V,E) is a directed forest. If we order V lexicographically by $\succ$ and write $V = \{v_1, \ldots, v_n\}$ with $v_1 \succ v_2 \succ \cdots \succ v_n$, then $(v_i, v_j) \in E$ implies $i < j$. Note that $n = |V| = 2(m+1)\cdot|Q|\cdot|\Delta|^{m+2}$, which belongs to $2^{O(N)}$. Let U be those words in V that contain either q_f or $\widehat{q}_f$; these words have outdegree 0 in the directed forest (V,E). Let $v_\alpha = 1q_0\pounds^{c\cdot N-1}w\lhd$. Thus, $\alpha - 1$ is the number of words from V that are lexicographically larger than $1q_0\pounds^{c\cdot N-1}w\lhd$. The number α can be easily calculated in logarithmic space from the input word w using simple arithmetic. We now have available all the data in order to construct the word $w(v_\alpha, U) \in \Gamma^*$ from Lemma 6.3.

The automaton $\mathcal{A}$ accepts w if and only if there is a path in (V,E) from v_α to a node in U. By Lemma 6.3 this holds if and only if $w(v_\alpha, U) \xleftrightarrow{*}_R 0$. Thus, it remains to show that the word $w(v_\alpha, U)$ can be generated by a small SLP. Recall the definition of the words σ_i and $\sigma[I] \in \Gamma^*$, where $1 \le i \le n = |V|$ and I is an interval of $(V, \succ)$ that we introduced before Lemma 6.3.

Note that for all $1 \le i, j \le n$, if $v_i = u_1\ell u_2 \to_S u_1ru_2 = v_j$ with $(\ell, r) \in S$, then

$j - i$ (i.e., the number of words from V that are lexicographically between v_i and v_j) is a number that only depends on the rule (ℓ, r) (and thus ℓ) and $|u_2|$. We call this number $\lambda(\ell, |u_2|)$; it belongs to $2^{O(N)}$ and can be calculated in logarithmic space from ℓ and $|u_2|$ using simple arithmetic. We now describe a small SLP that generates the word $\sigma[V] \in \Gamma^*$. For this let us assume that $Q = \{p_1, \ldots, p_{n_1}\}$ and $\Delta = \{a_1, \ldots, a_{n_2}\}$ with $p_i \succ p_{i+1}$, $\widehat{p}_i \succ \widehat{p}_{i+1}$, and $a_i \succ a_{i+1}$ (note that the order $\succ$ on the subalphabets Q, $\widehat{Q}$, and Δ, respectively, is arbitrary except that $1 \succ 0$). We introduce the following productions:[4]

$$
\begin{aligned}
A_i &\to \prod_{j=1}^{n_2} B_{i,j} A_{i+1} \widehat{B}_{i,j} \text{ for } 0 \le i < m \\
A_m &\to \prod_{j=1}^{n_2} B_{m,j} \widehat{B}_{m,j} \\
B_{i,j} &\to \prod_{k=1}^{n_1} \prod_{\ell=1}^{n_2} (C_{i,j,k,\ell}\$)^{|\Delta|^{m-i}} \quad \text{for } 0 \le i \le m,\ 1 \le j \le n_2 \\
C_{i,j,k,\ell} &\to \begin{cases} c_0^{n-\lambda(a_j p_k a_\ell, m-i)+1} & \text{if } a_j p_k a_\ell \in \mathrm{dom}(R) \\ c_1 & \text{if } a_j p_k a_\ell \notin \mathrm{dom}(R) \text{ and } p_k \neq q_f \\ c_2 & \text{if } p_k = q_f \end{cases} \\
\widehat{B}_{i,j} &\to \prod_{k=1}^{n_1} \prod_{\ell=1}^{n_2} (\widehat{C}_{i,j,k,\ell}\$)^{|\Delta|^{m-i}} \quad \text{for } 0 \le i \le m,\ 1 \le j \le n_2 \\
\widehat{C}_{i,j,k,\ell} &\to \begin{cases} c_0^{n-\lambda(a_j \widehat{p}_k a_\ell, m-i)+1} & \text{if } a_j \widehat{p}_k a_\ell \in \mathrm{dom}(R) \\ c_1 & \text{if } a_j \widehat{p}_k a_\ell \notin \mathrm{dom}(R) \text{ and } \widehat{p}_k \neq \widehat{q}_f \\ c_2 & \text{if } \widehat{p}_k = \widehat{q}_f \end{cases}
\end{aligned}
$$

The exponents that appear in the right-hand sides of the productions for the nonterminal $B_{i,j}$ and $\widehat{B}_{i,j}$, namely $|\Delta|^{m-i}$, are of size $2^{O(N)}$ and can therefore be replaced by sequences of ordinary productions. Note that $\mathrm{eval}(C_{i,j,k,\ell}) = \delta_s$ for every node v_s from $\Delta^i a_j p_k a_\ell \Delta^{m-i}$, whereas $\mathrm{eval}(\widehat{C}_{i,j,k,\ell}) = \delta_s$ for every node v_s from $\Delta^i a_j \widehat{p}_k a_\ell \Delta^{m-i}$. It follows that for all $0 \le i \le m$, all $u \in \Delta^i$, and all $1 \le j \le n_2$ we have (note that $u a_j Q \Delta^{m-i+1} \subseteq V$ is an interval of $(V, \succ)$)

$$
\mathrm{eval}(B_{i,j}) = \sigma[u a_j Q \Delta^{m-i+1}] \quad \text{and} \quad \mathrm{eval}(\widehat{B}_{i,j}) = \sigma[u a_j \widehat{Q} \Delta^{m-i+1}]. \tag{6.2}
$$

Claim. Let $0 \le i \le m$ and let $u \in \Delta^i$ be arbitrary. Then for the interval $I = \bigcup_{j=1}^{m-i+1} u \Delta^j (Q \cup \widehat{Q}) \Delta^{m-i-j+2}$ of the linear order $(V, \succ)$ we have $\sigma[I] = \mathrm{eval}(A_i)$.

The claim will be shown by induction on i (for $i = m$ down to 0). For $i = m$ the claim is true:

$$
\begin{aligned}
\mathrm{eval}(A_m) = \prod_{j=1}^{n_2} \mathrm{eval}(B_{m,j}) \mathrm{eval}(\widehat{B}_{m,j}) = &\qquad \text{(by (6.2))} \\
\prod_{j=1}^{n_2} \sigma[u a_j Q \Delta] \sigma[u a_j \widehat{Q} \Delta] = &\qquad (Q \succ \widehat{Q}) \\
\sigma[u \Delta (Q \cup \widehat{Q}) \Delta] &
\end{aligned}
$$

[4]The expression $\prod_{i=1}^{k} w_i$ is an abbreviation for $w_1 w_2 \cdots w_k$.

Now let $i < m$ and $u \in \Delta^i$. The words from the interval $\bigcup_{j=1}^{m-i+1} u\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+2}$ can be partitioned into the following decreasing sequence of consecutive intervals of $(V, \succ)$ (recall that $Q \succ \Delta \succ \widehat{Q}$ and $a_1 \succ a_2 \succ \cdots \succ a_{n_2}$):

$$\begin{array}{lll} ua_1 Q\Delta^{m-i+1} & \succ \bigcup_{j=1}^{m-i} ua_1\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1} & \succ ua_1\widehat{Q}\Delta^{m-i+1} \succ \\ ua_2 Q\Delta^{m-i+1} & \succ \bigcup_{j=1}^{m-i} ua_2\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1} & \succ ua_2\widehat{Q}\Delta^{m-i+1} \succ \cdots \succ \\ ua_{n_2} Q\Delta^{m-i+1} & \succ \bigcup_{j=1}^{m-i} ua_{n_2}\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1} & \succ ua_{n_2}\widehat{Q}\Delta^{m-i+1}. \end{array}$$

By induction, we have

$$\begin{aligned} &\sigma[\bigcup_{j=1}^{m-i} ua_1\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1}] = \cdots = \\ &\sigma[\bigcup_{j=1}^{m-i} ua_{n_2}\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1}] = \mathrm{eval}(A_{i+1}). \end{aligned}$$

Together with (6.2) we obtain

$$\sigma[\bigcup_{j=1}^{m-i+1} u\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+2}] = \prod_{j=1}^{n_2} \mathrm{eval}(B_{i,j})\mathrm{eval}(A_{i+1})\mathrm{eval}(\widehat{B}_{i,j}) = \mathrm{eval}(A_i).$$

This proves the claim. By setting $i = 0$ we get

$$\mathrm{eval}(A_0) = \sigma[\bigcup_{j=1}^{m+1} \Delta^j(Q \cup \widehat{Q})\Delta^{m-j+2}] = \sigma[V].$$

Let $\beta = |\sigma[I_{1,\alpha-1}]| \in 2^{O(N)}$. Arithmetic on numbers with $N^{O(1)}$ many bits allows to compute β in logspace from the input word w. Using the above productions and the number β, we can construct an SLP G of size polynomial in the input size N with $\mathrm{eval}(G) = (\#b_1^n)^n b_0^\beta \sigma[V] = w(v_\alpha, U)$. Recall that n is of size $2^{O(N)}$. Then our input word w is accepted by $\mathcal{A}$ if and only if $\mathrm{eval}(G) \xleftrightarrow{*}_R 0$. This proves the theorem. ∎

The following corollary solves an open problem from [23, 55].

Corollary 6.7. *There exists a fixed deterministic context-free language L such that the compressed membership problem for L is PSPACE-complete.*

Proof. Since every context-free language is contained in DSPACE($\log^2(n)$), the PSPACE upper bound can be deduced from Proposition 6.1. For the lower bound, notice that the language $\{w \in \Gamma^* \mid w \xrightarrow{*}_R 0\}$ is deterministic context-free for every monadic and confluent presentation and every $0 \in \Gamma$, see e.g. [11, Theorem 4.2.7]. If we choose the 2-monadic and confluent presentation from the proof of Theorem 6.5 for (Γ, R), then the language $\{w \in \Gamma^* \mid w \xrightarrow{*}_R 0\}$ is PSPACE-hard by the proof of Theorem 6.6. ∎

In [31] a language L is called *deterministic linear* if it is accepted by a deterministic 1-turn pushdown automaton.

COROLLARY 6.8. *There exists a fixed deterministic linear language L such that the compressed membership problem for L is PSPACE-complete.*

Proof. The language $\{w \in \Gamma^* \mid w \xrightarrow{*}_R 0\} \cap (\#b_1^+)^+ b_0^+((c_0^+ \cup c_1 \cup c_2)\$)^+$ is easily seen to be deterministic linear. Moreover, it contains a word of the form $w(v_\alpha, U)$ if and only if $w(v_\alpha, U) \xrightarrow{*}_R 0$. ☐

7. Compressed word problems in EXPSPACE. The largest class of monoid presentations that we consider in this paper are weight-lexicographic and confluent presentations: A presentation (Γ, R) is *weight-lexicographic* if there exist a linear order $\succ$ on the alphabet Γ and a weight-function $f : \Gamma^* \to \mathbb{N}$ such that for all $(s,t) \in R$ we have either $f(s) > f(t)$ or ($f(s) = f(t)$ and $s \succ t$). If the weight-function f is the length-function, i.e., $f(w) = |w|$, then (Γ, R) is called *length-lexicographic*.

THEOREM 7.1. *For every weight-lexicographic and confluent presentation (Γ, R), the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in EXPSPACE. Moreover, there exists a fixed length-lexicographic and confluent presentation (Γ, R) such that the compressed word problem for $\mathcal{M}(\Gamma, R)$ is EXPSPACE-complete.*

Proof. For the upper bound we can use a result from [41]: for every fixed weight-lexicographic and confluent presentation (Γ, R), the word problem for $\mathcal{M}(\Gamma, R)$ belongs to PSPACE. Thus, for two given SLPs G_1 and G_2 we can first generate the exponentially long words $\mathrm{eval}(G_1)$ and $\mathrm{eval}(G_2)$ and then check in space bounded polynomially in $|\mathrm{eval}(G_1)| + |\mathrm{eval}(G_1)|$ whether $\mathrm{eval}(G_1) \xleftrightarrow{*}_R \mathrm{eval}(G_2)$.

For the lower bound, let (Θ, S) be the presentation from the proof of Theorem 6.6, where this time, the simulated machine $\mathcal{A}$ is a fixed Turing-machine that accepts an EXPSPACE-complete language. Also for such a machine we may assume that it operates in alternating phases of left and right sweeps. The presentation (Θ, S) is length-lexicographic and confluent. W.l.o.g. the space bound for an input of length n is 2^n. The number of phases can be bounded by $2^{c \cdot 2^n}$ for some constant $c > 0$. Add to Θ an absorbing symbol 0 and add to S the following rules: $q_f \to 0$, $\widehat{q}_f \to 0$, $x0 \to 0$ for all $x \in \Theta$, and $0x \to 0$ for all $x \in \Theta$. We call the resulting presentation again (Θ, S); it is still length-lexicographic and confluent. Moreover, for an input word w of length n, w is accepted by $\mathcal{A}$ if and only if $1q_0\pounds^{c \cdot 2^n - 1} w \Box^{2^n - |w|} \triangleleft \xleftrightarrow{*}_S 0$ (where q_0 is the initial state of $\mathcal{A}$ and $\Box$ is the blank symbol), see also the proof of Theorem 6 in [41]. Finally, note that the word $1q_0\pounds^{c \cdot 2^n - 1} w \Box^{2^n - |w|} \triangleleft$ can be generated by an SLP of polynomial size in n. ☐

The language $\{w \in \Theta^* \mid w \xrightarrow{*}_S 0\}$, where (Θ, S) is the presentation from the previous proof is context-sensitive. Thus, we obtain the following result:

COROLLARY 7.2. *There exists a fixed context-sensitive language L such that the compressed membership problem for L is EXPSPACE-complete.*

8. Circuit complexity and compression. In this section we will investigate the compressed membership problem for languages from very low complexity classes. These classes are usually defined by uniform families of small depth Boolean circuit families. An equivalent and for our purpose more suitable definition is based on alternating Turing-machines with logarithmic time bounds, see Section 3.2. Recall that ALOGTIME denotes the class of all languages that can be recognized on an alternating Turing-machine in time $O(\log(n))$. Within ALOGTIME, we can define the logtime hierarchy: For $k \geq 1$ we denote by $\Sigma_k^{\log}$ (resp. $\Pi_k^{\log}$) the class of all languages that can be decided by an alternating Turing-machine in time $O(\log(n))$ within $k-1$ alternations (on every computation path) starting in an existential state (resp. universal state). In [4], $\Sigma_k^{\log} \cup \Pi_k^{\log}$ is proposed as a uniform version of AC_k^0, which is the class of all languages that can be recognized by a polynomial size, depth k family of unbounded fan-in Boolean circuits. The union $\bigcup_{k\geq 0}\Sigma_k^{\log}\cup\Pi_k^{\log}$ is also called the *logtime hierarchy* LH. By [63], LH is a strict hierarchy.

The well-known *polynomial time hierarchy* [64] is defined similarly to the logtime hierarchy: For $k\geq 1$ we denote by Σ_k^p (resp. Π_k^p) the class of all languages that can be decided by an alternating Turing-machine in polynomial time within $k-1$ alternations (on every computation path) starting in an existential state (resp. universal state).

For the further investigations we will need the following lemma.

LEMMA 8.1. *Incrementing a binary n-bit counter C (with arbitrary initial value) m times by 1 takes at total $O(n+m)$ steps on a deterministic Turing machine.*

Proof. W.l.o.g. assume that m is a power of 2. We assume that after each increment, the read-write head moves back to the least significant bit of C. If the first i bits of C are $1^{i-1}0$ in that order, then we need $2i$ steps for the increment. For $i>\log(m)$, this will happen only once during the m increments. Thus, it suffices to show that incrementing a binary $\log(m)$-bit counter m times by 1 takes at total $O(m)$ steps. There are at most $2^{\log(m)-i}=\frac{m}{2^i}$ numbers $t\in\{0,\ldots,m\}$ such that the first i bits of t are $1^{i-1}0$ in that order. Incrementing such a number by 1 takes $2i$ steps. Thus, incrementing the counter m times by 1 takes at total $\sum_{i=1}^{\log(m)}2i\cdot\frac{m}{2^i}\leq 2m\sum_{i=1}^{\infty}\frac{i}{2^i}=4m$ steps. ☐

Of course, an analogous statement for decrementing a counter is also true.

THEOREM 8.2. *For every language L in $\Sigma_k^{\log}$ (resp. $\Pi_k^{\log}$) the compressed membership problem belongs to Σ_k^p (resp. Π_k^p). Moreover, there exists a fixed language L in $\Sigma_k^{\log}$ (resp. $\Pi_k^{\log}$) such that the compressed membership problem for L is Σ_k^p-complete (resp. Π_k^p-complete).*

Proof. It suffices to prove the statement for Σ_k^p and $\Sigma_k^{\log}$, respectively, because Π_k^p and $\Pi_k^{\log}$ are the corresponding complementary classes. Let us first show that the compressed membership problem for L belongs to Σ_k^p in case L belongs to $\Sigma_k^{\log}$. Assume that L belongs to $\Sigma_k^{\log}$. Given a straight-line program G we simulate the $\Sigma_k^{\log}$-algorithm on $\mathrm{eval}(G)$. Since $|\mathrm{eval}(G)|\in 2^{O(n)}$ this simulation takes $O(n)$ steps. Moreover, the number of alternations during the simulation is k and we start in an existential state. Finally note that if the $\Sigma_k^{\log}$-machine has written a position $i\in\{1,\ldots,|\mathrm{eval}(G)|\}$ on its address-tape and queries the i-th position of $\mathrm{eval}(G)$, then in the simulation we have to determine the symbol $\mathrm{eval}(G)[i]$ which is possible in polynomial time (with respect to $|G|$).

Next we will construct a language L in $\Sigma_k^{\log}$ such that the compressed membership problem for L is Σ_k^p-complete. First assume that k is odd. In this case the following restricted version of QBF (quantified Boolean satisfiability), called $\mathrm{Q3SAT}_k$, is Σ_k^p-complete [70]:

INPUT: A quantified Boolean formula of the form

$$\Theta=\exists\overline{x}_1\forall\overline{x}_2\cdots\exists\overline{x}_k:\varphi(x_1,\ldots,x_n).$$

Here $\overline{x}_i=(x_{\ell_i},\cdots,x_{\ell_{i+1}-1})$ (where $\ell_1=1$ and $\ell_{k+1}=n+1$) is a sequence of Boolean variables and φ is a formula in 3-CNF (a conjunction of disjunctions, each containing exactly three literals) over the variables $x_1,\ldots,x_n$.

QUESTION: Is Θ a true quantified Boolean formula?

Let us take an instance Θ of $\mathrm{Q3SAT}_k$ of the above form. Assume that $\varphi=C_1\wedge C_2\wedge\cdots\wedge C_m$ where every C_i is a disjunction of three literals. Note that there are 2^n truth

input a string $w \in \Gamma^*$
$t := 2\lceil \log(|w|) \rceil$ (by [3, Lemma 6.1], t can be calculated in DLOGTIME)
$s := |w| - 1$ (again, by [3, Lemma 6.1], s can be calculated in DLOGTIME)
$p := 0$; $n := 0$;
for $i = 1$ **to** k **do**
    while $w[s] = a_i$ and $t > 0$ **do**
        $t := t - 1$; $s := s - 1$; $n := n + 1$;
        Guess existentially (if i is odd) or universally (if i is even) the n-th bit of the number p.
    endwhile
endfor
Copy the n-bit number p to the address tape, initialize a counter q to 0, and append q to the n bits of the address tape. When querying the input tape via the address tape, the machine $\mathcal{A}$ will interpret the content of the address tape (i.e., the concatenation of the n bits of p followed by the bits of q) as the number $p + 2^n \cdot q$.
while $w[p + 2^n \cdot q] = 1$ and $t > 0$ **do**
    $q := q + 1$; $t := t - 1$;
endwhile
if $w[p + 2^n \cdot q] = 0$ **then** reject **else** accept

FIG. 8.1. *The alternating Turing-machine $\mathcal{A}$*

assignments to the variables $x_1, \dots, x_n$ and each of these truth assignments can be represented by the vector $(b_1, \dots, b_n)$ where $b_i = 1$ if the variable x_i evaluates to true, otherwise $b_i = 0$. We order these vectors lexicographically, where the last position gets the highest significance, i.e., $(0, \dots, 0) < (1, 0, \dots, 0) < \cdots < (1, \dots, 1)$. Thus, we can speak of the j-th truth assignment ($0 \le j \le 2^n - 1$). For each disjunction C_i define the word $c_i = b_0 b_1 \cdots b_{2^n-1}$, where $b_j \in \{0, 1\}$ is the truth value of C_i under the j-th truth assignment. In [8] it is shown that the word c_i can be generated by an SLP of size $O(n)$. Let $d_i = \ell_{i+1} - \ell_i$, i.e., d_i is the number of variables in the i-th block $\overline{x}_i$. Let $\Gamma = \{0, 1, \$, a_1, \dots, a_k\}$ Finally, we define the word $w(\Theta) \in \Gamma^*$ by

$$w(\Theta) = c_1 c_2 \cdots c_m \$^{2^m} a_k^{d_k} \cdots a_2^{d_2} a_1^{d_1}.$$

Since every c_i can be generated by an SLP of size $O(n)$, the word $w(\Theta)$ can be generated by an SLP of polynomial size with respect to the size of the formula Θ. Note that $|w(\Theta)| = m \cdot 2^n + 2^m + n$. Thus $n \le \log(|w(\Theta)|)$ and also $m \le \log(|w(\Theta)|)$. The only use of the padding-factor $\$^{2^m}$ is to ensure $m \le \log(|w(\Theta)|)$. It remains to construct a $\Sigma_k^{\log}$-machine $\mathcal{A}$ that accepts a given input word of the form $w(\Theta)$ if and only if Θ is true. The behavior of this machine on inputs that are not of the form $w(\Theta)$ is not important; it is only important that the logarithmic time bound is respected, independently of the form of the input. This will be ensured by a counter t. In the following, we write $w[i]$ ($i \in \{0, \dots, |w| - 1\}$), where is the current content of the address tape, for the result of querying the input w via the random access mechanism. Note that in contrast to the definition in Section 3.1, we number the first position of w with 0 and the last position with $|w| - 1$. The alternating Turing-machine $\mathcal{A}$ is described in Figure 8.1. Let us consider an example before we continue with analyzing the machine $\mathcal{A}$.

Example 8.3. *Let* Θ *be the quantified Boolean formula*

$$\exists x_1 \forall x_2 \exists x_3 : (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3),$$

which evaluates to true. We have $c_1 = 11110111$ *and* $c_2 = 10111111$ *and thus*

$$w(\Theta) = 1111011110111111\$^4 a_3 a_2 a_1.$$

Consider the truth assignment $x_1 = x_2 = 1$, $x_3 = 0$. *This is the third assignment in lexicographic order (recall that we start with the* 0*-th assignment). The fact that* $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$ *is true under this assignment corresponds to the fact that* $w(\Theta)[3] = 1$ *and* $w(\Theta)[3 + 2^n] = w(\Theta)[11] = 1$. *This is verified by the machine* $\mathcal{A}$ *in the second while-loop, when* $p = 3$ *is guessed in the for-loop.*

Now let us analyze the behavior of the alternating Turing-machine $\mathcal{A}$ on an input $w \in \Gamma^*$. All counters (t, s, n, and q) in this algorithm only need $\log(|w|)$ bits and are incremented (or decremented) only $O(\log(|w|))$ times. Thus, by Lemma 8.1 all increments and decrements for the various counters need $O(\log(|w|))$ total time. Hence, the counter t enforces a logarithmic time bound of the whole algorithm. Since the number of alternations is precisely k and $\mathcal{A}$ starts in an existential state, the machine $\mathcal{A}$ is indeed a $\Sigma_k^{\log}$-machine. We claim that if the input w is of the form $w(\Theta)$ for an instance Θ of $\mathrm{Q3SAT}_k$, then the counter t does not reach the value 0 if $\mathcal{A}$ is started on $w(\Theta)$. Let us fix an instance Θ of $\mathrm{Q3SAT}_k$ with n variables and m clauses. Thus, $n \leq \log(|w(\Theta)|)$ and $m \leq \log(|w(\Theta)|)$. From the construction of $w(\Theta)$ it follows that $\mathcal{A}$ decrements the counter t only $n + m \leq 2\lceil \log(|w(\Theta)|) \rceil$ times when $\mathcal{A}$ runs on the input $w(\Theta)$. Thus, t does not reach the value 0. Using this observation, it is easy to see that $\mathcal{A}$ accepts $w(\Theta)$ if and only if Θ is true. This finishes the presentation of a language in $\Sigma_k^{\log}$ with a Σ_k^p-complete compressed membership problem for the case that k is odd. If k is even, one can argue analogously, one only has to replace the 3-CNF formula by a 3-DNF formula. The resulting variant of $\mathrm{Q3SAT}_k$ is Σ_k^p-complete [70]. ◻

Let us remark that the logtime hierarchy LH can be also characterized using first-order logic (FO) with ordering and the BIT predicate: LH = FO[<, BIT], see, e.g., [32] for definitions. Since for instance NP can be captured by existential second-order logic (NP = SO∃), it follows from Theorem 8.2 that FO[<, BIT] properties on strings cannot be translated into SO∃ properties on straight-line programs unless the polynomial time hierarchy collapses. In a setting without the BIT predicate similar definability issues are investigated in [1].

By Proposition 6.1, for every language in $\bigcup_{i \geq 0} \mathrm{NSPACE}(\log^i(n))$ the compressed membership problem belongs to PSPACE. It turns out that we find languages with a PSPACE-complete compressed membership problem already in ALOGTIME $\subseteq$ DSPACE($\log(n)$):

Theorem 8.4. *There exists a fixed language* L *in ALOGTIME such that the compressed membership problem for* L *is PSPACE-complete.*

Proof. We can reuse the construction from the previous proof, except that we start from an instance of QBF (quantified Boolean satisfiability) which is PSPACE-complete [52]. ◻

One should note that it is *not* the case that for every ALOGTIME-complete language the compressed membership problem is PSPACE-complete (unless P = PSPACE): The word problem for the finite group S_5 is ALOGTIME-complete [2] but the compressed word problem for S_5 is in P, in fact it is P-complete [7]. Another example is the word problem for the free group F_2 of rank 2. Its compressed membership problem is P-complete by Theorem 4.9, whereas the word problem for F_2 is ALOGTIME-hard [57] but not even known to be in ALOGITME. Thus, in the framework of straight-line programs, a general upgrading theorem analogously to [68] does not hold. The next theorem states this fact in a more general context. For this, we introduce a parallel notion of reducibility that we call LOGDCFL-reducibility.

A deterministic logspace-bounded AuxPDA is a deterministic pushdown automaton that has an auxiliary read-write working tape of length $O(\log(n))$ for an input of length n [65]. It is known that a language can be recognized by a deterministic logspace-bounded AuxPDA in polynomial time if and only if it belongs to the class LOGDCFL, which is the class of all problems that are logspace reducible to a deterministic context-free language [65]. We say that a function $f : \Gamma^* \to \Sigma^*$ is computable in LOGDCFL if there exists a deterministic logspace-bounded AuxPDA with an output tape that computes for an input $x \in \Gamma^*$ the word $f(x)$ on the output tape in polynomial time. This leads to the notion of LOGDCFL-reductions. LOGDCFL-reducibility is denoted by $\leq_{\mathrm{LOGDCFL}}$. It is easy to see that a LOGDCFL-computable function belongs to the functional class $\mathrm{L}^{\mathrm{LOGCFL}}$ of [27], which is the class of all functions that can be computed by a logspace transducer which has additional access to an oracle from LOGCFL. Since $\mathrm{L}^{\mathrm{LOGCFL}}$ is contained in functional NC^2 [27], we see that LOGDCFL-computable functions also belong to functional NC^2.

If $\mathcal{R}$ is any notion of reducibility, then we write $A \equiv_{\mathcal{R}} B$ for $A \leq_{\mathcal{R}} B \leq_{\mathcal{R}} A$. In the following proposition, we denote for a language K by $\mathcal{C}(K)$ the compressed membership problem for K.

PROPOSITION 8.5. *For every language L there exists a language K such that $L \equiv_{\mathrm{NC}^1} K \equiv_{\mathrm{LOGDCFL}} \mathcal{C}(K)$.*

Proof. Let us fix a language $L \subseteq \Gamma^*$, let $\# \notin \Gamma$ be a new symbol, and define

$$K = \{a_1 \# a_2 \#^2 a_3 \cdots \#^{n-1} a_n \mid a_1 a_2 \cdots a_n \in L, a_1, \ldots, a_n \in \Gamma\}.$$

Then $L \equiv_{\mathrm{NC}^1} K$ is easy to see.

That K is LOGDCFL-reducible to $\mathcal{C}(K)$ is trivial. Thus, it remains to show $\mathcal{C}(K) \leq_{\mathrm{LOGDCFL}} K$. Note that if an SLP G generates a word $a_1 \# a_2 \#^2 a_3 \cdots \#^{n-1} a_n$ (which has length $\frac{n(n+1)}{2}$), then $|G| \geq n$. This is true, because if for a nonterminal A, $\mathrm{eval}_G(A)$ contains more than one symbols from Γ, then A can occur only once in the whole derivation tree generated by G. Now a LOGDCFL-reduction from $\mathcal{C}(K)$ to K can be implemented as follows: For a given SLP G, a deterministic logspace-bounded AuxPDA generates the word $\mathrm{eval}(G)$ on the pushdown in the same way as context-free languages are recognized on pushdown automata. Moreover, every time the AuxPDA pushes a terminal on the pushdown, it writes that terminal on the output tape, increments a counter by 1 and removes the terminal from the pushdown. If the counter reaches the value $\frac{|G|(|G|+1)}{2} + 1$ (which has $O(\log(|G|))$ many bits in its binary representation), then the AuxPDA terminates immediately (this ensures a polynomial running time) and writes for instance ## on the output in order to ensure that the generated output does not belong to K. If the counter does not reach the value $\frac{m(m+1)}{2} + 1$, then the AuxPDA finally has produced the word $\mathrm{eval}(G)$ on the output. We have described a LOGDCFL-reduction from $\mathcal{C}(K)$ to K. □

From Proposition 8.5 it follows that if C is a complexity class that is closed under NC^1-reductions and such that C has complete problems under some notion $\mathcal{R}$ of reducibility that is weaker than LOGDCFL-reducibility (e.g., NC^2-reducibility), then $\mathcal{C}$ contains a language L such that both L and $\mathcal{C}(L)$ are complete for C under $\mathcal{R}$-reducibility.

We remark that also for hierarchical graph descriptions [38] (which can be viewed as graph-generating straight-line programs) the correlation between the complexity of a problem in its compressed and uncompressed variant, respectively, is quite loose.

9. Uniform variants. Many of the decision problems in this paper can be also investigated in a uniform setting. For a class $\mathcal{C}$ of monoid presentations define the *compressed uniform word problem* for the class $\mathcal{C}$ as the following decision problem:

INPUT: A monoid presentation (Γ, R) and two SLPs G_1 and G_2 over the terminal alphabet Γ.

QUESTION: Does $\mathrm{eval}(G_1) \overset{*}{\leftrightarrow}_R \mathrm{eval}(G_2)$ hold?

Similarly we can define the *compressed uniform membership problem* for a class $\mathcal{C}$ of languages. Here, we have to specify the representation of a language from $\mathcal{C}$. For various representations of regular languages, the complexity of the uniform compressed membership problem was investigated in [55]. The upper bound in the following result was also stated in [55].

THEOREM 9.1. *The compressed uniform membership problem for the class of all context-free languages (represented by context-free grammars) is PSPACE-complete.*

Proof. The lower bound follows from Corollary 6.7. For the upper bound it was argued in [55] that one can use a $\mathrm{DSPACE}(\log^2(n))$ algorithm for parsing context-free languages, in the same way as in the proof of Corollary 6.7. But here, a problem arises: The uniform membership problem for context-free languages is P-complete, and thus probably not contained in $\bigcup_{c>0} \mathrm{DSPACE}(\log^c(n))$. On the other hand, by [26] the uniform membership problem for context-free grammars in *Chomsky normal form* can be solved in $\mathrm{DSPACE}(\log^2(|G| + |w|))$, where $|G|$ is the size of the input grammar and w is the word that has to be tested for membership. Since every context-free grammar can be transformed in polynomial time into Chomsky normal form, we can argue analogously to the proof of Proposition 6.1. □

REFERENCES

[1] F. Afrati, H. Leiß, and M. de Rougemont. Definability and compression. *Fundamenta Informaticae*, 56:155–180, 2003.

[2] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[3] D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[4] D. A. M. Barrington, C.-J. Lu, P. B. Miltersen, and S. Skyum. Searching constant width mazes captures the AC^0 hierarchy. In M. Morvan, C. Meinel, and D. Krob, editors, *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 98), Paris (France)*, number 1373 in Lecture Notes in Computer Science, pages 73–83. Springer, 1998.

[5] G. Bauer and F. Otto. Finite complete rewriting systems and the complexity of the word problem. *Acta Informatica*, 21:521–540, 1984.

[6] M. Beaudry, M. Holzer, G. Niemann, and F. Otto. McNaughton families of languages. *Theoretical Computer Science*, 290(3):1581–1628, 2003.

[7] M. Beaudry, P. McKenzie, P. Péladeau, and D. Thérien. Finite monoids: From word to circuit evaluation. *SIAM Journal on Computing*, 26(1):138–152, 1997.

[8] P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *Journal of Computer and System Sciences*, 65(2):332–350, 2002.

[9] R. V. Book. Homogeneous Thue systems and the Church–Rosser property. *Discrete Mathematics*, 48:137–145, 1984.

[10] R. V. Book, M. Jantzen, B. Monien, C. P. Ó'Dúnlaing, and C. Wrathall. On the complexity of word problems in certain Thue systems. In J. Gruska and M. Chytil, editors, *Proceedings*

of the 10th International Symposium on Mathematical Foundations of Computer Science (MFCS'81), Štrbské Pleso (Czechoslovakia), number 118 in Lecture Notes in Computer Science, pages 216–223. Springer, 1981.
[11] R. V. Book and F. Otto. *String–Rewriting Systems*. Springer, 1993.
[12] W. W. Boone. The word problem. *Annals of Mathematics (2)*, 70:207–265, 1959.
[13] B. Borchert and A. Lozano. Succinct circuit representations and leaf language classes are basically the same concept. *Information Processing Letters*, 59(4):211–215, 1996.
[14] G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML documents. In *Proceedings of the Tenth International Symposium on Database Programming Languages (DBPL 2005), Trondheim (Norway)*. Springer, 2005. to appear.
[15] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.
[16] M. Charikar, E. Lehman, D. Liu, R. P. abd M. Prabhakaran, A. Rasala, A. Sahai, and A. Shelat. Approximating the smallest grammar: Kolmogorov complexity in natural models. In *Proceedings of the 34th Annual Symposium on Theory of Computing (STOC 2002), Montréal (Canada)*, pages 792–801. ACM Press, 2002.
[17] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.
[18] S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8:385–394, 1987.
[19] M. Farach and M. Thorup. String matching in Lempel-Ziv compressed strings. *Algorithmica*, 20(4):388–404, 1998.
[20] J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. The complexity of problems on graphs represented as OBDDs. *Chicago Journal of Theoretical Computer Science*, 1999.
[21] H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Control*, 56(3):183–198, 1983.
[22] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–completeness*. Freeman, 1979.
[23] L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding (extended abstract). In R. G. Karlsson and A. Lingas, editors, *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (SWAT 1996), Reykjavík (Iceland)*, number 1097 in Lecture Notes in Computer Science, pages 392–403. Springer, 1996.
[24] B. Genest and A. Muscholl. Pattern matching and membership for hierarchical message sequence charts. In S. Rajsbaum, editor, *In Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN 2002), Cancun (Mexico)*, number 2286 in Lecture Notes in Computer Science, pages 326–340. Springer, 2002.
[25] L. M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–99, 1977.
[26] L. M. Goldschlager. ε-productions in context-free grammars. *Acta Informatica*, 16:303–308, 1981.
[27] G. Gottlob, N. Leone, and F. Scarcello. Computing LOGCFL certificates. *Theoretical Computer Science*, 270(1–2):761–777, 2002.
[28] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.
[29] C. Hagenah. *Gleichungen mit regulären Randbedingungen über freien Gruppen*. PhD thesis, University of Stuttgart, Institut für Informatik, 2000.
[30] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158(1&2):143–159, 1996.
[31] M. Holzer and K.-J. Lange. On the complexities of linear LL(1) and LR(1) grammars. In Z. Ésik, editor, *Proceedings of the 9th International Symposium on Fundamentals of Computation Theory (FCT'93), Szeged (Hungary)*, number 710 in Lecture Notes in Computer Science, pages 299–308. Springer, 1993.
[32] N. Immerman. *Descriptive Complexity*. Springer, 1999.
[33] M. Jantzen. Confluent string rewriting. In *EATCS Monographs on Theoretical Computer Science*, volume 14. Springer, 1988.
[34] M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.
[35] K.-J. Lange and P. McKenzie. On the complexity of free monoid morphisms. In K.-Y. Chwa and O. H. Ibarra, editors, *Proceedings of the ISAAC'98, Taejon (Korea)*, number 1533 in Lecture Notes in Computer Science, pages 247–256. Springer, 1998.
[36] J. Larus. Whole program paths. In *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 259–269. ACM Press,

1999.

[37] E. Lehman and A. Shelat. Approximation algorithms for grammar-based compression. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), San Francisco (USA)*, pages 205–212, 2002.

[38] T. Lengauer and K. W. Wagner. The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems. *Journal of Computer and System Sciences*, 44:63–93, 1992.

[39] P. M. Lewis II, R. E. Stearns, and J. Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proceedings of the Sixth Annual IEEE Symposium on Switching Circuit Theory and Logic Design*, pages 191–202, 1965.

[40] R. J. Lipton and Y. Zalcstein. Word problems solvable in logspace. *Journal of the Association for Computing Machinery*, 24(3):522–526, 1977.

[41] M. Lohrey. Word problems and confluence problems for restricted semi-Thue systems. In L. Bachmair, editor, *Proceedings of the 11th International Conference on Rewrite Techniques and Applications (RTA 2000), Norwich (UK)*, number 1833 in Lecture Notes in Computer Science, pages 172–186. Springer, 2000.

[42] M. Lohrey. Word problems for 2-homogeneous monoids and symmetric logspace. In J. Sgall, A. Pultr, and P. Kolman, editors, *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001), Marianske Lazne (Czech Republic)*, number 2136 in Lecture Notes in Computer Science, pages 500–511. Springer, 2001.

[43] M. Lohrey. Word problems on compressed word. In J. Diaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), Turku (Finland)*, number 3142 in Lecture Notes in Computer Science, pages 906–918. Springer, 2004.

[44] M. Lohrey. Model-checking hierarchical structures. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005), Chicago (USA)*. IEEE Computer Society Press, 2005. 168–177.

[45] M. Lohrey and S. Maneth. Tree automata and XPath on compressed trees. In *Proceedings of the Tenth International Conference on Implementation and Application of Automata (CIAA 2005), Sophia Antipolis (France)*. Springer, 2005. to appear.

[46] S. Maneth and G. Busatto. Tree transducers and tree compressions. In I. Walukiewicz, editor, *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2004), Barcelona (Spain)*, number 2987 in Lecture Notes in Computer Science, pages 363–377. Springer, 2004.

[47] A. Markov. On the impossibility of certain algorithms in the theory of associative systems. *Doklady Akademii Nauk SSSR*, 55, 58:587–590, 353–356, 1947.

[48] P. McKenzie and K. W. Wagner. The complexity of membership problems for circuits over sets of natural numbers. In H. Alt and M. Habib, editors, *Proceedings of the20th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2003), Berlin (Germany)*, number 2607 in Lecture Notes in Computer Science, pages 571–582. Springer, 2003.

[49] M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In A. Apostolico and J. Hein, editors, *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching (CPM 97), Aarhus (Denmark)*, Lecture Notes in Computer Science, pages 1–11. Springer, 1997.

[50] M.-J. Nederhof and G. Satta. The language intersection problem for non-recursive context-free grammars. *Information and Computation*, 192(2):172–184, 2004.

[51] P. S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *American Mathematical Society, Translations, II. Series*, 9:1–122, 1958.

[52] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[53] C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.

[54] W. Plandowski. Testing equivalence of morphisms on context-free languages. In J. van Leeuwen, editor, *Second Annual European Symposium on Algorithms (ESA'94), Utrecht (The Netherlands)*, number 855 in Lecture Notes in Computer Science, pages 460–470. Springer, 1994.

[55] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In J. Karhumäki, H. A. Maurer, G. Paun, and G. Rozenberg, editors, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 262–272. Springer, 1999.

[56] E. Post. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 12(1):1–11, 1947.

[57] D. Robinson. *Parallel Algorithms for Group Word Problems.* PhD thesis, University of California, San Diego, 1993.

[58] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

[59] W. Rytter. Algorithms on compressed strings and arrays. In J. Pavelka, G. Tel, and M. Bartosek, editors, *Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'99, Theory and Practice of Informatics), Milovy (Czech Republic)*, number 1725 in Lecture Notes in Computer Science, pages 48–65. Springer, 1999.

[60] W. Rytter. Compressed and fully compressed pattern matching in one and two dimensions. *Proceedings of the IEEE*, 88(11):1769–1778, 2000.

[61] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1–3):211–222, 2003.

[62] W. Rytter. Grammar compression, LZ-encodings, and string algorithms with implicit input. In J. Diaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), Turku (Finland)*, number 3142 in Lecture Notes in Computer Science, pages 15–27. Springer, 2004.

[63] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th Annual Symposium on Theory of Computing (STOC 1983)*, pages 61–69. ACM Press, 1983.

[64] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[65] I. H. Sudborough. On the tape complexity of deterministic context–free languages. *Journal of the Association for Computing Machinery*, 25(3):405–414, 1978.

[66] H. Veith. Languages represented by Boolean formulas. *Information Processing Letters*, 63(5):251–256, 1997.

[67] H. Veith. How to encode a logical structure by an OBDD. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity*, pages 122–131. IEEE Computer Society, 1998.

[68] H. Veith. Succinct representation, leaf languages, and projection reductions. *Information and Computation*, 142(2):207–236, 1998.

[69] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986.

[70] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

[71] K. Yamagata, T. Uchida, T. Shoudai, and Y. Nakamura. An effective grammar-based compression algorithm for tree structured data. In T. Horváth, editor, *Proceedings of the 13th International Conference on Inductive Logic Programming (ILP 2003), Szeged (Hungary)*, number 2835 in Lecture Notes in Artificial Intelligence, pages 383–400. Springer, 2003.

[72] Y. Zhang and R. Gupta. Path matching in compressed control flow traces. In *Proceedings of the 12th Data Compression Conference (DCC 2002), Snowbird (Utah, USA)*, pages 132–141. IEEE Computer Society Press, 2002.

[73] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.