Groups Complex. Cryptol. **4** (2012), 241–299
DOI 10.1515/gcc-2012-0016

Algorithmics on SLP-compressed strings: A survey

Markus Lohrey

Abstract. Results on algorithmic problems on strings that are given in a compressed form via straight-line programs are surveyed. A straight-line program is a context-free grammar that generates exactly one string. In this way, exponential compression rates can be achieved. Among others, we study pattern matching for compressed strings, membership problems for compressed strings in various kinds of formal languages, and the problem of querying compressed strings. Applications in combinatorial group theory and computational topology and to the solution of word equations are discussed as well. Finally, extensions to compressed trees and pictures are considered.

Keywords. Algorithms for compressed strings, compressed word problems, computational complexity.

2010 Mathematics Subject Classification. 68Q45, 68Q17, 20F10.

1 Introduction

The topic of this paper are algorithms on compressed strings. The goal of such algorithms is to check properties of compressed strings and thereby beat a straight forward "decompress-and-check" strategy. There are three main applications for algorithms of this kind.

• In many areas, large string data have to be not only stored in compressed form, but the initial data has to be processed and analyzed as well. Here, it makes sense to design algorithms that directly operate on the compressed string representation in order to save the time and space for (de)compression. Such a scenario can be found for instance in large genom databases or XML processing.

• Large and often highly compressible strings may appear as intermediate data structures in algorithms. Here, one may try to store a compressed representation of these intermediate data structures and to process this representation. This may lead to more efficient algorithms. Examples for this strategy can be found for instance in combinatorial group theory [56,57,94,96,124], computational topology [36, 121, 123], interprocedural analysis [51], and bisimulation checking [60, 78].

• In some situations it makes sense to compute in a first phase a compressed representation of an input string, which makes regularities in the string explicit. These regularities may be exploited in a second phase for speeding up an algorithm. This principle is known as *acceleration by compression*. It was recently applied in order to speed up the Viterbi algorithm for analyzing hidden Markov models [82] as well as speeding up edit distance computation [30, 58].

When we talk about algorithms on compressed strings, we have to make precise the compressed representation we want to use. Such a representation should have two properties: (i) it should cover many compression schemes from practice and (ii) it should be mathematically easy to handle. Straight-line programs (SLPs) have both of these properties. An SLP is a context-free grammar that generates exactly one word. In an SLP, repeated subpatterns in a string have to be represented only once by introducing a nonterminal for the pattern. It is not difficult to see that with an SLP consisting of n productions, a string of length 2^n can be generated by repeated doubling. In this sense, an SLP can indeed be seen as a compressed representation of the string it generates.

Several dictionary-based compressed representations, like for instance run-length encoding and the family of Lempel–Ziv (LZ) encodings [144], can be converted efficiently into straight-line programs of similar size. Moreover, a certain variant of LZ77-encoding turns out to be equivalent to SLPs in the following sense: From the LZ77-encoding of a string one can compute in polynomial time an SLP for the string, and, vice versa, from an SLP for a string one can compute in polynomial time the LZ77-encoding of the string [114]. This implies that complexity results can be transferred from SLP-encoded input strings to LZ77-encoded input strings. We will discuss the relationship between LZ77 and SLPs in more detail in Section 3.

The problem of computing a small SLP for a given input string is known as *grammar-based compression*; it will be briefly discussed in Section 4. Although computing a size-minimal SLP for a given input string is not possible in polynomial time unless $\mathsf{P} = \mathsf{NP}$ [23], there are several algorithms that compute for a given input string of length n an SLP that is only by a factor $\log(n)$ larger than a minimal grammar [23, 118, 120].

In Sections 5–10 we will consider algorithmic problems for SLP-compressed strings. The following types of algorithmic problems on compressed strings will be studied:

- Comparing compressed strings (Section 5): Are two compressed strings equal or similar in a certain sense?
- Pattern matching for compressed strings (Section 6 and 8): Does a com-

pressed string occur (in a certain sense) as a pattern in another compressed string?

- Membership problems for compressed strings in regular and context-free languages (Section 11): Does a compressed string belong to a certain regular or context-free language?
- Querying problems for compressed strings (Section 10): What is the symbol at a given position in a compressed string?

In Section 7 we discuss the relationship between SLPs and leaf languages (an important tool in complexity theory), which is a useful technique for proving lower bounds on the complexity of algorithmic problems for SLP-compressed strings.

In Section 11 we present applications in combinatorial group theory. In particular, we survey results on the *compressed word problem* for groups. The compressed word problem for the finitely generated group G asks whether an SLP-encoded input string over the generators of the group represents the group identity. It turned out that the compressed word problem for a group G is the right tool in order to attack the (non-compressed) word problem for (finitely generated subgroups of) the automorphism group of G. For instance, along this line it was shown that the word problem for the automorphism group of a free group can be solved in polynomial time [124].

Similar to group theory, SLP-techniques can also be used in order to show that several problems in computational topology can be solved in polynomial time; this is the topic of Section 13. These problems mainly concern curves in 2-dimensional surfaces that are represented by so-called normal coordinates. With the help of SLPs, it was shown for instance that Dehn twists and geometric intersection numbers of curves that are given in normal coordinates can be computed in polynomial time [123].

Finally, in Section 15 we briefly discuss algorithmic problems for extensions of SLPs to trees (Section 15.1) and 2-dimensional pictures (Section 15.2).

Previous surveys on algorithms for compressed strings can be found in [41,115, 119].

2 Preliminaries

2.1 General notations

Let Γ be a finite alphabet. The *empty word* 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 $1 \leq i \leq n$. The set of symbols occurring in s is $\mathrm{alph}(s) = \{a_1, \ldots, a_n\}$. The *length* of s is $|s| = n$. For $a \in \Gamma$ and $1 \leq i \leq j \leq n$ let $|s|_a = |\{k \mid a_k = a\}|$ be the number of a's in s. We also write

$s[i] = a_i$, and $s[i, j] = a_i a_{i+1} \cdots a_j$. If $i > j$ we set $s[i, j] = \varepsilon$. Any word $s[i, j]$ is called a *factor* of s.

We assume some background in complexity theory [109]. In particular, the reader should be familiar with the complexity classes L (deterministic logarithmic space) P (deterministic polynomial time), NP (nondeterministic polynomial time), coNP (languages, whose complement belongs to NP), PSPACE (polynomial space), EXPTIME (deterministic exponential time), and NEXPTIME (nondeterministic exponential time). As usual, when talking about space bounded classes (e.g., logarithmic space), we only count the required space on the work tape (but not the space occupied by the input). A problem $A \subseteq \Sigma^*$ is *logspace many-one reducible* to a problem $B \subseteq \Gamma^*$ if there exists a logspace computable function $f : \Sigma^* \to \Gamma^*$ such that for all $x \in \Sigma^*$: $x \in A$ if and only if $f(x) \in B$. When talking about efficient algorithms (e.g., linear time algorithms) one has to specify the underlying computational model. Unless otherwise specified, we always refer to the *random access model* (RAM) with uniform cost measure. The latter means that arithmetic operations on arbitrary numbers can be carried out in constant time. This might seem unrealistic. On the other hand, all algorithms mentioned in this paper only have to store numbers with $O(n)$ many bits, where n is the length of the input.

We also assume some basic knowledge in automata theory. In particular, the reader should be familiar with regular languages and context-free languages. Background can be found in the classical text book [61].

2.2 Straight-line programs

Definition 1 (straight-line program (SLP)). A *straight-line program (SLP)* over the terminal alphabet Γ is a context-free grammar $\mathbb{A} = (V, \Gamma, S, P)$ (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 the following two conditions hold:

(1) For every $A \in V$ there exists exactly one production of the form $(A, \alpha) \in P$ for $\alpha \in (V \cup \Gamma)^*$.

(2) The relation $\{(A, B) \mid (A, \alpha) \in P, B \in \mathrm{alph}(\alpha)\}$ is acyclic.

A production (A, α) is also written as $A \to \alpha$. Clearly, the language generated by the SLP $\mathbb{A}$ consists of exactly one word that is denoted by $\mathrm{eval}(\mathbb{A})$. More generally, from every nonterminal $A \in V$ we can generate exactly one word that is denoted by $\mathrm{eval}_{\mathbb{A}}(A)$ (thus $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}_{\mathbb{A}}(S)$). We omit the index $\mathbb{A}$ if the underlying SLP is clear from the context.

The *derivation tree* of the SLP $\mathbb{A} = (V, \Gamma, S, P)$ is a finite rooted ordered tree, where every node is labelled with a symbol from $V \cup \Gamma$. The root is labelled with

the initial nonterminal S and every node that is labelled with a symbol from Γ is a leaf of the derivation tree. A node that is labelled with a nonterminal A such that $(A \to \alpha_1 \cdots \alpha_n) \in P$ (where $\alpha_1, \ldots, \alpha_n \in V \cup \Gamma$) has n children that are labeled from left to right with $\alpha_1, \ldots, \alpha_n$.

The size of the SLP $\mathbb{A} = (V, \Gamma, S, P)$ is $|\mathbb{A}| = \sum_{(A,\alpha)\in P} |\alpha|$. Every SLP can be transformed in linear time into an equivalent SLP in *Chomsky normal form*, where every production has the form (A, a) with $a \in \Gamma$ or (A, BC) with $B, C \in V$.

Example 2. Consider the SLP $\mathbb{A}$ over the terminal alphabet $\{a, b\}$ that consists of the following productions: $A_1 \to b$, $A_2 \to a$, and $A_i \to A_{i-1}A_{i-2}$ for $3 \le i \le 7$. The start nonterminal is A_7. Then $\mathrm{eval}(\mathbb{A}) = abaababaabaab$, which is the 7th Fibonacci word. The SLP $\mathbb{A}$ is in Chomsky normal form and $|\mathbb{A}| = 12$.

A simple induction shows that for every SLP $\mathbb{A}$ of size m one has $|\mathrm{eval}(\mathbb{A})| \le O(3^{m/3})$ [23, proof of Lemma 1]. On the other hand, it is straightforward to define an SLP $\mathbb{B}$ in Chomsky normal form of size $2n$ such that $|\mathrm{eval}(\mathbb{B})| \ge 2^n$. Hence, an SLP can be seen as a compressed representation of the string it generates, and exponential compression rates can be achieved in this way.

One may also allow exponential expressions of the form A^i for $A \in V$ and $i \in \mathbb{N}$ in 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 construction from [84] is often useful for proving lower complexity bounds, see, e.g., the end of Section 5.

Example 3. Let $\overline{w} = (w_1, \ldots, w_n)$ be a tuple of natural numbers. For a bit vector $\overline{x} \in \{0,1\}^n$ of length n let us define $\overline{x} \cdot \overline{w} = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$. Let $\overline{1}_k$ be the constant-1 vector $(1, 1, \ldots, 1)$ of length k and let $\overline{w}_k = (w_1, \ldots, w_k)$ and $s_k = \overline{1}_k \cdot \overline{w}_k = w_1 + \cdots + w_k$ for $1 \le k \le n$. Let $s = s_n = w_1 + w_2 + \cdots + w_n$. Finally, define the string

$$S(\overline{w}) = \prod_{\overline{x}\in\{0,1\}^n} a^{\overline{x}\cdot\overline{w}} b a^{s-\overline{x}\cdot\overline{w}}.$$

Here, the product $\prod_{\overline{x}\in\{0,1\}^n}$ means that we concatenate all string $a^{\overline{x}\cdot\overline{w}} b a^{s-\overline{x}\cdot\overline{w}}$ for $\overline{x} \in \{0,1\}^n$ and the order of concatenation is the lexicographic order on $\{0,1\}^n$, where the right-most bit has highest significance. For example, we have

$$\begin{aligned} S(2,3,5) &= ba^{10}\, a^2 ba^8\, a^3 ba^7\, a^5 ba^5\, a^5 ba^5\, a^7 ba^3\, a^8 ba^2\, a^{10} b \\ &= ba^{12} ba^{11} ba^{12} ba^{10} ba^{12} ba^{11} ba^{12} b. \end{aligned}$$

Let us show that there is an SLP $\mathbb{A}$ of size polynomial in $\sum_{i=1}^{n} \log(w_i)$ for the string $S(\overline{w})$. Note that $\sum_{i=1}^{n} \log(w_i)$ is roughly the length of the binary encoding of the tuple $\overline{w}$. The productions of $\mathbb{A}$ are

$$A_1 \to b a^{s+w_1} b,$$
$$A_{k+1} \to A_k a^{s-s_k+w_{k+1}} A_k \quad (1 \le k \le n-1).$$

Let A_n be the start nonterminal of $\mathbb{A}$.

We prove by induction on k that

$$\mathrm{eval}_{\mathbb{A}}(A_k) = \Bigg(\prod_{\overline{x} \in \{0,1\}^k \setminus \{\overline{1}_k\}} a^{\overline{x}\cdot\overline{w}_k} b a^{s-\overline{x}\cdot\overline{w}_k} \Bigg) a^{s_k} b.$$

The case $k = 1$ is clear, since

$$\mathrm{eval}_{\mathbb{A}}(A_1) = b a^{s+w_1} b = b a^s a^{s_1} b = a^0 b a^{s-0} a^{s_1} b.$$

For $k+1 \le n$ we obtain the following:

$$\begin{aligned}
&\Bigg(\prod_{\overline{x} \in \{0,1\}^{k+1} \setminus \{\overline{1}_{k+1}\}} a^{\overline{x}\cdot\overline{w}_{k+1}} b a^{s-\overline{x}\cdot\overline{w}_{k+1}} \Bigg) a^{s_{k+1}} b \\
&\quad = \underbrace{\Bigg(\prod_{\overline{x} \in \{0,1\}^k} a^{\overline{x}\cdot\overline{w}_k} b a^{s-\overline{x}\cdot\overline{w}_k} \Bigg)}_{\mathrm{eval}_{\mathbb{A}}(A_k) a^{s-s_k}} \\
&\qquad \cdot \underbrace{\Bigg(\prod_{\overline{x} \in \{0,1\}^k \setminus \{\overline{1}_k\}} a^{\overline{x}\cdot\overline{w}_k + w_{k+1}} b a^{s-\overline{x}\cdot\overline{w}_k - w_{k+1}} \Bigg) a^{w_{k+1}} a^{s_k} b}_{a^{w_{k+1}} \mathrm{eval}_{\mathbb{A}}(A_k)} \\
&\quad = \mathrm{eval}_{\mathbb{A}}(A_k) a^{s-s_k+w_{k+1}} \mathrm{eval}_{\mathbb{A}}(A_k) = \mathrm{eval}_{\mathbb{A}}(A_{k+1}).
\end{aligned}$$

For $k = n$ we finally get

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

This concludes Example 3.

Let us state some simple algorithmic problems that can be easily solved in polynomial time:

(1) Given an SLP $\mathbb{A}$ (w.l.o.g. in Chomsky normal form), calculate $|\mathrm{eval}(\mathbb{A})|$: One introduces for each nonterminal A of the SLP $\mathbb{A}$ a variable n_A which takes values in $\mathbb{N}$. For a production $A \to a$ of $\mathbb{A}$ we set $n_A = 1$, and for a production $A \to BC$ of $\mathbb{A}$ we set $n_A = n_B + n_C$. In this way, the length $|\mathrm{eval}(\mathbb{A})|$ can be computed with $|\mathbb{A}|$ many additions. Moreover, the number of bits of each value n_A is bounded by $O(\log(|\mathrm{eval}_{\mathbb{A}}(A)|)) \leq O(|\mathbb{A}|)$.

(2) Given an SLP $\mathbb{A}$ (w.l.o.g. in Chomsky normal form) and a number $1 \leq i \leq |\mathrm{eval}(\mathbb{A})|$, calculate $\mathrm{eval}(\mathbb{A})[i]$ (this problem is in fact P-complete [81], see Section 10): Basically, the algorithm walks down in the derivation tree for $\mathbb{A}$ to position i. At each stage, the algorithm stores a number p and a nonterminal A of $\mathbb{A}$ such that $1 \leq p \leq |\mathrm{eval}_{\mathbb{A}}(A)|$. Initially, $p = i$, and A is the initial nonterminal of $\mathbb{A}$. If the unique production for A is $A \to BC$, then there are two cases: If $1 \leq p \leq |\mathrm{eval}_{\mathbb{A}}(B)|$, then we continue with position p and the nonterminal B. Otherwise, we have $|\mathrm{eval}_{\mathbb{A}}(B)| + 1 \leq p \leq |\mathrm{eval}_{\mathbb{A}}(A)|$, and we continue with position $p - |\mathrm{eval}_{\mathbb{A}}(B)|$ (this number can be computed using the algorithm from the previous point) and the nonterminal C. Finally, if the unique production for A is $A \to a$, then we must have $p = 1$ and we output the terminal symbol a.

(3) Given an SLP $\mathbb{A}$ over the terminal alphabet Γ and a homomorphism $\rho : \Gamma^* \to \Sigma^*$, calculate an SLP $\mathbb{B}$ such that $\mathrm{eval}(\mathbb{B}) = \rho(\mathrm{eval}(\mathbb{A}))$: We obtain $\mathbb{B}$ by simply replacing every occurrence of a terminal symbol a in $\mathbb{A}$ by the string $\rho(a)$.

Straight-line programs can also be viewed as particular *circuits*. Given an algebraic structure $\mathcal{A}$ with operations $f_1, \ldots, f_n$ (of arbitrary arity) and constants $c_1, \ldots, c_m$, a circuit over $\mathcal{A}$ is a finite directed acyclic graph D, where

- every node of indegree 0 is labelled with one of the constants $c_1, \ldots, c_m$,
- every node of indegree $n \geq 1$ is labelled with an operation f_i of arity n,
- the incoming edges for a node are linearly ordered, and
- there is a unique node of outdegree 0 (the output node of D).

Such a circuit computes an element of $\mathcal{A}$. An SLP over the terminal alphabet Γ is nothing else than a circuit over the free monoid Γ^*, where the only operation is concatenation and the constants are the symbols from the alphabet Γ. It should be noted that in the literature, the term "straight-line program" often refers to circuits over the ring of integers or, more generally, polynomial rings, see, e.g., [3, 63, 69].

For some applications, an extension of SLPs, called *composition systems* in [42], is useful.[1] A composition system $\mathbb{A}$ is defined as an SLP but may also contain productions of the form $X \to Y[i, j]$ for $i, j \in \mathbb{N}$ with $1 \leq i \leq j$. Assume that the string $w = \mathrm{eval}_{\mathbb{A}}(Y)$ is already defined. Then we let $\mathrm{eval}_{\mathbb{A}}(X) = w[i, \max\{|w|, j\}]$. A composition system is in Chomsky normal form if all productions have the form $X \to a$, $X \to YZ$ or $X \to Y[i, j]$ for nonterminals X, Y, and Z and a terminal symbol a. The following result was shown by Hagenah in his Ph.D. thesis [54] (in German), see also [124].

Theorem 4. *From a given composition system $\mathbb{A}$ in Chomsky normal form with n nonterminals one can compute in time $O(n^2)$ an SLP $\mathbb{B}$ of size $O(n^2)$ such that* $\mathrm{eval}(\mathbb{B}) = \mathrm{eval}(\mathbb{A})$.

3 Straight-line programs and dictionary-based compression

As already remarked in the Introduction, there is a tight relationship between straight-line programs and dictionary-based compression, in particular, the LZ77 compression scheme. There are various variants of LZ77 compression. The following variant is used for instance in [118]: The *LZ77-factorization* of a non-empty word $s \in \Sigma^+$ is defined as $s = f_1 f_2 \cdots f_m$, where for all $1 \leq i \leq m$, f_i is the longest non-empty prefix of $f_i f_{i+1} \cdots f_m$ that is a factor of $f_1 f_2 \cdots f_{i-1}$ in case such a prefix exists, otherwise f_i is the first symbol of $f_i f_{i+1} \cdots f_m$. Then, the *LZ77-encoding* of w is the sequence $c_1 c_2 \cdots c_m$ such that for all $1 \leq i \leq m$, either $|f_i| = 1$ and $c_i = f_i$ or $|f_i| \geq 2$, $f_i = (f_1 f_2 \cdots f_{i-1})[p, q]$ and $f_i \neq (f_1 f_2 \cdots f_{i-1})[k, k + q - p]$ for all $1 \leq k < p$.

Theorem 5 ([23, 118]). *For every SLP $\mathbb{A}$, the number of factors in the LZ-factorization of* $\mathrm{eval}(\mathbb{A})$ *is bounded by* $|\mathbb{A}|$.

Theorem 6 ([23,118]). *From the LZ77-factorization of a given string $w \in \Sigma^*$, one can compute an SLP of size $O(\log(|w|/m) \cdot m)$ for w in time $O(\log(|w|/m) \cdot m)$, where m is the number of factors in the LZ77-factorization of w.*

Rytter [118] uses SLPs, whose derivation tree is an AVL-tree for the proof of Theorem 6. An AVL-tree is a binary tree such that for every node v the height of the left subtree of v and the height of the right subtree of v differ by at most 1. The construction of Charikar et al. [23] is based on so-called α-balanced SLPs. An SLP $\mathbb{A}$ is α-balanced, where $0 < \alpha \leq 1/2$, if the right-hand side of every

[1] The formalism in [42] differs in some minor details. Moreover, composition systems are called *collage systems* in [73] and interval grammars in [54].

production has length two and for every production $A \to \beta\gamma$ of $\mathbb{A}$ (where β and γ are nonterminals or terminal symbols) the following holds: If m (resp., n) is the length of the string derived from β (resp., γ), then

$$\frac{\alpha}{1-\alpha} \leq \frac{m}{n} \leq \frac{1-\alpha}{\alpha}.$$

Using the fact that the number of factors in the LZ77-factorization of an SLP-compressed word $\mathrm{eval}(\mathbb{A})$ is bounded by the size of the SLP $\mathbb{A}$ (Theorem 6) as well as the results from Section 6 on compressed pattern matching, one can also show that the LZ77-encoding of $\mathrm{eval}(\mathbb{A})$ can be computed in polynomial time from $\mathbb{A}$. Hence, SLPs and LZ77-encodings are equivalent with respect to polynomial time transformations.

4 Grammar-based compression

As explained in the Introduction, in this paper we mainly deal with algorithmic problems on strings, which are already given in compressed form as an SLP. In this section, we will briefly consider the problem of compression itself, i.e., the computation of a small SLP for a given input string. This problem is known as grammar-based compression, and is considered for instance in [23, 100, 118, 120]. For a string w let $\mathsf{opt}(w)$ be the size of a minimal SLP for w. Thus, there exists an SLP $\mathbb{A}$ with $\mathrm{eval}(\mathbb{A}) = w$, $|\mathbb{A}| = \mathsf{opt}(w)$ and for every SLP $\mathbb{B}$ with $\mathrm{eval}(\mathbb{B}) = w$, $|\mathbb{B}| \geq |\mathbb{A}|$ holds. The following theorem provides a lower bound for grammar-based compression.

Theorem 7 ([23]). *Unless* $\mathsf{P} = \mathsf{NP}$ *there is no polynomial time algorithm with the following specification:*

- *The input consists of a string* w *over some alphabet* Σ.
- *The output is a grammar* $\mathbb{A}$ *such that* $\mathrm{eval}(\mathbb{A}) = w$ *and*

$$|\mathbb{A}| \leq 8569/8568 \cdot \mathsf{opt}(w).$$

Theorem 7 is shown in [23] using a reduction from the vertex cover problem for graphs with maximal degree 3. In Theorem 7 it is important that the alphabet Σ is not fixed. It is an open problem whether there is a polynomial time algorithm that computes for a binary string $w \in \{0,1\}^*$ an SLP $\mathbb{A}$ such that $\mathrm{eval}(\mathbb{A}) = w$ and $|\mathbb{A}| = \mathsf{opt}(w)$. Some partial results can be found in [8].

The best known upper bound for grammar-based compression is provided by the following results that was shown independently by Charikar et al. [23] and Rytter [118]:

Theorem 8 ([23, 118]). *There is an $O(\log(|\Sigma|) \cdot n)$-time algorithm that computes for a given word $w \in \Sigma^*$ of length n an SLP $\mathbb{A}$ such that* $\mathrm{eval}(\mathbb{A}) = w$ *and* $|\mathbb{A}| \leq O(\log(n/\mathsf{opt}(w)) \cdot \mathsf{opt}(w))$*, i.e., the approximation ratio of the algorithm is* $O(\log(n/\mathsf{opt}(w)))$.

The algorithms of Charikar et al. and Rytter follow a similar strategy. First, the LZ77-encoding of the input string w is computed. This is possible in time $O(\log(|\Sigma|) \cdot |w|)$ using suffix trees, see, e.g., [52]. Let m be the number of factors in the LZ77-factorization of w. By Theorem 5, we have $m \leq \mathsf{opt}(w)$. In a second step, the LZ77-encoding of w is transformed into an SLP for w of size at most $O(\log(|w|/m) \cdot m) \leq O(\log(|w|/\mathsf{opt}(w)) \cdot \mathsf{opt}(w))$ using Theorem 6.

Rytter's technique [118] can also be used to transform an SLP $\mathbb{A}$ for a string of length n in time $O(|\mathbb{A}| \cdot \log(n))$ into an equivalent SLP of height $O(\log(n))$, where the height of an SLP is the height of the corresponding derivation tree. Hence, there is a very efficient algorithm for making SLPs balanced.

Let us remark that the existence of a polynomial time algorithm that computes from a given string w an SLP $\mathbb{A}$ for w of size $o(\log(|w|)/\log\log(|w|)) \cdot \mathsf{opt}(w)$ would imply a major progress on the problem of computing shortest addition chains, see [23]. In this problem, a sequence $k_1, \ldots, k_m$ of binary encoded natural numbers is given, and one is looking for a smallest circuit over $(\mathbb{N}, +)$ such that every k_i is the value of some gate. Currently the best approximation algorithm for this problem is by Yao [143] (from 1976) and its approximation ratio is $O(\log n/\log\log n)$, where $n = k_1 + \cdots + k_m$.

Some optimizations of Rytter's algorithm can be found in [17]. Another grammar-based compression algorithm with linear running time (for a fixed alphabet) and an approximation ratio of $O(\log(n/\mathsf{opt}(w)))$ was presented by Sakamoto [120]. His algorithm is based on the RePair compressor [77], which is another popular grammar-based compressor.

The algorithms from [23, 118, 120] all use space $\Omega(n)$ in order to generate an SLP for a string of length n, which might be a problem for large input texts. A grammar-based compression algorithm with an approximation ratio of $O(\log^2(n))$ that runs in linear time and needs no more space than the size of the output grammar was recently presented in [100].

Grammar-based compression using a weak form of context-sensitive grammars (so-called Σ-sensitive grammars) is discussed in [101].

5 Compressed equality checking

The most basic task for SLP-compressed strings is equality checking: Given two SLPs $\mathbb{A}$ and $\mathbb{B}$, does $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$ hold? Clearly, a simple decompress-and-

compare strategy is very inefficient. It takes exponential time to compute eval($\mathbb{A}$) and eval($\mathbb{B}$). Nevertheless a polynomial time algorithm exists. This was independently discovered by Hirshfeld, Jerrum, and Moller [60], Mehlhorn, Sundar, and Uhrig [104], and Plandowski [111].

Theorem 9 ([60, 104, 111]). *The following problem can be solved in polynomial time:*

Input *Two SLPs* $\mathbb{A}$ *and* $\mathbb{B}$.

Question $\text{eval}(\mathbb{A}) = \text{eval}(\mathbb{B})$*?*

In [104], the result is not explicitly stated but follows immediately from the main result. Mehlhorn et al. provide an efficient data structure D for a finite set of strings that supports the following operations:

- Add a string consisting of the symbol a to D.
- For two strings x and y in D, add the string xy (here, one may have $x = y$) to D. Moreover, the strings x and y remain in D (the latter property is called persistence).
- For a string x in D, and a position k in x, split x into its length-k prefix and the remaining suffix and add both strings to D. Again, x remains in D.
- Check whether two strings x and y from D are identical.

The idea is to compute for each variable a signature, which is a small number, and that allows to do the equality test in constant time by comparing signatures. A signature corresponds to a nonterminal of an SLP. The signature of a string is computed by iteratively breaking up the sequence into small blocks, which are encoded by integers using a pairing function. Adding the concatenation xy of two strings x and y to the data structure needs time $O(\log n(\log m \log^* m + \log n))$ for the mth operation, where n is the length of the resulting string (hence, $\log(n) \leq m$). This leads to a cubic time algorithm for checking equality of SLP-compressed strings. An improvement of the data structure from [104] was obtained by Alstrup, Brodal, and Rauhe [5]; see [4] for a long version. There, the complexity of adding the concatenation xy of two strings x and y to the data structure is improved to $O(\log n(\log^* m + \log|\Sigma|))$, where n and m have the same meaning as above and Σ is the underlying alphabet. This leads to the complexity of $O(n^2 \log^* n)$ (where $n = |\mathbb{A}| + |\mathbb{B}|$) for checking equality of SLP-compressed strings, which is currently the best upper bound. One can view the algorithms from [5, 104] as efficient methods for transforming an SLP $\mathbb{A}$ into a canonical SLP $\mathbb{A}$ such that $\text{eval}(\mathbb{A}) = \text{eval}(\mathbb{B})$ if and only if $\mathbb{A}'$ and $\mathbb{B}'$ are identical. One should note that the

machine model in [5] is a RAM that allows to compute bitwise XOR and AND of machine words in constant time.

In contrast to [104], the polynomial time algorithms of Hirshfeld et al. [60] and Plandowski [111] use combinatorial properties of words, in particular the periodicity lemma of Fine and Wilf [38]. This lemma states that if p and q are periods of a string w (i.e., $w[i] = w[i+p]$ and $w[j] = w[j+q]$ for all positions $1 \leq i \leq |w| - p$ and $1 \leq j \leq |w| - q$) and $p + q \leq |w|$ then also the greatest common divisor of p and q is a period of w. The algorithm from [60] achieves a running time of $O(n^4)$, where $n = |\mathbb{A}| + |\mathbb{B}|$.

Both, Plandowski and Hirshfeld et al. use Theorem 9 as a tool to solve another problem. Plandowski derives from Theorem 9 a polynomial time algorithm for testing whether two given morphisms (between free monoids) agree on a given context-free language. Hirshfeld et al. use Theorem 9 in order to check bisimilarity of two normed context-free processes (a certain class of infinite state systems) in polynomial time.

The equality problem for SLP-compressed strings can also be viewed as a very restricted case of the equality problem for context-free languages, where the two context-free languages are singleton sets. One might try to generalize Theorem 9 to larger classes of context-free grammars. Equality of arbitrary context-free languages is a well-known undecidable problem [61]. An important class in between SLPs and general context-free grammars are *acyclic context-free grammars* (also known as non-recursive context-free grammars). These are context-free grammars such that from a nonterminal A one cannot derive in ≥ 1 steps a word containing A. In other words, we only keep condition (2) in Definition 1. Acyclic context-free grammars generate only finite sets of words. Let G be an acyclic context-free grammar of size n. Then, every word generated by G has length $2^{O(n)}$, and $|L(G)| \leq 2^{2^{O(n)}}$.

Unfortunately, Theorem 9 cannot be generalized to acyclic context-free grammars, as the following result from [62] shows:

Theorem 10 ([62]). *It is* NEXPTIME*-complete to check for two given acyclic context-free grammars G_1 and G_2, whether $L(G_1) \neq L(G_2)$.*

In [107], it was shown that the problem of checking $L(G_1) \cap L(G_2) = \emptyset$ for two given acyclic context-free grammars G_1 and G_2 is PSPACE-complete.

Let us finally mention a related result of Lifshits [80], which when compared with Theorem 9 shows again the quite subtle borderline between tractability and intractability for problems on compressed strings. A function $f : \Sigma^* \to \mathbb{N}$ belongs to #P if there exists a nondeterministic polynomial time bounded Turing-machine M such that for every $x \in \Sigma^*$, $f(x)$ equals the number of accepting

paths of M on input x. A function $f : \Sigma^* \to \mathbb{N}$ is #P-complete if it belongs to #P and for every #P-function $g : \Gamma^* \to \mathbb{N}$ there is a logspace computable mapping $h : \Gamma^* \to \Sigma^*$ such that $h \circ f = g$. Functions that are #P-complete are computationally very powerful. By Toda's [134] famous result, every language from the polynomial time hierarchy can be decided in deterministic polynomial time with the help of a #P-function, briefly: $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$. For two strings u and v of the same length m, the *Hamming-distance* $d_H(u, v)$ is the number of positions $i \in \{1, \ldots, m\}$ such that $u[i] \neq v[i]$.

Theorem 11 ([80]). *The mapping* $(\mathbb{A}, \mathbb{B}) \mapsto d_H(\mathrm{eval}(\mathbb{A}), \mathrm{eval}(\mathbb{B}))$*, where* $\mathbb{A}$ *and* $\mathbb{B}$ *are SLPs, is* #P*-complete.*

This result can be shown using the construction from Example 3. Standard arguments show that the function that maps a tuple of binary encoded natural numbers $w_1, \ldots, w_n, t$ to the number of tuples $\overline{x} \in \{0,1\}^n$ such that $\overline{x} \cdot (w_1, \ldots, w_n) = t$ is #P-complete (this can be shown by the same reduction that reduces the 3SAT problem to the subsetsum problem); we use here the notation from Example 3. Now, given a tuple of binary encoded natural numbers $w_1, \ldots, w_n, t$, we construct in logspace two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that the following holds (as in Example 3 let $s = w_1 + \cdots + w_n$):

$$\mathrm{eval}(\mathbb{A}) = \prod_{\overline{x} \in \{0,1\}^n} a^{\overline{x} \cdot \overline{w}} b a^{s - \overline{x} \cdot \overline{w}}, \qquad \mathrm{eval}(\mathbb{B}) = (a^t b a^{s-t})^{2^n}.$$

The SLP $\mathbb{B}$ can easily be constructed, and Example 3 shows how to construct $\mathbb{A}$ (in logspace). Now, it is an easy observation that $d_H(\mathrm{eval}(\mathbb{A}), \mathrm{eval}(\mathbb{B}))$ is exactly the number of tuples $\overline{x} \in \{0,1\}^n$ such that $\overline{x} \cdot (w_1, \ldots, w_n) = t$.

6 Compressed pattern matching

A natural generalization of checking equality of two strings is pattern matching. In the classical *pattern matching problem* it is asked for given strings p (usually called the pattern) and t (usually called the text), whether p is a factor of t. There are dozens of linear time algorithms for this problem on uncompressed strings, most notably the well-known Knuth–Morris–Pratt algorithm [74]. It is therefore natural to ask, whether a polynomial time algorithm for pattern matching on SLP-compressed strings exists; this problem is sometimes called *fully compressed pattern matching* and is defined as follows:

Input Two SLPs $\mathbb{P}$ and $\mathbb{T}$.

Question Is $\mathrm{eval}(\mathbb{P})$ a factor of $\mathrm{eval}(\mathbb{T})$?

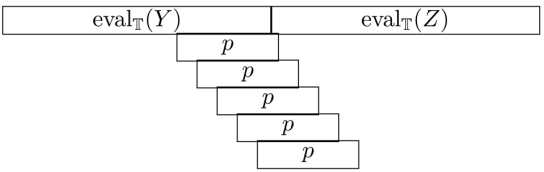

Figure 1. Occurrences of a pattern p that touch the cut of X, where $X \to YZ$.

The first polynomial time algorithm for fully compressed pattern matching was presented in [72], further improvements with respect to the running time were achieved in [41, 66, 80, 105].

Theorem 12 ([41, 66, 72, 80, 105]). *Fully compressed pattern matching can be solved in polynomial time.*

Basically, the algorithms from [41, 66, 72, 80, 105] can be divided into two classes: Those that are based on the periodicity lemma of Fine and Wilf [38] mentioned in the previous section [41, 72, 80, 105] and Jez's solution from [66] that does not use any non-trivial combinatorial properties of words.

An important observation that is crucial in [41,66,72,80,105] is the obvious fact that if a pattern p is a factor of eval($\mathbb{T}$) then there exists a production $X \to YZ$ in eval($\mathbb{T}$) such that p has an occurrence in $\text{eval}_{\mathbb{T}}(X) = \text{eval}_{\mathbb{T}}(Y)\text{eval}_{\mathbb{T}}(Z)$ that "touches" the cut (or border) position between $\text{eval}_{\mathbb{T}}(Y)$ and $\text{eval}_{\mathbb{T}}(Z)$ (called the cut of X). It is a consequence of the periodicity lemma of Fine and Wilf [38] mentioned in the previous section that the set of all starting positions of occurrences of p in $\text{eval}_{\mathbb{T}}(X)$ that touch the cut of X forms an arithmetic progression, i.e., a set of the form $\{b + i \cdot p \mid 0 \leq i < \ell\}$ (see Figure 1), which can be represented by the binary encoded triple (b, p, ℓ). Lifshits' algorithm [80] for instance computes for each nonterminal A of the pattern SLP $\mathbb{P}$ and each nonterminal X of the text SLP $\mathbb{T}$ the arithmetic progression corresponding to the occurrences of $\text{eval}_{\mathbb{P}}(A)$ in $\text{eval}_{\mathbb{T}}(X)$ that touch the cut of X. These arithmetic progressions are computed in a bottom-up fashion spending time $O(|\mathbb{P}|)$ for each progression, which results in the overall time bound $O(|\mathbb{P}|^2|\mathbb{T}|)$.

As already mentioned above, Jez's algorithm for fully compressed pattern matching [66] does not use non-trivial combinatorial properties of words. Moreover, it is the currently fastest algorithm; its running time is $O((|\mathbb{T}|+|\mathbb{P}|)\log(M)\cdot \log(|\mathbb{T}| + |\mathbb{P}|))$, where M is the length of eval($\mathbb{P}$). Since $\log(M) \in O(|\mathbb{P}|)$, the running time can be upper bounded by $O(n^2 \log(n))$, where n is the total input length. The basic idea of the algorithm is to rewrite the SLP for the pattern and the text simultaneously; Jez calls this process *recompression*. In a single round of this

recompression process, maximal blocks of the form a^n (for a letter a) and length-2 words ab (for a and b different letters) become new alphabet symbols. Moreover, the combined length of the (fully decompressed) text and pattern shrinks by a constant factor in each single recompression round. Hence, the recompression process terminates after $O(\log M)$ many rounds with a pattern of length 1. Some effort is necessary in order to show that during this process the intermediate SLPs are small. Jez's recompression technique uses some of the ideas from the equality-checking algorithms from [5, 104]. In particular the idea of breaking up a text into maximal blocks of the form a^n appears in [5, 104] too. One might say that Jez extends the techniques from [5, 104] so that pattern matching (instead of only equality checking) of SLP-compressed strings can be done efficiently.

Randomized algorithms for checking equality of compressed strings and fully compressed pattern matching are studied in [42, 128]. These algorithms are based on arithmetic modulo small prime numbers. The algorithm from [128] has a quadratic running time under the RAM model with *logarithmic cost measure*, which means that arithmetic operations on n-bit numbers need time $O(n)$. If $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$ then the algorithm will correctly output "yes"; if $\mathrm{eval}(\mathbb{A}) \neq \mathrm{eval}(\mathbb{B})$ then the algorithm may incorrectly output "yes" with a small error probability.

In the fully compressed pattern matching problem, the term "fully" refers to the fact that both the pattern and the text are compressed. In practical applications, the pattern is usually short in comparison to the text. In such a situation, it makes sense to consider the weaker variant, where only the text is compressed, but the pattern is given explicitly. This leads to the *semi-compressed pattern matching problem* (sometimes just called *compressed pattern matching problem*), where it is asked whether an uncompressed pattern p is a factor of a compressed text $\mathrm{eval}(\mathbb{T})$. Semi-compressed pattern matching has been studied intensively for various compression schemes, e.g., LZW (Lempel–Ziv–Welch) [7, 43] and LZ1 (a variant of LZ77 as defined in Section 3) [37, 44]. For SLP-compressed texts it is easy to show that compressed pattern matching can be solved in time $O(|p| \cdot |\mathbb{T}|)$. Assume that the text SLP $\mathbb{T}$ is in Chomsky normal form. As remarked earlier, it suffices to check for each nonterminal X of $\mathbb{T}$, whether there exists an occurrence of p in $\mathrm{eval}_{\mathbb{T}}(X)$ that touches the cut of X. For this, one computes for every nonterminal X the prefix pref_X of $\mathrm{eval}_{\mathbb{T}}(X)$ of length $\min\{|p|, |\mathrm{eval}_{\mathbb{T}}(X)|\}$ as well as the suffix suff_X of $\mathrm{eval}_{\mathbb{T}}(X)$ of length $\min\{|p|, |\mathrm{eval}_{\mathbb{T}}(X)|\}$. This is possible in time $O(|p| \cdot |\mathbb{T}|)$ by a straightforward bottom-up computation. Then, one only has to check for every production $X \to YZ$ of $\mathbb{T}$, whether p is a factor of $\mathsf{suff}_Y \mathsf{pref}_Z$, which is a string of length at most $2p$. This is again possible in time $O(|p| \cdot |\mathbb{T}|)$ using any linear time pattern matching algorithm for uncompressed strings. It should be also remarked that the currently best algorithm for semi-compressed pattern matching

for LZ1 [44] first transforms the LZ1-encoding of the text into an α-balanced SLP using the method from [23] (see Theorem 6) and then runs an algorithm for semi-compressed pattern matching for α-balanced SLPs.

Several other algorithmic problems that are related to compressed pattern matching were studied. In [25, 26, 39], self-indexes for SLP-compressed strings are constructed. The goal of such a self-index is to store an SLP $\mathbb{A}$ with small space overhead in such a form that for a given uncompressed pattern p one can quickly list all occurrences of p in eval($\mathbb{A}$). Bille et al. [15] present an efficient algorithm for approximative compressed pattern matching for SLP-compressed strings: Here, for a given pattern p, SLP $\mathbb{T}$, and $k \in \mathbb{N}$ the goal is to find all occurrences of factors of eval($\mathbb{T}$) that have distance at most k from p with respect to a certain distance measure (e.g., Hamming distance or edit distance). The problem of computing q-gram frequencies for SLP-compressed strings is studied in [48,49]. A q-gram is just a string of length q. An algorithm that computes in time $O(q \cdot n)$ for a Chomsky normal form SLP $\mathbb{A}$ of size n a list with the frequencies of all q-grams occurring in eval($\mathbb{A}$) is presented in [48]. This algorithm can be seen as a refinement of the algorithm for semi-compressed pattern matching outlined above. In [102], it is shown that the length of the longest common factor of two SLP-compressed strings can be computed in polynomial time.

7 Leaf languages and string compression

In this section, we discuss a technique that turned out to be useful for deriving lower complexity bounds for algorithmic problems on SLP-compressed strings. This technique is based on the concept of leaf languages from complexity theory. A detailed survey of leaf languages can be found in [139]. Here, we will only introduce the basic definitions necessary in order to understand the relationship to SLPs.

Let M be a nondeterministic Turing-machine. We say that M is *adequate* if it satisfies the following three properties:

- On every input M does not have an infinite computation path.
- A linear order $<$ is fixed on the set of transition tuples of M.
- To every configuration of M at most two transition tuples of M apply, i.e., every configuration has at most two successor configurations.

For an adequate Turing-machine M and an input w, the computation tree $T_M(w)$ of M on input w becomes a finite rooted ordered binary tree, where nodes are labelled with configurations of M. The root is labelled with the initial configuration

on input w. Now, assume that a node v of $T_M(w)$ is labelled with the configuration c, and c has exactly two successor configurations c_1 and c_2 (the case that c has at most one successor configuration is analogous). Assume moreover that c_i results from c by applying the transition tuple t_i to c and $t_1 < t_2$. Then, v has two children v_1 and v_2 in this order and v_i is labelled with the configuration c_i. The leaf string $\mathsf{leaf}(M, w)$ is defined as follows: Let $v_1, \ldots, v_m$ be the leaves of the ordered tree $T_M(w)$ in the natural left-to-right order, and set $b_i = 1$ (resp., $b_i = 0$) if the configuration that labels node v_i is accepting (resp., rejecting). Then, we set $\mathsf{leaf}(M, w) = b_1 \cdots b_m$.

With a language $L \subseteq \{0,1\}^*$, one can associate a complexity class $\mathsf{LEAF}^{\mathsf{p}}(L)$ as follows: A language $K \subseteq \Sigma^*$ belongs to the class $\mathsf{LEAF}^{\mathsf{p}}(L)$ if there exists an adequate nondeterministic polynomial time Turing-machine M with input alphabet Σ such that for every word $w \in \Sigma^*$ we have: $w \in K$ if and only if $\mathsf{leaf}(M, w) \in L$. In this way, many complexity classes can be defined in a uniform way. Here are a few examples:

- $\mathsf{NP} = \mathsf{LEAF}^{\mathsf{p}}(\{0,1\}^*1\{0,1\}^*)$.
- $\mathsf{coNP} = \mathsf{LEAF}^{\mathsf{p}}(1^*)$.
- $\mathsf{PP} = \mathsf{LEAF}^{\mathsf{p}}(\{x \in \{0,1\}^* \mid |x|_1 > |x|_0\})$; here PP stands for probabilistic polynomial time.[2]

Of course, the leaf language concept is not restricted to nondeterministic polynomial time Turing-machines. In the context of SLP-compression, logspace leaf language classes [64] turned out to be useful: A language $K \subseteq \Sigma^*$ belongs to the class $\mathsf{LEAF}^{\mathsf{log}}(L)$ if there exists an adequate nondeterministic logspace Turing-machine M with input alphabet Σ such that for every word $w \in \Sigma^*$ we have: $w \in K$ if and only if $\mathsf{leaf}(M, w) \in L$.

Note that the machine M together with an input w can be seen as a compressed representation of the string $\mathsf{leaf}(M, w)$. In a certain sense, if the machine M is logspace-bounded, then this form of compression is equivalent to SLP-compression:

Theorem 13 ([87]). *The following holds:*

- *Let M be an adequate nondeterministic logspace Turing-machine. From a given input $w \in \Sigma^*$ for M we can construct in logspace an SLP $\mathbb{A}$ over $\{0,1\}$ such that* $\mathrm{eval}(\mathbb{A}) = \mathsf{leaf}(M, w)$.

[2] If we view a nondeterministic Turing machine as a probabilistic machine that chooses successor configurations with uniform distribution, then PP can be defined as the class of all languages L for which there exists a probabilistic machine M such that for all inputs x: $x \in L$ if and only if M accepts x with probability $> 1/2$.

- *There exists an adequate nondeterministic logspace Turing-machine M that takes as input an SLP $\mathbb{A}$ in Chomsky normal form over the alphabet $\{0,1\}$ and produces the leaf string* $\mathsf{leaf}(M,\mathbb{A}) = \mathrm{eval}(\mathbb{A})$.

The proof of this result is quite simple. For the first statement, let n be the length of the input w and assume that M works in space $\log(n)$ on an input of length n. Then the nonterminals of the SLP $\mathbb{A}$ are all configurations of M, where the input tape contains the word w and the work tape is restricted to length $\log(n)$. There is only a polynomial number of such configurations. If the successor configurations of the configuration c are $c_1, \ldots, c_m$ and $m \geq 1$ (we must have $m \leq 2$ since M is adequate) then we add the production $c \to c_1 \cdots c_m$ to the SLP. If c is a configuration without successor configurations, then we add the production $c \to 1$ (resp., $c \to 0$) if c is an accepting (resp., rejecting) configuration. Finally, the initial nonterminal of the SLP $\mathbb{A}$ is the initial configuration corresponding to input w. Clearly, this SLP $\mathbb{A}$ produces the leaf string $\mathsf{leaf}(M,w)$.

For the second statement of Theorem 13 we take the adequate nondeterministic logspace Turing-machine M that stores on the work tape a nonterminal of the input SLP $\mathbb{A}$; logarithmic space suffices for this. If in a configuration c the work tape stores the nonterminal A, then M searches (deterministically) the unique production with left hand A. If this production is of the form $A \to 1$ (resp., $A \to 0$), then M terminates and accepts (resp., rejects). Otherwise, assume that the production for A has the form $A \to BC$. Then M branches nondeterministically into two successor configurations c_1 and c_2. From c_1 (resp., c_2), M updates the content of the work tape to the nonterminal B (resp., C).

The main result of [87] shows that also the leaf strings produced by nondeterministic polynomial time Turing-machines can be produced by SLPs in a certain way. We need a few more definitions.

It is often useful to restrict adequate Turing-machines further. We say that an adequate Turing-machine M is *fully balanced* if for every input w, the computation tree $T_M(w)$ has the same number of branching nodes (i.e., nodes with exactly two children) along every maximal path. In other words, if we contract edges (v,v') such that v' is the unique child of v, then $T_M(w)$ becomes a full binary tree. Figure 2 shows a typical shape for the computation tree of a fully balanced adequate Turing-machine. For two binary strings $u = a_1 a_2 \cdots a_n$ and $v = b_1 b_2 \cdots b_n$ ($a_1, b_1, \ldots, a_n, b_n \in \{0,1\}$) we define the convolution $u \otimes v = (a_1,b_1)(a_2,b_2)\cdots(a_n,b_n)$. Finally, let $\rho : (\{0,1\} \times \{0,1\})^* \to \{0,1\}^*$ be the morphism with $\rho(0,0) = \rho(0,1) = \varepsilon$, $\rho(1,0) = 0$, and $\rho(1,1) = 1$. The following result was shown in [87]:

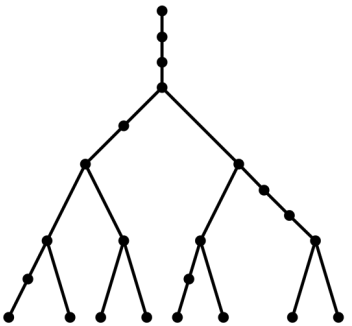

Figure 2. A computation tree of a fully balanced adequate Turing-machine.

Theorem 14 ([87]). *Let M be a fully balanced adequate polynomial time Turing-machine such that for some polynomial $p(n)$ and for every input w, every maximal path in the computation tree $T_M(w)$ has exactly $p(|w|)$ many branching nodes. From a given input $w \in \Sigma^*$ for M we can construct in logspace two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that $|\mathrm{eval}(\mathbb{A})| = |\mathrm{eval}(\mathbb{B})|$ and $\mathsf{leaf}(M, w) = \rho(\mathrm{eval}(\mathbb{A}) \otimes \mathrm{eval}(\mathbb{B}))$.*

The proof of Theorem 14 can be seen as a refinement of the construction from Example 3. It is based on an encoding of the subsetsum problem. In the following sections, we will see two applications of Theorem 14.

8 Compressed problems for subsequences

In many applications in computational biology, approximate occurrences of a pattern in a text are more relevant than exact matches, see, e.g., [52]. A very simple formalization of an approximate occurrence is that of a *subsequence*. The string $p = a_1 \cdots a_n$ is a subsequence of $t \in \Sigma^*$, if $t \in \Sigma^* a_1 \Sigma^* \cdots a_n \Sigma^*$, see Figure 3. Note the difference between the notions of a factor and a subsequence. The *fully compressed subsequence matching problem* is defined as follows:

Input Two SLPs $\mathbb{P}$ and $\mathbb{T}$.

Question Is $\mathrm{eval}(\mathbb{P})$ a subsequence of $\mathrm{eval}(\mathbb{T})$?

When comparing fully compressed subsequence matching with fully compressed pattern matching, it is the more liberal notion of a pattern occurrence that makes the application of periodicity properties of words (in particular, the periodicity lemma of Fine and Wilf) impossible. Indeed it turned out in [81,87] that fully compressed subsequence matching is much harder than fully compressed pattern matching.

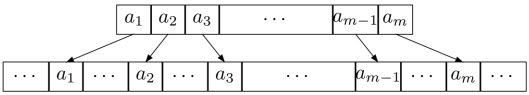

Figure 3. Subsequences.

Recall the definition of the complexity class PP (probabilistic polynomial time) from the previous section. This class contains computationally very difficult problems. A famous result of Toda [134] states that every language from the polynomial time hierarchy is polynomial time Turing-reducible to a language from PP, briefly: $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$. Moreover, PP contains the class $\mathsf{P}_{\parallel}^{\mathsf{NP}}$ (*parallel access to* NP), which consists of all problems that can be accepted by a deterministic polynomial time machine with access to an oracle from NP and such that furthermore all questions to the oracle are asked in parallel [109, 138].

Theorem 15 ([87]). *The fully compressed subsequence matching problem belongs to* PSPACE *and is* PP*-hard.*

Theorem 15 improves a result from [81] stating that the fully compressed subsequence matching problem is hard for the class $\mathsf{P}_{\parallel}^{\mathsf{NP}}$.

Membership of fully compressed subsequence matching in PSPACE is easy to prove. The straightforward greedy linear time algorithm for uncompressed strings becomes a polynomial space bounded algorithm for SLP-compressed strings. In polynomial space, we store a position $p_{\mathbb{P}}$ in the string $\mathrm{eval}(\mathbb{P})$ and a position $p_{\mathbb{T}}$ in the string $\mathrm{eval}(\mathbb{T})$ (initially, $p_{\mathbb{T}} = p_{\mathbb{P}} = 1$). In each step, we search for the smallest position $p \geq p_{\mathbb{T}}$ such that $\mathrm{eval}(\mathbb{T})[p] = \mathrm{eval}(\mathbb{P})[p_{\mathbb{P}}]$. If we find such a position, we set $p_{\mathbb{P}} := p_{\mathbb{P}} + 1$ and $p_{\mathbb{T}} := p + 1$.

Hardness of fully compressed subsequence matching for the class PP is derived in [87] from Theorem 14. It is easy to see that there exists a fixed fully balanced adequate polynomial time Turing-machine M for which it is PP-complete to check whether the number of accepting computations of M on a given input w is at least m (a given binary coded number). Let us fix an input w for M and a binary coded number m. By Theorem 14, we can compute in logspace two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that $|\mathrm{eval}(\mathbb{A})| = |\mathrm{eval}(\mathbb{B})|$ and $\mathsf{leaf}(M, w) = \rho(\mathrm{eval}(\mathbb{A}) \otimes \mathrm{eval}(\mathbb{B}))$. Let $n = |\mathrm{eval}(\mathbb{A})| = |\mathrm{eval}(\mathbb{B})| \geq 1$; the binary encoding of this number can be computed in logspace in this particular case. For the number m we can w.l.o.g. assume that $m \leq n$ (otherwise, there cannot be m accepting computations on input w).

We define two morphisms ϕ_1 and ϕ_2 as follows:

$$\phi_1(0) = 0^{n+1}, \qquad \phi_1(1) = (10)^n,$$
$$\phi_2(0) = 0(10)^n, \qquad \phi_2(1) = (10)^{n+1}.$$

It is straightforward to compute in logspace SLPs $\mathbb{C}$ and $\mathbb{D}$ such that

$$\text{eval}(\mathbb{C}) = \phi_1(\text{eval}(\mathbb{A}))(10)^m \quad \text{and} \quad \text{eval}(\mathbb{D}) = \phi_2(\text{eval}(\mathbb{B})).$$

Moreover, it is not too hard to show that $\text{eval}(\mathbb{C})$ is a subsequence of $\text{eval}(\mathbb{D})$ if and only if $\rho(\text{eval}(\mathbb{A}) \otimes \text{eval}(\mathbb{B}))$ contains at least m many 1's. The latter holds if and only if the number of accepting computations of M on input w is at least m. This concludes our proof sketch for the PP-hardness of fully compressed subsequence matching.

A corollary of the PP-hardness of fully compressed subsequence matching is PP-hardness of the *longest common subsequence problem* and the *shortest common supersequence problem* on SLP-compressed strings, even when both problems are restricted to two input strings. These problems have many applications, e.g., in computational biology [52]. Another interesting property of the fully compressed subsequence matching problem is that it can be reduced to its own complement [81]. Hence, the fully compressed subsequence matching problem and its complementary problem have the same complexity.

As for pattern matching, also the length of the pattern in subsequence matching is quite often small in comparison to the length of the text. In such a situation it makes sense to consider the complexity of checking whether an uncompressed pattern $p \in \Sigma^*$ is a subsequence of a compressed string $\text{eval}(\mathbb{T})$. Let us call this problem the *semi-compressed subsequence matching problem*. There is a straightforward algorithm for semi-compressed subsequence matching that works in time $O(|p| \cdot |\mathbb{T}|)$, see [132, 142]. Assume that the SLP $\mathbb{T}$ is in Chomsky normal form. Let $p = a_1 \cdots a_m$. The idea is to compute for every position $1 \leq i \leq m$ and every variable X of $\mathbb{T}$ the length $\ell(i, X)$ of the longest prefix of $a_i \cdots a_m$ that is a subsequence of $\text{eval}_{\mathbb{T}}(X)$. The values $\ell(i, X)$ satisfy a simple recursion, which allows to compute all these values in time $O(|p| \cdot |\mathbb{T}|)$. More complex problems related to semi-compressed subsequence matching (e.g., counting the number of shortest factors of $\text{eval}(\mathbb{T})$, which contain p as a subsequence) are considered in [22, 133, 142].

The structural complexity of a generalization of semi-compressed subsequence matching was studied by Markey and Schnoebelen [99]. LOGCFL is the class of all languages that are logspace reducible to a context-free language [131]; it coincides with the class of all languages that can be recognized by a uniform boolean circuit family of polynomial size and logarithmic depth, where all $\wedge$-gates have bounded

fan-in and all $\vee$-gates have unbounded fan-in [137]. It is conjectured that LOGCFL is a proper subclass of P.

Theorem 16 ([99]). *The following problem belongs to* LOGCFL*:*

Input *Strings* $p_0, p_1, \ldots, p_n \in \Sigma^*$ *and an SLP* $\mathbb{T}$ *over* Σ.
Question *Does* $\mathrm{eval}(\mathbb{T}) \in p_0\Sigma^* p_1 \Sigma^* \cdots p_{n-1}\Sigma^* p_n$ *hold?*

9 Compressed membership problems

The complexity of membership problems for various classes of formal languages is a classical topic at the borderline between formal language theory and complexity theory. It is therefore natural to consider membership problems in a compressed setting. For a language $L \subseteq \Sigma^*$, the *compressed membership problem* is the following decision problem:

Input An SLP $\mathbb{A}$ over the alphabet Σ.
Question Does $\mathrm{eval}(\mathbb{A}) \in L$ hold?

An immediate corollary of Theorem 13 is the following result:

Corollary 17. *For every language* $L \subseteq \{0,1\}^*$*, the compressed membership problem for the language* L *is complete for the logspace leaf language class* $\mathsf{LEAF}^{\log}(L)$.

If we fix some formalism for describing languages, we may also consider a uniform version of the compressed membership problem, where the language L is part of the input. In this section, we will survey results on the complexity of this problem for various kinds of automata, regular expression, and grammars.

9.1 Regular languages

Let us start with regular languages, the most basic languages in formal language theory.

Theorem 18 ([10, 99, 114]). *The following holds:*

- *For a given nondeterministic finite automaton* $\mathcal{A}$ *with* n *states and an SLP* $\mathbb{A}$ *it can be checked in time* $O(|\mathbb{A}| \cdot n^3)$ *whether* $\mathrm{eval}(\mathbb{A}) \in L(\mathcal{A})$.
- *There exists a fixed regular language with a* P*-complete compressed membership problem.*

The first statement is easy to show. Let $\mathbb{A}$ be an SLP in Chomsky normal form over the alphabet Σ and let $\mathcal{A}$ be a nondeterministic finite automaton with state set

Q and input alphabet Σ. Without loss of generality assume that $Q = \{1, \ldots, n\}$. The set of transition tuples of $\mathcal{A}$ can be represented by a bunch of boolean matrices $B_a \in \{0,1\}^{n \times n}$, one for each input symbol $a \in \Sigma$. Entry (i, j) of B_a is 1 if and only if there is an a-labelled transition from state i to state j. Clearly, there is a path from state i to state j that is labelled with the word $a_1 \cdots a_m$ ($a_1, \ldots, a_m \in \Sigma$) if and only if entry (i, j) of the product matrix $B_{a_1} B_{a_2} \cdots B_{a_m}$ is 1. This simple observation allows us to compute for each nonterminal C of the SLP $\mathbb{A}$ a boolean matrix B_C such that entry (i, j) of B_C is 1 if and only if there is an $\mathrm{eval}_{\mathbb{A}}(C)$-labelled path from state i to state j: If $C \to a \in \Sigma$ is a production of $\mathbb{A}$, we set $B_C = B_a$. If $C \to DE$ is a production of $\mathbb{A}$, then we set $B_C = B_D B_E$. If S is the initial nonterminal of $\mathbb{A}$, then $\mathrm{eval}(\mathbb{A}) \in L(\mathcal{A})$ if and only if in the matrix B_S there is a 1-entry at a position (i, j), where i is an initial state of $\mathcal{A}$ and j is a final state of $\mathcal{A}$. In this way, we can check whether $\mathrm{eval}(\mathbb{A}) \in L(\mathcal{A})$ using $|\mathbb{A}|$ many multiplications of boolean $(n \times n)$-matrices, which leads to the time bound $O(|\mathbb{A}| \cdot n^3)$. Actually, by using a better algorithm for boolean matrix multiplication than the standard n^3 school method, the time bound $O(|\mathbb{A}| \cdot n^3)$ can be improved. For about 20 years, the Coppersmith–Winograd algorithm was the asymptotically best algorithm for matrix multiplication with a complexity of $O(n^{2,3755})$. This bound was recently improved by Williams [136] to $O(n^{2,3727})$.

For a given *deterministic* finite automaton $\mathcal{A}$ with n states and an SLP $\mathbb{A}$, one can verify $\mathrm{eval}(\mathbb{A}) \in L(\mathcal{A})$ in time $O(|\mathbb{A}| \cdot n)$ [114]. Instead of $|\mathbb{A}|$ many boolean matrix multiplications, only $|\mathbb{A}|$ compositions of functions from $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$ are necessary in the deterministic case. In [12] the following generalization has been shown: From a given SLP $\mathbb{A}$ over an alphabet Σ and a deterministic finite state transducer τ with input alphabet Σ and n states, one can compute in time $O(|\mathbb{A}| \cdot n)$ an SLP $\mathbb{B}$ of size $O(|\mathbb{A}| \cdot n)$ for the output string $\tau(\mathrm{eval}(\mathbb{A}))$.

The second statement of Theorem 18 is stated in [99]. The proof is based on a result from [10], which states that there exists a fixed finite monoid M (in fact, any finite non-solvable group can be taken for M) such that the circuit evaluation problem for M is P-complete. The latter problem asks whether a given circuit over M (i.e., a circuit, where all input gates are labelled with constants from M and all internal gates are labelled with the multiplication operation of M) evaluates to a given element of M. A circuit over M can be viewed as an SLP that generates a word over the alphabet M. The second statement of Theorem 18 follows, since the set of all words over the alphabet M that evaluate to a particular element of M is a regular language. P-hardness of the circuit evaluation problem for a finite non-solvable group is shown in [10] using a technique of Barrington [9]. In this paper, Barrington proved that the word problem for every finite non-solvable group is complete for the circuit complexity class NC^1.

9.2 Compressed membership for regular expressions

When the regular language is not given by an automaton but by a regular expression, the complexity of the compressed membership problem depends on the operators that we allow in expressions. The same phenomenon is already well-known for uncompressed strings. For instance, whereas the membership problem for ordinary regular expressions (where only the Kleene-operators $\cdot, \cup, *$ are allowed) is NL-complete [68], membership for semi-extended regular expressions (where in addition also intersection is allowed) is LOGCFL-complete [110].

Let us be a bit more formal: For a set $\mathcal{C}$ of language operations, $\mathcal{C}$-expressions over the alphabet Σ are built up from constants in $\Sigma \cup \{\varepsilon\}$ using the operations from $\mathcal{C}$. Thus, ordinary regular expressions are $\{\cdot, \cup, *\}$-expressions. The language defined by ρ is $L(\rho)$. The length $|\rho|$ of an expression is defined as follows: For $\rho \in \Sigma \cup \{\varepsilon\}$ set $|\rho| = 1$. If $\rho = \mathrm{op}(\rho_1, \ldots, \rho_n)$, where op is an n-ary operator, we set $|\rho| = 1 + |\rho_1| + \cdots + |\rho_n|$. For a set $\mathcal{C}$ of language operations, the *compressed membership problem for* $\mathcal{C}$ is the following computational problem:

Input A $\mathcal{C}$-expression ρ and an SLP $\mathbb{A}$.

Question Does $\mathrm{eval}(\mathbb{A}) \in L(\rho)$ hold?

Beside the Kleene-operators $\cdot, \cup, *$, we also consider intersection ($\cap$), complement ($\neg$), squaring (^{2}, where $L^2 = \{uv \mid u, v \in L\}$), and shuffle ($\|$). The latter operator is defined as follows: For words $u, v \in \Sigma^*$ let

$$u \parallel v = \{u_0 v_0 u_1 v_1 \cdots u_n v_n \mid n \geq 0, u = u_0 \cdots u_n, v = v_0 \cdots v_n\}.$$

For $L, K \subseteq \Sigma^*$ let $L \parallel K = \bigcup_{u \in L, v \in K} u \parallel v$. It is well known that the class of regular languages is closed under $\cap, \neg, ^2$, and $\|$, but each of these operators leads to more succinct representations of regular languages. Operator sets that received particular interest in the literature are $\{\cdot, \cup, \neg\}$ (star-free expressions), $\{\cdot, \cup, *, \cap\}$ (semi-extended regular expressions), $\{\cdot, \cup, *, \neg\}$ (extended regular expressions), and $\{\cdot, \cup, *, \|\}$ (shuffle expressions).

The next theorem gives a rather complete picture on the complexity of compressed membership problems for regular expressions. It characterizes the complexity for all operator sets

$$\{\cdot, \cup\} \subseteq \mathcal{C} \subseteq \{\cdot, \cup, *, \cap, \neg, \|, ^2\}.$$

The class $\mathsf{ATIME}(\exp(n), O(n))$ denotes the class of all problems that can be solved on an alternating Turing-machine in exponential time, but the number of alternations (i.e., transitions between an existential and a universal state or vice versa) has to be bounded by $O(n)$ on every computation path. Completeness results for

$\mathsf{ATIME}(\exp(n), O(n))$ are typical for logical theories [28], but we are not aware of any other completeness results for this class in formal language theory.

Theorem 19 ([85]). *Let* $\{\cdot, \cup\} \subseteq \mathcal{C} \subseteq \{\cdot, \cup, *, \cap, \neg, \|, {}^2\}$. *The compressed membership problem for* $\mathcal{C}$ *is*

(1) *in* P *if* $\mathcal{C} \subseteq \{\cdot, \cup, \cap\}$ *or* $\mathcal{C} \subseteq \{\cdot, \cup, *\}$,

(2) NP*-complete if* $\{\cdot, \cup, \|\} \subseteq \mathcal{C} \subseteq \{\cdot, \cup, \cap, \|\}$,

(3) PSPACE*-complete if*

(i) $\neg \notin \mathcal{C}$ *or* $\| \notin \mathcal{C}$ *and*

(ii) $\mathcal{C}$ *contains* $\{\cdot, \cup, \neg\}$ *or* $\{\cdot, \cup, {}^2\}$ *or* $\{\cdot, \cup, *, \cap\}$ *or* $\{\cdot, \cup, *, \|\}$,

(4) $\mathsf{ATIME}(\exp(n), O(n))$*-complete if* $\{\cdot, \cup, \neg, \|\} \subseteq \mathcal{C}$.

The first statement (membership in P for $\mathcal{C} \subseteq \{\cdot, \cup, \cap\}$ or $\mathcal{C} \subseteq \{\cdot, \cup, *\}$) is clear: The case $\mathcal{C} \subseteq \{\cdot, \cup, *\}$ reduces to the uniform compressed membership problem for nondeterministic finite automata (since $\{\cdot, \cup, *\}$-expressions can be transformed in polynomial time into nondeterministic finite automata), see Theorem 18. The case $\mathcal{C} \subseteq \{\cdot, \cup, \cap\}$ is also clear, since for a $\{\cdot, \cup, \cap\}$-expression ρ the length of every word in $L(\rho)$ is bounded by $|\rho|$. Hence, the compressed membership problem for $\{\cdot, \cup, \cap\}$ reduces to the (uncompressed) membership problem for $\{\cdot, \cup, \cap\}$-expressions, which can be solved in polynomial time. By the same argument, the compressed membership problem for $\{\cdot, \cup, \cap, \|\}$ reduces to the (uncompressed) membership problem for $\{\cdot, \cup, \cap, \|\}$-expressions. But the latter problem is known to be NP-complete; in fact this already holds for $\{\cdot, \|\}$-expressions [103, 140], which yields statement (2) of Theorem 19. The upper complexity bounds in statement (3) and (4) of Theorem 19 can be shown by rather straightforward algorithms (using for (3) the well-known correspondence between PSPACE and alternating polynomial time).

Let us now turn to the lower bounds in (3) and (4) from Theorem 19. PSPACE-hardness for $\{\cdot, \cup, \neg\}$ is shown in [85] by a reduction from the quantified boolean satisfiability problem; the reduction is mainly taken from [98], where it was shown that model checking the temporal logic LTL on SLP-compressed strings is PSPACE-complete. PSPACE-hardness for $\{\cdot, \cup, {}^2\}$ and $\{\cdot, \cup, *, \cap\}$ is shown by generic reductions from the acceptance problem for a polynomial space bounded machine. Finally, the PSPACE lower bound for $\{\cdot, \cup, *, \|\}$ is proved by a reduction from the intersection non-emptiness problem for regular expressions (over the standard operator set $\{\cdot, \cup, *\}$), which is a well-known PSPACE-complete problem [76]. The $\mathsf{ATIME}(\exp(n), O(n))$-hardness part from (4) uses a corresponding result from [88] on the model-checking problem for monadic second-order logic over SLP-compressed words over a unary alphabet.

9.3 Compressed finite automata

A *compressed nondeterministic finite automaton* (CNFA for short) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is a finite set of transitions of the form $(p, \mathbb{A}, q)$, where p and q are states and $\mathbb{A}$ is an SLP over Σ. The size of $\mathcal{A}$ is $|\mathcal{A}| = |Q| + \sum_{(p,\mathbb{A},q)\in\delta} |\mathbb{A}|$. We say that a word w labels a path from state p to state q in $\mathcal{A}$ if there exists a sequence of transitions

$$(p_0, \mathbb{A}_0, p_1), (p_1, \mathbb{A}_1, p_2), \ldots, (p_{n-1}, \mathbb{A}_{n-1}, p_n) \in \delta$$

($n \geq 0$) such that $p_0 = p$, $p_n = q$, and $w = \mathrm{eval}(\mathbb{A}_0) \cdots \mathrm{eval}(\mathbb{A}_{n-1})$. The language $L(\mathcal{A}) \subseteq \Sigma^*$ is the set of all words that label a path from the initial state q_0 to some final state $q_f \in F$. A *compressed deterministic finite automaton* (CDFA for short) is a CNFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ such that for every state $p \in Q$, and all transitions $(p, \mathbb{A}_1, q_1), (p, \mathbb{A}_2, q_2) \in \delta$, neither $\mathrm{eval}(\mathbb{A}_1)$ is a prefix of $\mathrm{eval}(\mathbb{A}_2)$, nor $\mathrm{eval}(\mathbb{A}_2)$ is a prefix of $\mathrm{eval}(\mathbb{A}_1)$ The *compressed membership problem for CNFAs (resp., CDFAs)* is the following decision problem:

Input A CNFA (resp., CDFA) $\mathcal{A}$ and an SLP $\mathbb{A}$.

Question Does $\mathrm{eval}(\mathbb{A}) \in L$ hold?

Theorem 20 ([65]). *The compressed membership problem for CNFAs is* NP*-complete, whereas the compressed membership problem for CDFAs is* P*-complete.*

In his proof of Theorem 20, Jez developed his recompression technique that he later applied in [66] to get his $O((|\mathbb{T}| + |\mathbb{P}|)\log(M)\log(|\mathbb{T}| + |\mathbb{P}|))$-algorithm for fully compressed pattern matching, see Section 6.

The *order* of a word u is the largest number n such that u can be written as $u = v^n w$, where w is a prefix of v. If $\mathbb{A}_1, \ldots, \mathbb{A}_n$ is a list of all SLPs that occur as labels in the CNFA $\mathcal{A}$, then we set

$$\mathsf{ord}(\mathcal{A}) = \max\{\mathsf{ord}(\mathrm{eval}(\mathbb{A}_i)) \mid 1 \leq i \leq n\}.$$

Note that $\mathsf{ord}(\mathcal{A})$ is in general exponential in the size of $\mathcal{A}$. By the following result, the compressed membership problem for CNFAs $\mathcal{A}$ with small $\mathsf{ord}(\mathcal{A})$ can be solved efficiently.

Theorem 21 ([92]). *Given a CNFA $\mathcal{A}$ and an SLP $\mathbb{A}$, we can check* $\mathrm{eval}(\mathbb{A}) \in L(\mathcal{A})$ *in time polynomial in* $|\mathbb{A}|$, $|\mathcal{A}|$, *and* $\mathsf{ord}(\mathcal{A})$.

9.4 Context-free languages

In this section, we consider the compressed membership problem for context-free languages. It is easy to see that for every context-free language the compressed membership problem can be solved in polynomial space. This result even holds in a uniform setting, where the input consists of a context-free grammar G and an SLP $\mathbb{A}$, and it is asked whether $\mathrm{eval}(\mathbb{A}) \in L(G)$:

Theorem 22 ([84]). *For a given context-free grammar G and an SLP $\mathbb{A}$ it can be checked in polynomial space whether* $\mathrm{eval}(\mathbb{A}) \in L(G)$.

First, one has to transform the grammar G into Chomsky normal form; this is possible in polynomial time. By [47] the uniform (uncompressed) membership problem for context-free grammars in Chomsky normal form can be solved in $\mathsf{DSPACE}(\log^2(|G| + |w|))$, where $|G|$ is the size of the input grammar (in Chomsky normal form) and w is the word that has to be tested for membership. This algorithm can be used to check for a given context-free grammar G in Chomsky normal form and an SLP $\mathbb{A}$ in $\mathsf{DSPACE}(\log^2(|G| + 2^{|\mathbb{A}|}))$, i.e., in polynomial space, whether $\mathrm{eval}(\mathbb{A}) \in L(G)$.

In [84], a fixed deterministic context-free language with a PSPACE-complete compressed membership problem was constructed using a generic reduction from the acceptance problem for polynomial space machines. Alternatively, the existence of such a language can also be deduced from Corollary 17 and the following result from [64]: There is a deterministic context-free language L such that $\mathsf{PSPACE} = \mathsf{LEAF}^{\log}(L)$. In [21], this result was sharpened: There even exists a 1-turn deterministic context-free language L such that $\mathsf{PSPACE} = \mathsf{LEAF}^{\log}(L)$. A deterministic pushdown automaton P is 1-turn, if for every input, the unique computation of P can be divided into two phases: In a first phase, the height of the pushdown does not decreases in every step, whereas in the second phase, the height of the pushdown does not increase in every step. Moreover, in [21] also a deterministic 1-counter language L (i.e., a context-free language that is accepted by a deterministic pushdown automaton with a unary pushdown alphabet) with $\mathsf{PSPACE} = \mathsf{LEAF}^{\log}(L)$ is constructed. Hence, there also exist fixed 1-turn deterministic context-free languages and deterministic 1-counter languages with a PSPACE-complete compressed membership problem.

In [87] it was shown that even the important subclass of visibly pushdown languages [6] contains languages with a PSPACE-complete compressed membership problem. Let us first define visibly pushdown automata and the associated languages. Let Σ_c and Σ_r be two disjoint finite alphabets (call symbols and return symbols) and let $\Sigma = \Sigma_c \cup \Sigma_r$. A *visibly pushdown automaton* (VPA) [6] over (Σ_c, Σ_r) is a tuple $V = (Q, q_0, \Gamma, \bot, \Delta, F)$, where Q is a finite set of states,

$q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, Γ is the finite set of stack symbols, $\bot \in \Gamma$ is the initial stack symbol, and

$$\Delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\bot\})) \cup (Q \times \Sigma_r \times \Gamma \times Q)$$

is the set of transitions.[3] A configuration of V is a triple from $Q \times \Sigma^* \times (\Gamma \setminus \{\bot\})^*\bot$. For two configurations (p, au, v) and (q, u, w) (with $a \in \Sigma$, $u \in \Sigma^*$) we write $(p, au, v) \Rightarrow_V (q, u, w)$ if one of the following three cases holds:

- $a \in \Sigma_c$ and $w = \gamma v$ for some $\gamma \in \Gamma$ with $(p, a, q, \gamma) \in \Delta$,
- $a \in \Sigma_r$ and $v = \gamma w$ for some $\gamma \in \Gamma$ with $(p, a, \gamma, q) \in \Delta$,
- $a \in \Sigma_r$, $u = v = \bot$, and $(p, a, \bot, q) \in \Delta$.

The language $L(V)$ is defined as

$$L(V) = \{w \in \Sigma^* \mid \exists f \in F, u \in (\Gamma \setminus \{\bot\})^*\bot : (q_0, w, \bot) \Rightarrow_V^* (f, \varepsilon, u)\}.$$

The VPA V is deterministic if for every $p \in Q$ and $a \in \Sigma$ the following hold:

- If $a \in \Sigma_c$, then there is at most one pair $(q, \gamma) \in Q \times \Gamma$ with $(p, a, q, \gamma) \in \Delta$.
- If $a \in \Sigma_r$, then for every $\gamma \in \Gamma$ there is at most one $q \in Q$ with $(p, a, \gamma, q) \in \Delta$.

A VPA V is called a 1-turn VPA, if $L(V) \subseteq \Sigma_c^* \Sigma_r^*$. In this case $L(V)$ is called a *linear visibly pushdown language*.

Visibly pushdown automata and their languages have many decidability results and closure results that do not hold for general context-free languages. For instance, for every VPA V there exists a deterministic VPA V' with $L(V) = L(V')$, the class of visibly pushdown languages is effectively closed under boolean operations, and equivalence, inclusion, and intersection non-emptiness are decidable for VPAs, see [6] where also precise complexity results are shown. Moreover, also the membership problem seems to be easier for visibly pushdown languages than for general context-free languages. By a classical result from [50], there exists a context-free language with a LOGCFL-complete membership problem. For visibly pushdown languages the complexity of the membership problem decreases to the circuit complexity class NC^1 [34] and is therefore of the same complexity as for regular languages [9] (complexity theorists conjecture that NC^1 is a proper subclass of LOGCFL). In contrast to this, by the following theorem, compressed membership is in general PSPACE-complete even for linear visibly pushdown languages, whereas it is P-complete for regular languages (Theorem 18):

[3] In [6], the input alphabet may also contain internal symbols, on which the automaton does not touch the stack at all. For our lower bound, we will not need internal symbols.

Theorem 23 ([87]). *There is a fixed 1-turn visibly pushdown language with a* PSPACE*-complete compressed membership problem.*

This result can be deduced from Theorem 14. By the main result of [59] there exists a regular language $K \subseteq \{0,1\}^*$ such that for every language L in PSPACE there exists a fully balanced adequate polynomial time Turing-machine M with the following properties:[4]

- There is a polynomial $p(n)$ such that for every input w, every maximal path in the computation tree $T_M(w)$ has exactly $p(|w|)$ many branching nodes.
- For every input w, $w \in L$ if and only if $\mathsf{leaf}(M, w) \in K$.

Hence, by Theorem 14, the following problem is PSPACE-hard:

Input Two SLPs $\mathbb{A}$ and $\mathbb{B}$ over the terminal alphabet $\{0,1\}$ with $|\mathrm{eval}(\mathbb{A})| = |\mathrm{eval}(\mathbb{B})|$.

Question Does $\mathrm{eval}(\mathbb{A}) \otimes \mathrm{eval}(\mathbb{B}) \in \rho^{-1}(K)$ hold?

Since the class of regular languages is closed under inverse morphisms, the language $\rho^{-1}(K) \subseteq (\{0,1\} \times \{0,1\})^*$ is regular too. Fix a deterministic finite automaton D for $\rho^{-1}(K)$. Let $\overline{0}, \overline{1}$ be two new symbols. For a word $w = a_1 \cdots a_n$ with $a_1, \ldots, a_n \in \{0,1\}$ let $\overline{w} = \overline{a}_n \cdots \overline{a}_1$. It remains to construct a 1-turn visibly pushdown automaton P such that

$$L(P) = \{\overline{u}v \mid u, v \in \{0,1\}^*, |u| = |v|, u \otimes v \in L(D)\}$$

(then, we get $\mathrm{eval}(\mathbb{A}) \otimes \mathrm{eval}(\mathbb{B}) \in \rho^{-1}(K)$ if and only if $\overline{\mathrm{eval}(\mathbb{A})}\mathrm{eval}(\mathbb{B}) \in L(P)$). This is straightforward: While reading a symbol from $\{\overline{0}, \overline{1}\}$, the visibly pushdown automaton stores the symbol on the stack. When reading a symbol $a \in \{0,1\}$, the automaton checks if the topmost stack symbol is $\overline{a}$. If not, the automaton blocks, otherwise it pops $\overline{a}$ from the stack.

10 Querying compressed strings

One of the simplest questions one can ask about a given string is whether a certain position in the string carries a certain symbol. Let us formally define the *compressed querying problem* as follows:

[4] The proof of the second result uses again Barrington's technique mentioned at the end of Section 9.1.

Input An SLP $\mathbb{A}$ over an alphabet Σ, a binary-coded number $1 \leq i \leq |\mathrm{eval}(\mathbb{A})|$, and a symbol $a \in \Sigma$.

Question Does $\mathrm{eval}(\mathbb{A})[i] = a$ hold?

Theorem 24 ([81]). *Compressed querying is* P*-complete.*

We already argued in Section 2.2 that compressed querying can be solved in polynomial time. P-completeness of compressed querying is shown in [81] by a reduction from the P-complete *super increasing subsetsum problem* [71], which is the following problem:

Input Binary coded natural numbers $t, w_1, \ldots, w_n$ such that $w_k < \sum_{i=1}^{k-1} w_i$ for all $1 \leq k \leq n$.

Question Do there exist $b_1, \ldots, b_n \in \{0, 1\}$ such that $t = \sum_{i=1}^{n} b_i w_i$?

Assume that $w_1, \ldots, w_n$ are binary coded natural numbers such that $w_k < \sum_{i=1}^{k-1} w_i$ for all $1 \leq k \leq n$. In [81], it is shown how to construct in logspace an SLP $\mathbb{A}$ over the alphabet $\{a, b\}$ such that $|\mathrm{eval}(\mathbb{A})| = 1 + \sum_{i=1}^{n} w_i$ and for all $0 \leq m \leq \sum_{i=1}^{n} w_i$ the following holds: $\mathrm{eval}(\mathbb{A})[m + 1] = b$ if and only if there exist $b_1, \ldots, b_n \in \{0, 1\}$ such that $m = \sum_{i=1}^{n} b_i w_i$. The construction is very similar to those from Example 3.

P-completeness of compressed querying does not imply that there do not exist highly efficient polynomial time algorithms for this problem. When considering efficient algorithms for compressed querying it makes sense to distinguish between *preprocessing time* and *query time*: During preprocessing time, an auxiliary data structure is constructed from the input SLP $\mathbb{A}$ that allows to compute in a second phase for any given binary coded number $1 \leq i \leq |\mathrm{eval}(\mathbb{A})|$ the symbol $\mathrm{eval}(\mathbb{A})[i]$ within the query time bound. Hence, the preprocessing phase is executed only once.

The straightforward algorithm (see Section 2.2) that first computes the lengths of all strings generated by nonterminals of $\mathbb{A}$ and then walks down the derivation tree of $\mathbb{A}$ needs

- preprocessing time $\Theta(|\mathbb{A}|)$ in order to compute the lengths of all strings generated by nonterminals of $\mathbb{A}$, followed by
- query time $\Theta(h)$, where h is the height of $\mathbb{A}$, for walking down the derivation tree of $\mathbb{A}$.

As usual we assume here the RAM model with uniform cost measure. Note that the extra space in order to store the lengths of all strings generated by nonterminals of $\mathbb{A}$ is $\Theta(|\mathbb{A}|)$.

The query time of this algorithm can be reduced from $\Theta(h)$ to $\Theta(\log(n))$ by making the SLP $\mathbb{A}$ balanced using the technique from [118]. As mentioned in Section 4, this needs (preprocessing) time $O(|\mathbb{A}| \cdot \log(n))$; moreover, the best bound for the size of the resulting balanced SLP is $O(|\mathbb{A}| \cdot \log(n))$. Hence, the extra space increases from $\Theta(|\mathbb{A}|)$ to $O(|\mathbb{A}| \cdot \log(n))$. The following result from [15] combines a query time of $O(\log(n))$ with a preprocessing time and extra space of $O(|\mathbb{A}|)$:

Theorem 25 ([15]). *From a given SLP $\mathbb{A}$ such that $n = |\mathrm{eval}(\mathbb{A})|$ one can compute on a RAM in (preprocessing) time $O(|\mathbb{A}|)$ a data structure of size $O(|\mathbb{A}|)$ that allows to compute in (query) time $O(\log(n))$ the symbol $\mathrm{eval}(\mathbb{A})[i]$ for a given binary coded number $1 \leq i \leq n$.*

Some lower bounds on the preprocessing and query time for the compressed querying problem can be found in [24].

11 Compressed word problems for groups

In recent years, techniques for SLP-compressed strings were successfully applied in order to get more efficient algorithms for problems in combinatorial group theory. In this section, we briefly survey some of these results. We will restrict our consideration to the compressed word problem and its application to the word problem for automorphism groups. Background in combinatorial group theory can be found for instance in [95, 130].

Let G be a finitely generated group, and let Σ be a finite generating set for G. This means that every element of G can be written as a product of elements from $\Sigma \cup \Sigma^{-1}$. The *word problem* of G with respect to Σ is the following decision problem:

Input A word $w \in (\Sigma \cup \Sigma^{-1})^*$.

Question Does $w = 1$ hold in the group G?

It is well known and easy to see that if Γ is another finite generating set for G, then the word problem for G with respect to Σ is logspace many-one reducible to the word problem for G with respect to Γ. This justifies to speak just of the word problem for the group G.

The word problem was introduced in the pioneering work of Dehn from 1910 [31] in relation with topological questions. Novikov [108] and independently Boone [16] constructed examples of finitely presented groups (i.e., finitely generated groups with only finitely many defining relations) with an undecidable word

problem. Despite this negative result, many natural classes of groups with decidable word problems were found. Prominent examples are for instance finitely generated linear groups, automatic groups [35], and one-relator groups. With the advent of computational complexity theory, also the complexity of word problems became an active research area. For instance, it was shown that for a finitely generated linear group the word problem can be solved in logarithmic space [83, 129], that automatic groups have polynomial time solvable (in fact quadratic) word problems [35], and that the word problem for a one-relator group is primitive recursive [19].

For a group G let $\mathrm{Aut}(G)$ be the automorphism group of G. In general, $\mathrm{Aut}(G)$ is not necessarily finitely generated even if G is finitely generated. For the investigation of the complexity of the word problem of a finitely generated subgroup of $\mathrm{Aut}(G)$, a compressed variant of the word problem for G turned out to be useful. Assume again that G is finitely generated by Σ. The *compressed word problem* of G with respect to Σ is the following computational problem:

Input An SLP $\mathbb{A}$ over the alphabet $\Sigma \cup \Sigma^{-1}$.

Question Does $\mathrm{eval}(\mathbb{A}) = 1$ hold in G?

It is easy to see that also for the compressed word problem the complexity does not depend on the chosen generating set, which allows to speak of the compressed word problem for the group G. The compressed word problem for G can be also viewed as the following question: Does a given circuit over G evaluate to the identity element of G. One of the main applications of the compressed word problem to "non-compressed" decision problems is the following result:

Theorem 26 ([94, 124]). *Let H be a finitely generated subgroup of* $\mathrm{Aut}(G)$. *Then the word problem for H is logspace many-one reducible to the compressed word problem for G.*

The proof of Theorem 26 is quite simple. Given a sequence of H-generators (and hence automorphisms of G) $\varphi_1, \varphi_2, \ldots, \varphi_n$, it has to be checked whether the composition $\varphi = \varphi_1 \circ \varphi_2 \circ \cdots \circ \varphi_n$ is the identity mapping on G. But this is equivalent to $\varphi(a) = a$ for each generator $a \in \Sigma$ of G. Now, one can easily construct in logspace from the sequence $\varphi_1, \varphi_2, \ldots, \varphi_n$ and a generator $a \in \Sigma$ an SLP $\mathbb{A}_a$ over the alphabet $\Sigma \cup \Sigma^{-1}$ such that $\mathrm{eval}(\mathbb{A}_a)$ represents the group element $\varphi(a)$. Hence, one has to check for all $a \in \Sigma$, whether $\mathrm{eval}(\mathbb{A}_a) = a$ in the group G, which is an instance of the compressed word problem for G.

Clearly, for a finite group the compressed word problem can be solved in polynomial time. The following theorem is an immediate corollary of the results from [10].

Theorem 27 ([10]). *Let G be a finite group.*

- *If G is not solvable, then the compressed word problem for G is* P*-complete.*
- *If G is solvable, then the compressed word problem for G belongs to the class* $\mathsf{DET} \subseteq \mathsf{NC}^2$.[5]

So, for instance, the compressed word problem for the symmetric group on 5 elements (a non-solvable group) is P-complete.

Compressed word problems for infinite groups were considered for the first time in [84]. For free groups, the following result was shown:

Theorem 28 ([84]). *The compressed word problem for a finitely generated free group F of rank r can be solved in polynomial time. If $r > 1$ then this problem is* P*-complete.*

The proof for P-hardness for $r > 1$ uses a construction of Robinson [116], where it is shown that the (uncompressed) word problem for the free group of rank 2 is hard for the circuit complexity class NC^1. Let us sketch a polynomial time algorithm for the compressed word problem for a finitely generated free group F. Let Σ be a free set of generators for F and let $\mathbb{A}$ be an SLP over the terminal alphabet $\Sigma \cup \Sigma^{-1}$. By Theorem 4 it suffices to construct a composition system $\mathbb{B}$ over the same alphabet that generates the reduced normal form of $\mathrm{eval}(\mathbb{A})$. This normal form is obtained by iteratively deleting occurrences of subwords aa^{-1} with $a \in \Sigma \cup \Sigma^{-1}$ from $\mathrm{eval}(\mathbb{A})$ (the order in which cancelation is done is not relevant). The composition system $\mathbb{B}$ contains for each variable X of $\mathbb{A}$ a variable X' such that $\mathrm{eval}_{\mathbb{B}}(X')$ is the reduced normal form of $\mathrm{eval}_{\mathbb{A}}(X)$. Let $X \to YZ$ be a production of $\mathbb{A}$ and assume that $\mathbb{B}$ already contains enough productions such that $\mathrm{eval}_{\mathbb{B}}(Y')$ (resp., $\mathrm{eval}_{\mathbb{B}}(Z')$) is the reduced normal form of $\mathrm{eval}_{\mathbb{A}}(Y)$ (resp., $\mathrm{eval}_{\mathbb{A}}(Z)$). Let $\mathrm{eval}_{\mathbb{B}}(Y') = a_1 a_2 \cdots a_m$ and $\mathrm{eval}_{\mathbb{B}}(Z') = b_1 b_2 \cdots b_k$. In the concatenation $\mathrm{eval}_{\mathbb{B}}(Y')\mathrm{eval}_{\mathbb{B}}(Z')$ cancellation may only occur at the border between $\mathrm{eval}_{\mathbb{B}}(Y')$ and $\mathrm{eval}_{\mathbb{B}}(Z')$. We have to compute the length n of the longest suffix of $\mathrm{eval}_{\mathbb{B}}(Y')$ that cancels against the length-n prefix of $\mathrm{eval}_{\mathbb{B}}(Z')$, i.e., we compute the maximal number n such that

$$\begin{aligned}(\mathrm{eval}_{\mathbb{B}}(Y')[m-n+1,m])^{-1} &= a_m^{-1} a_{m-1}^{-1} \cdots a_{m-n+1}^{-1} \\ &= b_1 b_2 \cdots b_n = (\mathrm{eval}_{\mathbb{B}}(Z'))[1,n]. \qquad (1)\end{aligned}$$

For a particular n we can check equation (1) in polynomial time by the extension of Theorem 9 to composition systems (which holds by Theorem 4). Now,

[5] DET is the class of all problems that are NC^1-reducible to the problem of computing the determinant of a given integer matrix.

the maximal n satisfying (1) can be computed in polynomial time as well using binary search. Finally, we add to the composition system $\mathbb{B}$ the production $X' \to Y'[1, m-n]Z'[n+1, k]$.

A direct corollary of Theorem 26 and 28 is the following result, which solves an open problem from [70].

Corollary 29 ([124]). *Let F be a finitely generated free group. The word problem for* $\mathrm{Aut}(F)$ *(the automorphism group of F, which is finitely generated) can be solved in polynomial time.*

Several closure properties for the complexity of the compressed word problem where shown:

(1) If H is a finitely generated subgroup of G and the compressed word problem for G can be solved in polynomial time, then the same holds for H [94].

(2) If G is a finite-index subgroup of H and the compressed word problem for G can be solved in polynomial time, then the same holds for H [94].

(3) Let A and B be finite isomorphic subgroups of the group G, let $\varphi : A \to B$ be an isomorphism, and assume that the compressed word problem for G can be solved in polynomial time. Then, also the compressed word problem for the HNN-extension $H = \langle G, t; t^{-1}at = \varphi(a)(a \in A)\rangle$ can be solved in polynomial time [56].

(4) Let G and H be groups, which both contain the finite subgroup A. If the compressed word problems of G and H can be solved in polynomial time, then also the compressed word problem for the amalgamated free product $G *_A H$ can be solved in polynomial time [56].

(5) Let $G_1, G_2, \ldots, G_n$ be groups and let I be a symmetric and irreflexive binary relation on the set $\{1, \ldots, n\}$. The *graph product* $\mathsf{GP}(G_1, G_2, \ldots, G_n, I)$ results from the free product $G_1 * G_2 * \cdots * G_n$ by taking the quotient with respect to the commutation relations $ab = ba$ for all $a \in G_i$ and $b \in G_j$ with $(i, j) \in I$. Assume that the compressed word problems for the groups $G_1, \ldots, G_n$ can be solved in polynomial time. Then, also the compressed word problem for the graph product $\mathsf{GP}(G_1, G_2, \ldots, G_n, I)$ can be solved in polynomial time [57].

In all these results, polynomial time can be replaced by any complexity class that is closed under polynomial time Turing-reductions.

The results (1)–(5) yield polynomial time algorithms for a quite large class of groups. First of all, (5) implies by taking $G_1 = G_2 = \cdots = G_n = \mathbb{Z}$ that the compressed word problems for right-angled Artin groups (which are also known

as graph groups or free partially commutative groups) can be solved in polynomial time. Due to their rich subgroup structure, right-angled Artin groups received a lot of attention in group theory during the last few years [14, 29, 45]. With (1) and (5) it follows that for every group G which virtually embeds into a right-angled Artin group (i.e., G has a finite index subgroup that embeds into a right-angled Artin group) the compressed word problem can be solved in polynomial time. Groups that virtually embed into a right-angled Artin group are also known as *virtually special*. Recently, it has been shown that virtually special groups form a quite large class of groups. For instance, every Coxeter group is virtually special [55] and every fully residually free group is virtually special [141].[6] It follows that for every Coxeter group as well as for every fully residually free group the compressed word problem can be solved in polynomial time. For fully residually free groups, this result was directly shown by Macdonald [96] using a generalization of the technique for free groups.

For finitely generated linear groups, the word problem can be solved in logarithmic space [83, 129]. We currently do not know whether also for these groups the compressed word problem can be solved in polynomial time. The best we can achieve is a randomized polynomial time algorithm for the complement of the compressed word problem. More precisely, a language L belongs to the complexity class RP (randomized polynomial time) if there exists a randomized polynomial time algorithm A such that:

- if $x \notin L$ then Prob[A accepts x] $= 0$,
- if $x \in L$ then Prob[A accepts x] $\geq 1/2$.

The choice of the failure probability $1/2$ in case $x \in L$ is arbitrary: By repeating the algorithm c times (where c is some constant), we can reduce the failure probability to $(1/2)^c$ and still have a randomized polynomial time algorithm. A language L belongs to the class coRP, if the complement of L belongs to RP. This means that there exists a randomized polynomial time algorithm A such that:

- if $x \notin L$ then Prob[A accepts x] $\leq 1/2$,
- if $x \in L$ then Prob[A accepts x] $= 1$.

Theorem 30 ([94]). *If G is a finitely generated linear group, then the compressed word problem for G belongs to* coRP.

The idea for the proof of Theorem 30 is to reduce (by using results from [83, 129]) the compressed word problem for a finitely generated linear group G to the

[6] A group G is fully residually free, if for every tuple $(g_1, \ldots g_n)$ of elements from $G \setminus \{1\}$ there exists a free group F and a homomorphism $\varphi : G \to F$ such that $\varphi(g_i) \neq 1$ for all $1 \leq i \leq n$.

problem, whether a circuit over a polynomial ring $\mathbb{Z}[x_1, \ldots, x_n]$ or $\mathbb{F}_p[x_1, \ldots, x_n]$ (for p a prime) evaluates to the zero polynomial. This problem is known as *algebraic identity testing*. It belongs to coRP by [63]. Whether algebraic identity testing belongs to P is a major open problem in complexity theory, see [1] for a survey.

It also makes sense to consider the compressed word problem for a finitely generated monoid M. In that case, the input consists of two SLPs $\mathbb{A}$ and $\mathbb{B}$ over a generating set of M, and it is asked, whether $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$ in M. For instance, Theorem 9 says that the compressed word problem for a finitely generated free monoid can be solved in polynomial time. In [86], it was shown that the compressed word problem for a finitely generated free inverse monoid of rank $r \geq 2$ is complete for Π_p^2, which is the universal second level of the polynomial time hierarchy. It can be defined as the class of all languages $L \subseteq \Sigma^*$ such that there exists a polynomial time set $P \subseteq \Sigma^* \times \{0,1\}^* \times \{0,1\}^*$ and a polynomial $p(x)$ such that for all $u \in \Sigma^*$ we have ($\{0,1\}^{\leq m}$ denotes the set of binary words of length at most m):

$$u \in L \iff \forall v \in \{0,1\}^{\leq p(|u|)}\ \exists w \in \{0,1\}^{\leq p(|u|)} : (u, v, w) \in P$$

The class Π_2^p contains both NP and coNP.

Recently, power circuits [106], which are a compression scheme for representing huge integers (of multiply exponential size), have been used for solving the word problem for the one-relator Baumslag group and Higman's group [33].

12 Word equations

Let Σ be a finite alphabet and $\mathcal{X}$ be a finite set of variables. A word equation over $(\Sigma, \mathcal{X})$ is a pair (u, v) with $u, v \in (\Sigma \cup \mathcal{X})^*$. For a mapping $\sigma : \mathcal{X} \to \Sigma^*$ let $\widehat{\sigma} : (\Sigma \cup \mathcal{X})^* \to \Sigma^*$ be the unique homomorphism with $\widehat{\sigma}(a) = a$ for $a \in \Sigma$ and $\widehat{\sigma}(x) = \sigma(x)$. A solution of the word equation (u, v) is a mapping $\sigma : \mathcal{X} \to \Sigma^*$ such that $\widehat{\sigma}(u) = \widehat{\sigma}(v)$. In the following, we write a word equation (u, v) as $u = v$. A system of word equations over $(\Sigma, \mathcal{X})$ is a conjunction $\mathcal{S} = \bigwedge_{i=1}^{n} u_i = v_i$ of word equations over $(\Sigma, \mathcal{X})$. If σ is a solution of every word equation $u_i = v_i$, then σ is called a solution of $\mathcal{S}$. Standard arguments show that from a given system of word equations $\mathcal{S}$ one can compute (in logspace) a single word equation $u = v$ such that the set of solutions of $u = v$ is equal to the set of solutions of the system $\mathcal{S}$, see, e.g., [32].

Deciding whether a given word equation has a solution turned out to be very difficult. In the 1950s, Markov conjectured that it is undecidable whether a given word equation has a solution. He knew that solvability of word equations can be

reduced to Hilbert's 10th problem, and his goal was to prove the undecidability of Hilbert's 10th problem by proving undecidability of solvability of word equations. Whereas Hilbert's 10th problem was finally shown to be undecidable by the seminal work of Matiyasevich from 1971, solvability of word equations was shown to be decidable by Makanin [97] in 1977. Makanin's algorithm is very complicated, both with respect to its computational complexity and its technical difficulty. Over the years, the estimate on the time and space complexity of Makanin's algorithm was gradually improved. Currently, the best known bound is EXPSPACE [53]. In 1999, Plandowski came up with an alternative algorithm for checking solvability of word equations, which put the problem into PSPACE [112]. This is still the best upper complexity bound. The best known lower bound is NP, and it was repeatedly conjectured that the precise complexity is NP too.

Now, consider the situation, where for every variable $x \in \mathcal{X}$ a length-constraint $\ell(x) \in \mathbb{N}$ is specified in binary representation. Consider the question whether a given word equation $u = v$ has a solution that respects the length constraints. Clearly, this problem is decidable. In fact, the naive algorithm that guesses for every variable $x \in \mathcal{X}$ a word of length $\ell(x)$ over the alphabet Σ and then checks whether the guess is indeed a solution leads to a NEXPTIME-algorithm (recall that we represent the numbers for the length constraints in binary). This trivial complexity bound can drastically be improved by the following result of Plandowski and Rytter [113]:

Theorem 31 ([113]). *The following problem can be solved in polynomial time:*

Input *A system of word equations $\mathcal{S}$ over $(\Sigma, \mathcal{X})$ and a mapping $\ell : \mathcal{X} \to \mathbb{N}$, where every value $\ell(x)$ with $x \in \mathcal{X}$ is binary encoded.*

Question *Is there a solution $\sigma : \mathcal{X} \to \Sigma^*$ of $\mathcal{S}$ such that $|\sigma(x)| = \ell(x)$ for all $x \in \mathcal{X}$?*

Moreover, in case the answer is affirmative, one can compute in polynomial time for every variable $x \in \mathcal{X}$ an SLP $\mathbb{A}_x$ such that the mapping $x \mapsto \mathrm{eval}(\mathbb{A}_x)$ is a solution of $\mathcal{S}$ and $|\mathrm{eval}(\mathbb{A}_x)| = \ell(x)$ for every $x \in \mathcal{X}$.

A system of word equations $\mathcal{S}$ is *quadratic* if every variable $x \in \mathcal{X}$ occurs at most twice in $\mathcal{S}$. For quadratic systems of word equations, Diekert and Robson [117] improved the polynomial time upper bound in Theorem 31 to linear time:

Theorem 32 ([117]). *The following problem can be solved in linear time:*

Input *A quadratic system $\mathcal{S}$ of word equations over $(\Sigma, \mathcal{X})$ and a mapping $\ell : \mathcal{X} \to \mathbb{N}$, where every value $\ell(x)$ with $x \in \mathcal{X}$ is binary encoded.*

Question *Is there a solution $\sigma : \mathcal{X} \to \Sigma^*$ of $\mathcal{S}$ such that $|\sigma(x)| = \ell(x)$ for all $x \in \mathcal{X}$?*

Moreover, in case the answer is affirmative, one can compute in linear time for every variable $x \in \mathcal{X}$ an SLP $\mathbb{A}_x$ such that the mapping $x \mapsto \mathrm{eval}(\mathbb{A}_x)$ is a solution of $\mathcal{S}$ and $|\mathrm{eval}(\mathbb{A}_x)| = \ell(x)$ for every $x \in \mathcal{X}$.

The second statement of Theorem 32 is not stated explicitly in [117] but follows easily from the proof.

Let σ be a solution for a word equation $u = v$. We say that σ is *minimal*, if for every solution σ' of $u = v$ we have $|\sigma(u)| \leq |\sigma'(u)|$. We say that $|\sigma(u)|$ is the length of a minimal solution of $u = v$. By the following result of Plandowski and Rytter [113], minimal solutions for word equations are highly compressible:

Theorem 33 ([113]). *Let $u = v$ be a word equation over $(\Sigma, \mathcal{X})$ and let $n = |uv|$. Assume that $u = v$ has a solution and let N be the length of a minimal solution of $u = v$. Then, for every minimal solution σ of $u = v$, the word $\sigma(u)$ can be generated by an SLP of size $O(n^2 \log^2(N)(\log n + \log\log N))$.*

In [67], Jez applied his recompression technique from [65, 66] (see Section 6) to word equations and obtained an alternative PSPACE-algorithm for solving word equations. Moreover, his technique yields a $O(\mathsf{poly}(n) \log N)$ bound in Theorem 33 (instead of the $O(n^2 \log^2(N)(\log n + \log\log N))$ bound); this result is not explicitly state in [67].

13 Computational topology

In [36, 121, 123] efficient algorithms for SLP-compressed strings are used to solve various problems in computational topology efficiently. Fix a *connected, compact, and orientable surface* M, possibly with a boundary ∂M, see for instance [130] for more details. Typical examples are the 2-dimensional sphere and the torus (both have an empty boundary). A *simple path* in M is the image of a continuous injective mapping f from the unit interval $[0, 1] \subseteq \mathbb{R}$ into M. If moreover $f([0, 1]) \cap \partial M = \{f(0), f(1)\}$, then the simple path is called a *simple arc*. A *simple cycle* in M is the image of a continuous injective mapping f from the unit circle S_1 into M such that $f(S_1) \cap \partial M = \emptyset$. A *simple curve* in M is either a simple arc or a simple cycle. A *simple multi-curve* in M is the union of finitely many pairwise disjoint simple curves. Two simple multi-curves α and β are *isotopic*, if there exists a continuous deformation from α into β that leaves the boundary ∂M fixed. For instance, any two simple cycles on the 2-dimensional sphere are isotopic.

A *triangulation* T of M consists of a finite set V of points in M and a set E of simple paths in M (the edges of the triangulation) such that (i) each endpoint of a path $\alpha \in E$ belongs to V, (ii) if two paths from E intersect, then the intersection is a point of V, and (iii) every connected component of $M \setminus E$ is an open disc that is bounded by three paths from E (a triangle of T). A simple multi-curve α in M is normal w.r.t. T if (i) for every $\beta \in E$ the intersection $\alpha \cap \beta$ consists of finitely many points that are different from the endpoints of β, and (ii) the intersection of α with a triangle of T consists of finitely many paths and for each of these paths the endpoints are located on different sides of the triangle. The latter means that if α enters a triangle at the side β, then it has to leave the triangle at a side different from β.

For each edge e let e^{-1} be a formal inverse of e and choose an orientation $\vec{e}$ of e. A normal simple curve α can be represented by a finite word over the alphabet $E \cup E^{-1}$ as follows: Choose an orientation of α and follow the simple curve in this orientation (if α is a simple cycle then the starting point is chosen arbitrarily). Each time α intersects $\vec{e}$ from left to right (resp., from right to left) we write down e (resp., e^{-1}). The resulting word is called an *intersection sequence* of α. Clearly, in case α is closed, this word is not unique, since a cyclic rotation of an intersection sequence is again an intersection sequence. A normal simple multi-curve can be represented by the set of intersection sequences of its connected components. More compact representations of normal simple (multi-)curves are known. First of all, one can use SLPs in order to describe long intersection sequences. Another better known representation are normal coordinates. The vector of normal coordinates of the normal simple multi-curve α is the tuple $(|\alpha \cap \beta|)_{\beta \in E}$ of natural numbers. One obtains a succinct description of the curve by encoding these natural numbers in binary. Two normal simple multi-curves with the same normal coordinates must be isotopic. The number of connected components of α can grow exponentially with the total number of bits of the normal coordinates of α. Hence, there is no chance to compute from the binary encoded normal coordinates of α in polynomial time a set of SLPs for the intersection sequences of the connected components of α; simply because the number of these SLPs is too large. On the other hand, if α is connected then such an SLP can be computed by point (1) of Theorem 34 below. Moreover, the number of non-isotopic connected components of α is at most $6t$, where t is the number of triangles of the triangulation T [121, Lemma 3].

Theorem 34. *Let M be a connected, compact, and orientable surface M and let $T = (V, E)$ be a triangulation for M. The following problems can be solved in polynomial time, where the input consists of binary encoded normal coordinates for normal simple multi-curves α and β.*

(i) *Compute an SLP for an intersection sequence of α, assuming that α is connected and hence a normal simple curve* [123].

(ii) *Compute the binary encoded normal coordinates of the connected component of α that contains a given intersection point of α and T* [121].

(iii) *Compute the number of connected components of α* [121].

(iv) *Compute a list of binary encoded normal coordinates of connected components $\alpha_1, \ldots, \alpha_n$ of α, such that every connected component of α is isotopic to exactly one α_i* [121].

(v) *Check, whether α and β are isotopic (assuming that $\partial M \neq \emptyset$)* [121].

(vi) *Compute the binary encoded normal coordinates of a Dehn twist of α along β assuming that β is a simple cycle (see* [130] *for the definition of a Dehn-twist)* [123].

(vii) *Compute the geometric intersection number of α and β* [123].[7]

Note that the converse of point (1) is straightforward: One can compute the binary encoded normal coordinates of a normal simple curve α from an SLP for an intersection sequence of α by simply counting the number of occurrences of each symbol $e \in E \cup E^{-1}$ in the generated intersection sequence.

The main tool for the proof of Theorem 34 are word equations and their connection to SLPs, see Section 12. For instance, for point (1) Schaefer et al. construct a quadratic system of word equations with length constraints such that the solutions of the system are exactly the intersection sequences of α. Then, Theorem 32 can be used in order to construct in linear time an SLP for a solution of the system. This technique was first used by Schaefer et al. in [122] in order to show that string graphs can be recognized in NP. Let us also remark that the algorithm from [123] for computing Dehn twists in polynomial time (see point (6) in Theorem 34) uses the fact that the unique freely reduced normal form for a given SLP-compressed word over $\Sigma \cup \Sigma^{-1}$ can be computed in polynomial time, see the paragraph after Theorem 28.

In [123] also Dehn–Thurston coordinates, which are another succinct representation of simple multi-curves, are discussed. It is shown that Theorem 34 is also true with Dehn–Thurston coordinates instead of normal coordinates.

[7] The geometric intersection number of α and β is the minimum of $|\alpha' \cap \beta'|$, where α' (resp., β') is isotopic to α (resp., β).

14 Other applications

Let us briefly discuss some other applications of algorithms for SLPs, in particular Theorem 9 (equality testing for SLP-compressed strings is in P). For precise definitions concerning the discussed problems we refer the interested reader to the original literature.

One of the earliest applications of Theorem 9 concerns normed context-free processes [60, 78]. In [60], the first algorithm for checking bisimulation equivalence of normed context-free processes was developed; in [78] a more efficient algorithm was presented. In both papers, checking bisimulation equivalence of normed context-free processes is reduced to checking equality of SLP-compressed words. In [51], Theorem 9 is applied in the area of program analysis. It is shown that the so-called interprocedural global value numbering problem for unary uninterpreted function symbols can be solved in polynomial time.

Applications of Theorem 9 for general context-free languages can be found in [13, 135]. In [135] it shown that one can decide in polynomial time whether a context-free grammar with terminal alphabet $\Sigma \cup \overline{\Sigma}$ (where $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$ is a disjoint copy of Σ) only generates well-bracketed strings. Here we view $\overline{a}$ as the closing bracket of a, so for instance $ab\overline{b}c\overline{ca}$ is well-bracketed but $a\overline{b}$ is not. In [13], this result is generalized as follows. Assume that R is a terminating and confluent string rewriting system over an alphabet Σ such that the presented monoid Σ^*/R is cancellative. Then by [13] one can decide in polynomial time for a given context-free grammar G whether $L(G)$ is contained in the set of all strings over Σ that can be reduced with R to the empty word.

In [93], Theorem 9 is used in order to solve the isomorphism problem for regular words (a particular class of colored linear orders) in polynomial time. A regular word is given by a set of equations $\{X_i := w_i \mid 1 \leq i \leq n\}$, where every right-hand side w_i is a word over the variables X_i and terminal letters. This is an SLP, where we do not require condition (2) (acyclicity) from Definition 1. Starting from the distinguished variable X_1, we obtain an infinite derivation tree, where every leaf is labelled with a terminal symbol. The generated regular word is obtained by taking the natural left-to-right order on the leaves of the derivation tree, together with the labels on the leaves. For instance, the recursive equation $X_1 := abX_1$ defines the regular word, where the underlying linear order is $(\mathbb{N}, \leq)$, and every even (resp., odd) position is labelled with a (resp., b). Another example is $X_1 := X_1aX_1bX_1$. It is not hard to see that this equation defines the regular word consisting of the rational numbers with the natural order, which are densely colored with a and b (between every two different rationals, there is an a-colored point as well as a b-colored point). In [93], it is shown that one can decide for two given sets of equations in polynomial time, whether they define isomorphic

regular words. The proof is based on a reduction to equality for SLP-compressed strings.

15 Extensions to trees and pictures

The idea of grammar-based compression can be extended from strings to more complicated structures. In fact, all one needs is a set of operations on a domain D of objects, that construct "larger" D-objects from smaller ones. Using straight-line programs over D with these operations allows to construct "very large" D-objects.

15.1 Trees

For background on trees, tree grammars, and tree automata see [27]. The concept of an SLP can easily be generalized from strings to trees. Here, trees are rooted node-labelled trees. Every node has a label from an alphabet Σ. Moreover, with every symbol $a \in \Sigma$ a natural number (the rank of a) is associated. If a tree node v is labelled with a symbol of rank n, then v has exactly n children, which are linearly ordered. Such trees can conveniently be represented as terms. The size $|t|$ of a tree t is the number of nodes of t. Here is an example:

Example 35. Let f be a symbol of rank 2, h a symbol of rank 1, and a a symbol of rank 0 (a constant). Then the term $h(f(h(f(h(h(a)),a)),h(f(h(h(a)),a))))$ corresponds to the node-labelled tree of size 14, shown in Figure 4.

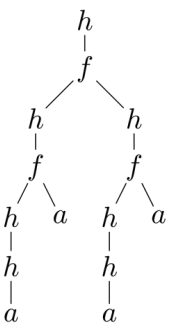

Figure 4. A node-labelled tree.

A tree straight-line program (TSLP for short and also called SLCF tree grammar in [89, 91]) over the terminal alphabet Σ (which is a ranked alphabet in the above sense) is a tuple $\mathbb{A} = (N, \Sigma, S, P)$, such that N is a finite set of

ranked symbols (the nonterminals), $S \in N$ has rank 0 (the initial nonterminal) and P is a finite set of productions of the form $A(y_1, \ldots, y_n) \to t$ where A is a nonterminal of rank n and t is a tree built up from the ranked symbols in $\Sigma \cup N$ and the parameters $y_1, \ldots, y_n$ which are considered as symbols of rank 0 (i.e., constants). Moreover, as for ordinary SLPs, it is required that every nonterminal occurs on the left-hand side of exactly one production, and the relation $\{(A, B) \in N \times N \mid (A(y_1, \ldots, y_n) \to t) \in P, B \text{ occurs in } t\}$ is acyclic. A TSLP $\mathbb{A}$ generates a tree $\text{eval}(\mathbb{A})$ in the natural way. During the derivation process, the parameters $y_1, \ldots, y_n$ are instantiated with concrete trees. Instead of giving a formal definition, let us consider and example.

Example 36. Let S, A, B, C be nonterminals, let S be the start nonterminal and let the TSLP $\mathbb{A}$ consist of the following productions:

$$\begin{aligned} S &\to A(B(a), B(a)), \\ A(x_1, x_2) &\to C(C(x_1, a), C(x_2, a)), \\ C(x_1, x_2) &\to h(f(x_1, x_2)), \\ B(x) &\to h(h(x)). \end{aligned}$$

Then $\text{eval}(G)$ is the tree from Example 35. It can be derived as follows:

$$\begin{aligned} S &\to A(B(a), B(a)) \\ &\to C(C(B(a), a), C(B(a), a)) \\ &\to C(C(h(h(a)), a), C(h(h(a)), a)) \\ &\to h(f(C(h(h(a)), a), C(h(h(a)), a))) \\ &\to h(f(h(f(h(h(a)), a)), h(f(h(h(a)), a)))). \end{aligned}$$

The size of a TSLP $\mathbb{A} = (N, \Sigma, S, P)$ is defined as the sum of the sizes of all right-hand sides of P. Due to multiple occurrences of parameters in right-hand sides, TSLPs are able to generate trees of doubly exponential size.

Example 37. Let the TSLP $\mathbb{A}_n$ consist of the following productions:

$$\begin{aligned} S &\to A_0(a), \\ A_i(y_1) &\to A_{i+1}(A_{i+1}(y_1)) \quad \text{for } 0 \leq i < n, \\ A_n(y_1) &\to f(y_1, y_1). \end{aligned}$$

Then $\text{eval}(\mathbb{A}_n)$ is a complete binary tree of height 2^n. Thus,

$$|\text{eval}(\mathbb{A}_n)| = 2 \cdot 2^{2^n} - 1.$$

Example 37 motivates the definition of linear TSLPs. A TSLP $\mathbb{A} = (N, S, P)$ is linear if for every rule $(A(y_1, \ldots, y_n) \to t) \in P$, each of the parameters $y_1, \ldots, y_n$ occurs exactly once in the tree t. The TSLP from Example 36 is linear. In contrast to non-linear TSLPs, for a linear TSLP $\mathbb{A}$ the size of tree $\mathrm{eval}(\mathbb{A})$ is bounded by $2^{O(|\mathbb{A}|)}$. Efficient algorithms that generate for a given input tree t a linear TSLP $\mathbb{A}$ with $\mathrm{eval}(\mathbb{A}) = t$ are described in [18,90] and [2] (for a slightly different type of grammars). A TSLP is nothing else than a context-free tree grammar that generates a single tree. An extension of TSLPs to higher order tree grammars was recently proposed in [75].

Let us now consider algorithmic problems for TSLP-compressed trees. Concerning equality checking, we have the following:

Theorem 38 ([18, 125]). *For two given TSLPs* $\mathbb{A}$ *and* $\mathbb{B}$ *it can be checked in* PSPACE*, whether* $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$. *For two given linear TSLPs* $\mathbb{A}$ *and* $\mathbb{B}$ *it can be checked in* P*, whether* $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$.

For the PSPACE-bound in the first statement, one has to notice that a TSLP $\mathbb{A}$ can also be viewed as a graph generating grammar that produces a node-labelled directed acyclic graph $\mathsf{dag}(\mathbb{A})$, whose unfolding is $\mathrm{eval}(\mathbb{A})$. This graph grammar is obtained by merging in a right-hand side $t(y_1, \ldots, y_n)$ of a production $A(y_1, \ldots, y_n) \to t$ all occurrences of a variable y_i into a single y_i-labelled node. The number of nodes of $\mathsf{dag}(\mathbb{A})$ is bounded singly exponential in the size of $\mathbb{A}$, hence a node of $\mathsf{dag}(\mathbb{A})$ can be represented in polynomial space. A path in $\mathsf{dag}(\mathbb{A})$ that starts in the root can be represented by the sequence of natural numbers $n_1, \ldots, n_k$, such that in the ith step the path moves from the current node to its n_ith child node. A PSPACE-algorithm for checking $\mathrm{eval}(\mathbb{A}) \neq \mathrm{eval}(\mathbb{B})$ (this suffices, since PSPACE is closed under complement) simply guesses such a path p step by step and thereby stores the nodes of $\mathsf{dag}(\mathbb{A})$ and $\mathsf{dag}(\mathbb{B})$, respectively, that are reached by the path p. The algorithm rejects as soon as these two nodes are labelled by different symbols. It remains open, whether equality of TSLP-compressed trees can be checked more efficiently.

For the second statement of Theorem 38 one constructs in polynomial time from a linear TSLP $\mathbb{A}$ a (string generating) SLP $\mathbb{A}'$ such that $\mathrm{eval}(\mathbb{A}')$ represents a depth-first left-to-right transversal of the tree $\mathrm{eval}(\mathbb{A})$. For two TSLPs $\mathbb{A}$ and $\mathbb{B}$ we have $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$ if and only if $\mathrm{eval}(\mathbb{A}') = \mathrm{eval}(\mathbb{B}')$. Hence, equality of trees that are represented by linear TSLPs can be reduced to checking equality of SLP-compressed strings, which can be checked in polynomial time by Theorem 9. In [125], the second statement of Theorem 38 is shown by a direct extension of Plandowski's algorithm for string SLPs.

In [89, 91], the problem of evaluating tree automata over TSLP-compressed input trees is considered. A tree automaton runs on an node-labelled input tree bottom-up and thereby assigns states to tree nodes. Transitions are of the form $(q_1, \ldots, q_n, f, q)$, where f is a symbol (node label) of rank n and $q_1, \ldots, q_n, q$ are states of the tree automaton. Then a *run* of the tree automaton is a mapping ρ from the tree nodes to states that is consistent with the set of transitions in the following sense: If a tree node is labelled with the symbol f (of rank n) and $v_1, \ldots, v_n$ are the children of v in that order, then $(\rho(v_1), \ldots, \rho(v_n), f, \rho(v))$ must be a transition of the tree automaton. A tree automaton accepts a tree if there is a run that assigns a final state to the root of the tree (every tree automaton has a distinguished set of final states).

Theorem 39. *The following complexity results hold:*

- *It is* PSPACE*-complete to check for a given TSLP* $\mathbb{A}$ *and a given tree automaton A, whether A accepts* eval($\mathbb{A}$) [89].
- *It is* P*-complete to check for a given linear TSLP* $\mathbb{A}$ *and a given tree automaton A, whether A accepts* eval($\mathbb{A}$) [91].

The polynomial time algorithm in the linear case works in two steps:

(1) Transform in polynomial time the input TSLP $\mathbb{A}$ into a TSLP $\mathbb{B}$ such that eval($\mathbb{A}$) = eval($\mathbb{B}$) and every nonterminal of $\mathbb{B}$ has rank at most one. This is the difficult step in [91].

(2) Evaluate the input tree automaton A on eval($\mathbb{B}$) in polynomial time, using the fact that every nonterminal of $\mathbb{B}$ has rank at most one, see [89].

Several other algorithmic problems for TSLP-compressed input trees are studied in [20, 40, 79, 126, 127].

15.2 Pictures

A picture over the alphabet Σ is a mapping $p : \{1, \ldots, n\} \times \{1, \ldots, m\} \to \Sigma$ for some $n, m \geq 1$. We say that (n, m) is the dimension of the picture p. For more information on pictures see [46]. A picture can be viewed as a 2-dimensional word. We define two partial concatenation operations for pictures. Given two pictures p_1 and p_2, where p_1 has dimension (n, m) and p_2 has dimension (n, k), we define the horizontal concatenation $(p_1, p_2) : \{1, \ldots, n\} \times \{1, \ldots, m + k\} \to \Sigma$ (a picture of dimension $(n, m + k)$) as follows, where $x \in \{1, \ldots, n\}$ and $y \in \{1, \ldots, m + k\}$:

$$(p_1, p_2)(x, y) = \begin{cases} p_1(x, y) & \text{if } y \leq m, \\ p_2(x, y - m) & \text{if } y > m. \end{cases}$$

Given two pictures p_1 and p_2, where p_1 has dimension (m,n) and p_2 has dimension (k,n), we define the vertical concatenation $\binom{p_2}{p_1} : \{1,\ldots,m+k\} \times \{1,\ldots,n\} \to \Sigma$ (a picture of dimension $(m+k,n)$) as follows, where $x \in \{1,\ldots,m+k\}$ and $y \in \{1,\ldots,n\}$:

$$\binom{p_2}{p_1}(x,y) = \begin{cases} p_1(x,y) & \text{if } x \le m, \\ p_2(x-m,y) & \text{if } x > m. \end{cases}$$

It is now straightforward to define 2-dimensional straight-line programs (briefly 2SLPs). The productions of a 2SLP $\mathbb{A}$ have one of the forms

$$X \to a, \qquad X \to (Y,Z), \qquad X \to \binom{Z}{Y},$$

where X, Y, Z are nonterminals and a is a terminal symbol. In order to generate a picture, we have to require that the dimensions match. Define the dimension of a nonterminal X inductively as follows. If the unique production for X has the form $X \to a$, then the dimension of X is $(1,1)$. If the unique production for X has the form $X \to (Y,Z)$, then, inductively the dimension (n,m) (resp., (k,ℓ)) of Y (resp., Z) is already defined. We require that $n = k$ and define the dimension of X as $(n, m+\ell)$. The constraint for a production $X \to \binom{Z}{Y}$ is analogous. If the dimension of the initial nonterminal S is (n,m), then the 2SLP $\mathbb{A}$ defines a picture $\mathrm{eval}(\mathbb{A})$ of dimension (n,m).

Algorithmic problems for 2SLPs were studied in [11]. For equality checking, the following was shown; recall the definition of the complexity class coRP from Section 11.

Theorem 40 ([11]). *The problem of checking* $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$ *for two given 2SLPs belongs to the complexity class* coRP.

For the proof, one encodes a picture p of dimension (m,n) over the alphabet $\{1,\ldots,k\}$ as the polynomial

$$\mathsf{poly}_p(x,y) = \sum_{1 \le i \le m} \sum_{1 \le j \le n} p(i,j) x^{i-1} y^{j-1}.$$

For two 2SLPs $\mathbb{A}$ and $\mathbb{B}$, we have $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$ if and only if

$$\mathsf{poly}_{\mathrm{eval}(\mathbb{A})}(x,y) - \mathsf{poly}_{\mathrm{eval}(\mathbb{B})}(x,y)$$

is the zero polynomial. Now, from a given 2SLP $\mathbb{A}$ one can construct in polynomial time a circuit over the polynomial ring $\mathbb{Z}[x,y]$ that defines the polynomial

$\mathsf{poly}_{\mathrm{eval}(\mathbb{A})}(x, y)$. Hence, the initial question whether $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$ can be reduced to the problem whether a circuit over the polynomial ring $\mathbb{Z}[x, y]$ evaluates to the zero polynomial. This is an instance of the algebraic identity testing that we already encountered after Theorem 30 and that belongs to coRP. It remains open, whether equality of 2SLP-compressed pictures can be checked in polynomial time.

Also pattern matching problems can naturally be defined for pictures. A picture p of dimension (k, ℓ) occurs in the picture q of dimension (m, n) if there exist $1 \le i \le m$ and $1 \le j \le n$ such that (i) $i + k - 1 \le m$, (ii) $j + \ell - 1 \le n$, and (iii) for all $0 \le x \le k-1$ and $0 \le y \le \ell-1$ we have $q(i+x, j+y) = p(x+1, y+1)$. The following two results from [11] pinpoint the complexity of 2-dimensional pattern matching:

Theorem 41 ([11]). *It is* coNP*-complete to check for an explicitly given picture p and a 2SLP $\mathbb{A}$, whether p occurs in* $\mathrm{eval}(\mathbb{A})$.

The class Σ_2^p consists of all complements of Π_2^p-sets, see the end of Section 11 for the definition of Π_2^p. Hence, $L \subseteq \Sigma^*$ belongs to Σ_2^p if there exists a polynomial time set $P \subseteq \Sigma^* \times \{0,1\}^* \times \{0,1\}^*$ and a polynomial $p(x)$ such that for all $u \in \Sigma^*$ we have:

$$u \in L \iff \exists v \in \{0,1\}^{\le p(|u|)}\ \forall w \in \{0,1\}^{\le p(|u|)} : (u, v, w) \in P. \qquad (2)$$

Theorem 42 ([11]). *It is Π_2^p-complete to check for two given 2SLP $\mathbb{A}$ and $\mathbb{B}$, whether* $\mathrm{eval}(\mathbb{A})$ *occurs in* $\mathrm{eval}(\mathbb{B})$.

The upper complexity bounds in Theorems 41 and 42 are straightforward. For instance, for two given 2SLPs $\mathbb{A}$ and $\mathbb{B}$ the definition of "$\mathrm{eval}(\mathbb{A})$ occurs in $\mathrm{eval}(\mathbb{B})$" directly leads to a formula of the general form (2). The lower complexity bound in Theorem 41 is shown by a reduction from 3SAT. Assume that $F(x_1, \ldots, x_n) = \bigwedge_{i=1}^{n} C_i$ is a boolean formula, where every C_i is a conjunction of three literals (a literal is a boolean variable or a negated boolean variable) and $x_1, \ldots, x_m$ are the variables that occur in F. Hence there are 2^m truth assignments. For a given clause $C_i = (A \vee B \vee C)$, where A, B, and C are literals, one can construct an SLP $\mathbb{A}_i$ over the alphabet $\{0,1\}$ such that $|\mathrm{eval}(\mathbb{A}_i)| = 2^m$ and for all $1 \le j \le 2^m$, $\mathrm{eval}(\mathbb{A}_i)[j]$ is the truth value of C_i under the jth truth assignment in lexicographic order. Now, one can easily construct a 2SLP $\mathbb{A}$ such that $\mathrm{eval}(\mathbb{A})$ is a picture of dimension $(n, 2^m)$ such that the ith row of $\mathrm{eval}(\mathbb{A})$ is the word $\mathrm{eval}(\mathbb{A}_i)$ for every $1 \le i \le n$. Hence, the formula F is satisfiable if and only if the picture $\mathrm{eval}(\mathbb{A})$ contains a column only consisting of 1's. The lower bound in Theorem 42 is shown by a reduction from a Π_2^p-complete variant of the subsetsum problem.

16 Open problems

Let us state some open problems that in the opinion of the author deserve further investigations.

- Is the problem of checking $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$ for two given SLPs $\mathbb{A}$ and $\mathbb{B}$ P-complete, or does this problem belong to the parallel complexity class NC (see Section 5)? The latter would mean that the problem can be solved in polylogarithmic time using polynomially many processors. The same question can be considered also for the fully compressed pattern matching problem (see Section 6).
- What is the precise complexity of the fully compressed subsequence matching problem (see Section 8)? The author conjectures that this problem is PSPACE-complete.
- Is the compressed word problem (see Section 11) for a finitely generated linear group solvable in polynomial time? What about subclasses of linear groups, e.g., braid groups and polycyclic groups.
- The *compressed subgroup membership problem* for a free group $F(\Sigma)$ is the following problem: Given SLPs $\mathbb{A}, \mathbb{B}_1, \ldots, \mathbb{B}_n$ (over $\Sigma \cup \Sigma^{-1}$), does the word $\mathrm{eval}(\mathbb{A})$ represent a group element from the subgroup of $F(\Sigma)$ generated by $\mathrm{eval}(\mathbb{B}_1), \ldots, \mathrm{eval}(\mathbb{B}_n)$? We are only aware of an exponential time algorithm for this problem.
- Can equality of 2SLP-compressed pictures (see Section 15.2) be checked in polynomial time?

Acknowledgments. The author is very grateful to Pawel Gawrychowski, Artur Jez, Sebastian Maneth, Wojciech Rytter, Marcus Schäfer, and Manfred Schmidt-Schauß for many valuable comments and reading a preliminary version of this paper.

Bibliography

[1] M. Agrawal and R. Saptharishi, Classifying polynomials and identity testing, *Current Trends in Science – Platinum Jubilee Special* (2009), 149–162.

[2] T. Akutsu, A bisection algorithm for grammar-based compression of ordered trees, *Inf. Process. Lett.* **110** (2010), no. 18-19, 815–820.

[3] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen and P. B. Miltersen, On the complexity of numerical analysis, *SIAM J. Comput.* **38** (2009), no. 5, 1987–2006.

[4] S. Alstrup, G. S. Brodal and T. Rauhe, Dynamic pattern matching, Technical Report DIKU Report 98/27, Department of Computer Science, University of Copenhagen, 1998.

[5] S. Alstrup, G. S. Brodal and T. Rauhe, Pattern matching in dynamic texts, in: *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2000), ACM/SIAM (2000), 819–828.

[6] R. Alur and P. Madhusudan, Visibly pushdown languages, in: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing* (STOC 2004), ACM Press (2004), 202–211.

[7] A. Amir, G. Benson and M. Farach, Let sleeping files lie: Pattern matching in Z-compressed files, *J. Comput. Syst. Sci. Int.* **52** (1996), no. 2, 299–307.

[8] J. Arpe and R. Reischuk, On the complexity of optimal grammar-based compression, in: *Proceedings of the Data Compression Conference* (DCC 2006), IEEE Computer Society (2006), 173–182.

[9] D. A. M. Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, *J. Comput. Syst. Sci. Int.* **38** (1989), 150–164.

[10] M. Beaudry, P. McKenzie, P. Péladeau and D. Thérien, Finite monoids: From word to circuit evaluation, *SIAM J. Comput.* **26** (1997), no. 1, 138–152.

[11] P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski and W. Rytter, On the complexity of pattern matching for highly compressed two-dimensional texts, *J. Comput. Syst. Sci. Int.* **65** (2002), no. 2, 332–350.

[12] A. Bertoni, C. Choffrut and R. Radicioni, Literal shuffle of compressed words, in: *Proceeding of the 5th IFIP International Conference on Theoretical Computer Science* (IFIP TCS 2008), Springer (2008), 87–100.

[13] A. Bertoni, C. Choffrut and R. Radicioni, The inclusion problem of context-free languages: Some tractable cases, *Int. J. Found. Comput. Sci.* **22** (2011), no. 2, 289–299.

[14] M. Bestvina and N. Brady, Morse theory and finiteness properties of groups, *Invent. Math.* **129** (1997), no. 3, 445–470.

[15] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti and O. Weimann, Random access to grammar-compressed strings, in: *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2011), SIAM (2011), 373–389.

[16] W. W. Boone, The word problem, *Ann. of Math. (2)* **70** (1959), 207–265.

[17] I. Burmistrov and L. Khvorost, Straight-line programs: A practical test, in: *Proceedings of the 1st International Conference on Data Compression, Communications and Processing* (CCP 2011), IEEE (2011), 76–81.

[18] G. Busatto, M. Lohrey and S. Maneth, Efficient memory representation of XML document trees, *Information Systems* **33** (2008), no. 4–5, 456–474.

[19] F. B. Cannonito and R. W. Gatterdam, The word problem and power problem in 1-relator groups are primitive recursive, *Pacific J. Math.* **61** (1975), no. 2, 351–359.

[20] A. G. Carles Creus and G. Godoy, One-context unification with STG-compressed terms is in NP, in: *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications* (RTA 2012), Leibniz International Proceedings in Informatics (2012), 149–164.

[21] H. Caussinus, P. McKenzie, D. Thérien and H. Vollmer, Nondeterministic NC^1 computation, *J. Comput. Syst. Sci. Int.* **57** (1998), no. 2, 200–212.

[22] P. Cégielski, I. Guessarian, Y. Lifshits and Y. Matiyasevich, Window subsequence problems for compressed texts, in: *Proceedings of the International Computer Science Symposium in Russia* (CSR 2006), Lecture Notes in Comput. Sci. 3967, Springer (2006), 127–136.

[23] M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai and A. Shelat, The smallest grammar problem, *IEEE Trans. Inf. Theory* **51** (2005), no. 7, 2554–2576.

[24] S. Chen, E. Verbin and W. Yu, Data structure lower bounds on random access to grammar-compressed strings, technical report (2012), `http://arxiv.org/abs/1203.1080`.

[25] F. Claude and G. Navarro, Improved grammar-based compressed indexes, technical report (2011), `http://arxiv.org/abs/1110.4493`.

[26] F. Claude and G. Navarro, Self-indexed grammar-based compression, *Fundam. Inform.* **111** (2011), no. 3, 313–337.

[27] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison and M. Tommasi, Tree automata techniques and applications, preprint (2007), `http://tata.gforge.inria.fr`.

[28] K. J. Compton and C. W. Henson, A uniform method for proving lower bounds on the computational complexity of logical theories, *Ann. Pure Appl. Logic* **48** (1990), 1–79.

[29] J. Crisp and B. Wiest, Embeddings of graph braid and surface groups in right-angled artin groups and braid groups, *Algebr. Geom. Topol.* **4** (2004), 439–472.

[30] M. Crochemore, G. M. Landau and M. Ziv-Ukelson, A subquadratic sequence alignment algorithm for unrestricted scoring matrices, *SIAM J. Comput.* **32** (2003), no. 6, 1654–1673.

[31] M. Dehn, Über die Toplogie des dreidimensionalen Raumes, *Math. Ann.* **69** (1910), 137–168.

[32] V. Diekert, Makanin's algorithm, in: *Algebraic Combinatorics on Words*, Cambridge University Press (2001), 342–390.

[33] V. Diekert, J. Laun and A. Ushakov, Efficient algorithms for highly compressed data: The word problem in Higman's group is in P, in: *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science* (STACS 2012), Leibniz International Proceedings in Informatics 14 (2012), 218–229.

[34] P. W. Dymond, Input-driven languages are in $\log n$ depth, *Inform. Process. Lett.* **26** (1988), no. 5, 247–250.

[35] D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson and W. P. Thurston, *Word Processing in Groups*, Jones and Bartlett, Boston, 1992.

[36] J. Erickson and A. Nayyeri, Tracing compressed curves in triangulated surfaces, in: *Proceedings of the 28th Symposium on Computational Geometry* (SOCG 2012), ACM (2012), 131–140.

[37] M. Farach and M. Thorup, String matching in Lempel-Ziv compressed strings, *Algorithmica* **20** (1998), no. 4, 388–404.

[38] N. J. Fine and H. S. Wilf, Uniqueness theorems for periodic functions, *Proc. Amer. Math. Soc.* **16** (1965), 109–114.

[39] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich and S. J. Puglisi, A faster grammar-based self-index, in: *Proceedings of the 6th International Conference on Language and Automata Theory and Applications* (LATA 2012), Lecture Notes in Comput. Sci. 7183, Springer (2012), 240–251.

[40] A. Gascón, G. Godoy and M. Schmidt-Schauß, Unification and matching on compressed terms, *ACM Trans. Comput. Log.* **12** (2011), no. 4, article number 26.

[41] L. Gasieniec, M. Karpinski, W. Plandowski and W. Rytter, Efficient algorithms for Lempel-Ziv encoding (extended abstract), in: *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory* (SWAT 1996), Lecture Notes in Comput. Sci. 1097, Springer (1996), 392–403.

[42] L. Gasieniec, M. Karpinski, W. Plandowski and W. Rytter, Randomized efficient algorithms for compressed strings: The finger-print approach (extended abstract), in: *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching* (CPM 1996), Lecture Notes in Comput. Sci. 1075, Springer (1996), 39–49.

[43] P. Gawrychowski, Optimal pattern matching in LZW compressed strings, in: *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2011), SIAM (2011), 362–372.

[44] P. Gawrychowski, Pattern matching in Lempel-Ziv compressed strings: Fast, simple, and deterministic, in: *Proceedings of the 19th Annual European Symposium on Algorithms* (ESA 2011), Lecture Notes in Comput. Sci. 6942, Springer (2011), 421–432.

[45] R. Ghrist and V. Peterson, The geometry and topology of reconfiguration, *Adv. in Appl. Math.* **38** (2007), no. 3, 302–323.

[46] D. Giammarresi and A. Restivo, Two-dimensional languages, in: *Handbook of Formal Languages*, volume 3, Springer (1997), 216–267.

[47] L. M. Goldschlager, ε-productions in context-free grammars, *Acta Inform.* **16** (1981), 303–308.

[48] K. Goto, H. Bannai, S. Inenaga and M. Takeda, Fast q-gram mining on SLP compressed strings, in: *Proceedings of the 18th International Symposium on String Processing and Information Retrieval* (SPIRE 2011), Lecture Notes in Comput. Sci. 7024, Springer (2011), 278–289.

[49] K. Goto, H. Bannai, S. Inenaga and M. Takeda, Speeding-up q-gram mining on grammar-based compressed texts, technical report (2012), `http://arxiv.org/abs/1202.3311`.

[50] S. Greibach, The hardest context-free language, *SIAM J. Comput.* **2** (1973), no. 4, 304–310.

[51] S. Gulwani and A. Tiwari, Computing procedure summaries for interprocedural analysis, in: *Proceedings of the 16th European Symposium on Programming* (ESOP 2007), Lecture Notes in Comput. Sci. 4421, Springer (2007), 253–267.

[52] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1999.

[53] C. Gutiérrez, Satisfiability of word equations with constants is in exponential space, in: *Proceedings of the 39th Annual Symposium on Foundations of Computer Science* (FOCS 1998), IEEE Computer Society Press (1998), 112–119.

[54] C. Hagenah, *Gleichungen mit regulären Randbedingungen über freien Gruppen*, Ph.D. thesis, University of Stuttgart, Institut für Informatik, 2000.

[55] F. Haglund and D. T. Wise, Coxeter groups are virtually special, *Adv. Math.* **224** (2010), no. 5, 1890–1903.

[56] N. Haubold and M. Lohrey, Compressed word problems in HNN-extensions and amalgamated products, *Theory Comput. Syst.* **49** (2011), no. 2, 283–305.

[57] N. Haubold, M. Lohrey and C. Mathissen, Compressed conjugacy and the word problem for outer automorphism groups of graph groups, in: *Proceedings of Developments in Language Theory* (DLT 2010), Lecture Notes in Comput. Sci. 6224, Springer (2010), 218–230. Long version in *Internat. J. Algebra Comput.*, to appear.

[58] D. Hermelin, G. M. Landau, S. Landau and O. Weimann, Unified compression-based acceleration of edit-distance computation, technical report (2010), `http://arxiv.org/abs/1004.1194`.

[59] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer and K. W. Wagner, On the power of polynomial time bit-reductions, in: *Proceedings of the 8th Annual Structure in Complexity Theory Conference*, IEEE Computer Society Press (1993), 200–207.

[60] Y. Hirshfeld, M. Jerrum and F. Moller, A polynomial algorithm for deciding bisimilarity of normed context-free processes, *Theoret. Comput. Sci.* **158** (1996), no. 1–2, 143–159.

[61] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison–Wesley, Reading, 1979.

[62] H. B. Hunt III, D. J. Rosenkrantz and T. G. Szymanski, On the equivalence, containment, and covering problems for the regular and context-free languages, *J. Comput. Syst. Sci.* **12** (1976), no. 2, 222–268.

[63] O. H. Ibarra and S. Moran, Probabilistic algorithms for deciding equivalence of straight-line programs, *J. ACM* **30** (1983), no. 1, 217–228.

[64] B. Jenner, P. McKenzie and D. Thérien, Logspace and logtime leaf languages, *Inform. and Comput.* **129** (1996), no. 1, 21–33.

[65] A. Jez, Compressed membership for NFA (DFA) with compressed labels is in NP (P), in: *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science* (STACS 2012), Leibniz International Proceedings in Informatics 14 (2012), 136–147.

[66] A. Jez, Faster fully compressed pattern matching by recompression, in: *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming* (ICALP 2012), Lecture Notes in Comput. Sci. 7391, Springer (2012), 533–544.

[67] A. Jez, Recompression: A simple and powerful technique for word equations, technical report (2012), `http://arxiv.org/abs/1203.3705`.

[68] T. Jiang and B. Ravikumar, A note on the space complexity of some decision problems for finite automata, *Inform. Process. Lett.* **40** (1991), no. 1, 25–31.

[69] E. Kaltofen, Greatest common divisors of polynomials given by straight-line programs, *J. ACM* **35** (1988), no. 1, 231–264.

[70] I. Kapovich, A. Miasnikov, P. Schupp and V. Shpilrain, Generic-case complexity, decision problems in group theory, and random walks, *J. Algebra* **264** (2003), no. 2, 665–694.

[71] H. J. Karloff and W. L. Ruzzo, The iterated mod problem, *Inform. and Comput.* **80** (1989), no. 3, 193–204.

[72] M. Karpinski, W. Rytter and A. Shinohara, Pattern-matching for strings with short descriptions, in: *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching* (CPM 1995), Lecture Notes in Comput. Sci. 937, Springer (1995), 205–214.

[73] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara and S. Arikawa, Collage system: A unifying framework for compressed pattern matching, *Theoret. Comput. Sci.* **1** (2003), no. 298, 253–272.

[74] D. E. Knuth, J. H. Morris, Jr. and V. R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.* **6** (1977), no. 2, 323–350.

[75] N. Kobayashi, K. Matsuda and A. Shinohara, Functional programs as compressed data, in: *Proceedings of the ACM SIGPLAN 2012 Workshop on Partial Evaluation and Program Manipulation* (PEPM 2012), ACM (2012), 121–130.

[76] D. Kozen, Lower bounds for natural proof systems, in: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (FOCS 1977), IEEE Computer Society Press (1977), 254–266.

[77] N. J. Larsson and A. Moffat, Offline dictionary-based compression, in: *Proceedings of the Data Compression Conference* (DCC 1999), IEEE Computer Society Press (1999), 296–305.

[78] S. Lasota and W. Rytter, Faster algorithm for bisimulation equivalence of normed context-free processes, in: *Proceedings of the 31th International Symposium on Mathematical Foundations of Computer Science* (MFCS 2006), Lecture Notes in Comput. Sci. 4162, Springer (2006), 646–657.

[79] J. Levy, M. Schmidt-Schauß and M. Villaret, The complexity of monadic second-order unification, *SIAM J. Comput.* **38** (2008), no. 3, 1113–1140.

[80] Y. Lifshits, Processing compressed texts: A tractability border, in: B. Ma and K. Zhang, editors, *Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching* (CPM 2007), Lecture Notes in Comput. Sci. 4580, Springer (2007), 228–240.

[81] Y. Lifshits and M. Lohrey, Querying and embedding compressed texts, in: *Proceedings of the 31th International Symposium on Mathematical Foundations of Computer Science* (MFCS 2006), Lecture Notes in Comput. Sci. 4162, Springer (2006), 681–692.

[82] Y. Lifshits, S. Mozes, O. Weimann and M. Ziv-Ukelson, Speeding up HMM decoding and training by exploiting sequence repetitions, *Algorithmica* **54** (2009), no. 3, 379–399.

[83] R. J. Lipton and Y. Zalcstein, Word problems solvable in logspace, *J. ACM* **24** (1977), no. 3, 522–526.

[84] M. Lohrey, Word problems and membership problems on compressed words, *SIAM J. Comput.* **35** (2006), no. 5, 1210–1240.

[85] M. Lohrey, Compressed membership problems for regular expressions and hierarchical automata, *Internat. J. Found. Comput. Sci.* **21** (2010), no. 5, 817–841.

[86] M. Lohrey, Compressed word problems for inverse monoids, in: *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science* (MFCS 2011), Lecture Notes in Comput. Sci. 6907, Springer (2011), 448–459.

[87] M. Lohrey, Leaf languages and string compression, *Inform. and Comput.* **209** (2011), no. 6, 951–965.

[88] M. Lohrey, Model-checking hierarchical structures, *J. Comput. Syst. Sci. Int.* **78** (2012), no. 2, 461–490.

[89] M. Lohrey and S. Maneth, The complexity of tree automata and XPath on grammar-compressed trees, *Theoret. Comput. Sci.* **363** (2006), no. 2, 196–210.

[90] M. Lohrey, S. Maneth and R. Mennicke, Tree structure compression with repair, in: *Proceedings of the Data Compression Conference* (DCC 2011), IEEE Computer Society (2011), 353–362.

[91] M. Lohrey, S. Maneth and M. Schmidt-Schauß, Parameter reduction and automata evaluation for grammar-compressed trees, *J. Comput. System Sci.* **78** (2012), no. 5, 1651–1669.

[92] M. Lohrey and C. Mathissen, Compressed membership in automata with compressed labels, in: *Proceedings of the 6th International Computer Science Symposium in Russia* (CSR 2011), Lecture Notes in Comput. Sci. 6651, Springer (2011), 275–288.

[93] M. Lohrey and C. Mathissen, Isomorphism of regular trees and words, in: *Proceeding of the 38th International Colloquium on Automata, Languages and Programming* (ICALP 2011), Lecture Notes in Comput. Sci. 6756, Springer (2011), 210–221.

[94] M. Lohrey and S. Schleimer, Efficient computation in groups via compression, in: *Proceedings of Computer Science in Russia* (CSR 2007), Lecture Notes in Comput. Sci. 4649, Springer (2007), 249–258.

[95] R. C. Lyndon and P. E. Schupp, *Combinatorial Group Theory*, Springer, 1977.

[96] J. Macdonald, Compressed words and automorphisms in fully residually free groups, *Internat. J. Algebra Comput.* **20** (2010), no. 3, 343–355.

[97] G. S. Makanin, The problem of solvability of equations in a free semigroup (in Russian), *Math. Sbornik* **103** (1977), 147–236; translation in *Math. USSR Sbornik* **32** (1977), 129–198.

[98] N. Markey and P. Schnoebelen, Model checking a path, in: *Proceedings of the 14th International Conference on Concurrency Theory* (CONCUR 2003), Lecture Notes in Comput. Sci. 2761, Springer (2003), 248–262.

[99] N. Markey and P. Schnoebelen, A PTIME-complete matching problem for SLP-compressed words, *Inform. Process. Lett.* **90** (2004), no. 1, 3–6.

[100] S. Maruyama, M. Takeda, M. Nakahara and H. Sakamoto, An online algorithm for lightweight grammar-based compression, in: *Proceedings of the 1st International Conference on Data Compression, Communications and Processing* (CCP 2011), IEEE (2011), 19–28.

[101] S. Maruyama, Y. Tanaka, H. Sakamoto and M. Takeda, Context-sensitive grammar transform: Compression and pattern matching, in: *Proceedings of the 15th International Symposium on String Processing and Information Retrieval* (SPIRE 2008), Lecture Notes in Comput. Sci. 5280, Springer (2008), 27–38.

[102] W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura and K. Hashimoto, Efficient algorithms to compute compressed longest common substrings and compressed palindromes, *Theoret. Comput. Sci.* **410** (2009), no. 8–10, 900–913.

[103] A. J. Mayer and L. J. Stockmeyer, The complexity of word problems – This time with interleaving, *Inform. and Comput.* **115** (1994), no. 2, 293–311.

[104] K. Mehlhorn, R. Sundar and C. Uhrig, Maintaining dynamic sequences under equality tests in polylogarithmic time, *Algorithmica* **17** (1997), no. 2, 183–198.

[105] M. Miyazaki, A. Shinohara and M. Takeda, An improved pattern matching algorithm for strings in terms of straight-line programs, in: *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching* (CPM 1997), Lecture Notes in Comput. Sci. 1264, Springer (1997), 1–11.

[106] A. G. Myasnikov, A. Ushakov and D. W. Won, Power circuits, exponential algebra, and time complexity, *Internat. J. Algebra Comput.* **22** (2012), no. 6, 1–51.

[107] M.-J. Nederhof and G. Satta, The language intersection problem for non-recursive context-free grammars, *Inform. and Comput.* **192** (2004), no. 2, 172–184.

[108] P. S. Novikov, On the algorithmic unsolvability of the word problem in group theory, *Amer. Math. Soc. Transl. Ser. 2* **9** (1958), 1–122.

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

[110] H. Petersen, The membership problem for regular expressions with intersection is complete in LOGCFL, in: *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science* (STACS 2002), Lecture Notes in Comput. Sci. 2285, Springer (2002), 513–522.

[111] W. Plandowski, Testing equivalence of morphisms on context-free languages, in: *Proceedings of the 2nd Annual European Symposium on Algorithms* (ESA 1994), Lecture Notes in Comput. Sci. 855, Springer (1994), 460–470.

[112] W. Plandowski, Satisfiability of word equations with constants is in PSPACE, *J. ACM* **51** (2004), no. 3, 483–496.

[113] W. Plandowski and W. Rytter, Application of Lempel-Ziv encodings to the solution of word equations, in: *Proceedings of the 25th International Colloquium on Automata, Languages and Programming* (ICALP 1998), Lecture Notes in Comput. Sci. 1443, Springer (1998), 731–742.

[114] W. Plandowski and W. Rytter, Complexity of language recognition problems for compressed words, in: *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, Springer (1999), 262–272.

[115] R. Radicioni and A. Bertoni, Grammatical compression: compressed equivalence and other problems, *Discrete Math. Theor. Comput. Sci.* **12** (2010), no. 4, 109–126.

[116] D. Robinson, *Parallel algorithms for group word problems*, Ph.D. thesis, University of California, San Diego, 1993.

[117] J. M. Robson and V. Diekert, On quadratic word equations, in: *Proceedings of the16th Annual Symposium on Theoretical Aspects of Computer Science* (STACS 1999), Lecture Notes in Comput. Sci. 1563, Springer (1999), 217–226.

[118] W. Rytter, Application of Lempel-Ziv factorization to the approximation of grammar-based compression, *Theoret. Comput. Sci.* **302** (2003), no. 1–3, 211–222.

[119] W. Rytter, Grammar compression, LZ-encodings, and string algorithms with implicit input, in: *Proceedings of the 31st International Colloquium on Automata, Languages and Programming* (ICALP 2004), Lecture Notes in Comput. Sci. 3142, Springer (2004), 15–27.

[120] H. Sakamoto, A fully linear-time approximation algorithm for grammar-based compression, *J. Discrete Algorithms* **3** (2005), no. 2–4, 416–430.

[121] M. Schaefer, E. Sedgwick and D. Štefankovič, Algorithms for normal curves and surfaces, in: *Proceedings of the 8th Annual International Conference on Computing and Combinatorics* (COCOON 2002), Lecture Notes in Comput. Sci. 2387, Springer (2002), 370–380.

[122] M. Schaefer, E. Sedgwick and D. Štefankovič, Recognizing string graphs in NP, *J. Comput. Syst. Sci. Int.* **67** (2003), no. 2, 365–380.

[123] M. Schaefer, E. Sedgwick and D. Štefankovič, Computing Dehn twists and geometric intersection numbers in polynomial time, in: *Proceedings of the 20th Annual Canadian Conference on Computational Geometry* (CCCG 2008), 111–114.

[124] S. Schleimer, Polynomial-time word problems, *Comment. Math. Helv.* **83** (2008), no. 4, 741–765.

[125] M. Schmidt-Schauß, Polynomial equality testing for terms with shared substructures, Technical Report 21, Institut für Informatik, J. W. Goethe-Universität Frankfurt am Main, 2005.

[126] M. Schmidt-Schauß, Matching of compressed patterns with character-variables, in: *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications* (RTA 2012), Leibniz International Proceedings in Informatics 15 (2012), 272–287.

[127] M. Schmidt-Schauß, D. Sabel and A. Anis, Congruence closure of compressed terms in polynomial time, in: *Proceedings of the 8th International Symposium on Frontiers of Combining Systems* (FroCos 2011), Lecture Notes in Comput. Sci. 6989, Springer (2011), 227–242.

[128] M. Schmidt-Schauß and G. Schnitger, Fast equality test for straight-line compressed strings, *Inf. Process. Lett.* **112** (2012), no. 8–9, 341–345.

[129] H.-U. Simon, Word problems for groups and contextfree recognition, in: *Proceedings of Fundamentals of Computation Theory* (FCT 1979), Akademie-Verlag (1979), 417–422.

[130] J. Stillwell, *Classical Topology and Combinatorial Group Theory*, 2nd ed., Springer, 1995.

[131] I. H. Sudborough, On the tape complexity of deterministic context–free languages, *J. ACM* **25** (1978), no. 3, 405–414.

[132] A. Tiskin, Faster subsequence recognition in compressed strings, *J. Math. Sci. (N.Y.)* **158** (2009), no. 5, 759–769.

[133] A. Tiskin, Towards approximate matching in compressed strings: Local subsequence recognition, in: *Proceedings of the 6th International Computer Science Symposium in Russia* (CSR 2011), Lecture Notes in Comput. Sci. 6651, Springer (2011), 401–414.

[134] S. Toda, PP is as hard as the polynomial-time hierarchy, *SIAM J. Comput.* **20** (1991), no. 5, 865–877.

[135] A. Tozawa and Y. Minamide, Complexity results on balanced context-free languages, in: *Proceedings of the 10th International Conference on Foundations of Software Science and Computational Structures* (FOSSACS 2007), Lecture Notes in Comput. Sci. 4423, Springer (2007), 346–360.

[136] V. Vassilevska Williams, Multiplying matrices faster than Coppersmith-Winograd, in: *Proceedings of the 44th ACM Symposium on Theory of Computing* (STOC 2012), ACM (2012), 887–898.

[137] H. Venkateswaran, Properties that characterize LOGCFL, *J. Comput. Syst. Sci. Int.* **43** (1991), 380–404.

[138] K. W. Wagner, The complexity of combinatorial problems with succinct input representation, *Acta Inform.* **23** (1986), no. 3, 325–356.

[139] K. W. Wagner, Leaf language classes, in: *Proceedings of the 4th International Conference on Machines, Computations, and Universality* (MCU 2004), Lecture Notes in Comput. Sci. 3354, Springer (2005), 60–81.

[140] M. K. Warmuth and D. Haussler, On the complexity of iterated shuffle, *J. Comput. Syst. Sci. Int.* **28** (1984), no. 3, 345–358.

[141] D. T. Wise, Research announcement: the structure of groups with a quasiconvex hierarchy, *Electron. Res. Announc.Math. Sci.* **16** (2009), 44–55.

[142] T. Yamamoto, H. Bannai, S. Inenaga and M. Takeda, Faster subsequence and don't-care pattern matching on compressed texts, in: *Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching* (CPM 2011), Lecture Notes in Comput. Sci. 6661, Springer (2011), 309–322.

[143] A. C.-C. Yao, On the evaluation of powers, *SIAM J. Comput.* **5** (1976), no. 1, 100–103.

[144] J. Ziv and A. Lempel, A universal algorithm for sequential data compression, *IEEE Trans. Inf. Theory* **23** (1977), no. 3, 337–343.

Received June 6, 2012.

Author information

Markus Lohrey, Institut für Informatik, Universität Leipzig,
Augustusplatz 10–11, 04109 Leipzig, Germany.
E-mail: lohrey@informatik.uni-leipzig.de