Leaf languages and string compression[☆]

Markus Lohrey

Universität Leipzig, Institut für Informatik, Germany

Abstract

Tight connections between leaf languages and strings compressed by straight-line programs (SLPs) are established. It is shown that the compressed membership problem for a language L is complete for the leaf language class defined by L via logspace machines. A more difficult variant of the compressed membership problem for L is shown to be complete for the leaf language class defined by L via polynomial time machines. As a corollary, it is shown that there exists a fixed linear visibly pushdown language for which the compressed membership problem is PSPACE-complete. For XML languages, it is shown that the compressed membership problem is coNP-complete. Furthermore it is shown that the embedding problem for SLP-compressed strings is hard for PP (probabilistic polynomial time).

Keywords: Leaf languages, straight-line programs, compressed string, complexity theory

1. Introduction

Leaf languages were introduced in [9, 37] and have become an important concept in complexity theory. Let us consider a nondeterministic Turing machine M. For a given input x, one considers the yield string of the ordered computation tree (i.e., the string obtained by listing all leaves from left to right), where accepting (resp. rejecting) leaf configurations yield the letter 1 (resp. 0). This string is called the *leaf string* corresponding to the input x. For a given language $K \subseteq \{0,1\}^*$ let $\mathsf{LEAF}(M, K)$ denote the set of all inputs

[☆]This work is supported by the German Research Foundation (DFG) via the research project ALKODA.

Email address: lohrey@informatik.uni-leipzig.de (Markus Lohrey)

for M such that the corresponding leaf string belongs to K. By fixing K and taking for M all nondeterministic polynomial time machines, one obtains the polynomial time leaf language class $\mathsf{LEAF}_a^{\mathsf{P}}(K)$. The index a indicates that we allow Turing machines with arbitrary (non-balanced) computation trees. If we restrict to machines with balanced computation trees, we obtain the class $\mathsf{LEAF}_b^{\mathsf{P}}(K)$. See [17, 19, 21] for a discussion of the different shapes for computation trees.

Many complexity classes can be defined in a uniform way with this construction. For instance, $\mathsf{NP} = \mathsf{LEAF}_x^P(0^*1\{0,1\}^*)$ and $\mathsf{coNP} = \mathsf{LEAF}_x^P(1^*)$ for both $x = a$ and $x = b$. In [18], it was shown that $\mathsf{PSPACE} = \mathsf{LEAF}_b^{\mathsf{P}}(K)$ for a fixed regular language K. In [21], logspace leaf language classes $\mathsf{LEAF}_a^{\mathsf{L}}(K)$ and $\mathsf{LEAF}_b^L(K)$, where M varies over all (resp. all balanced) nondeterministic logspace machines, were investigated. Among other results, a fixed deterministic context-free language K with $\mathsf{PSPACE} = \mathsf{LEAF}_a^{\mathsf{L}}(K)$ was presented. In [10], it was shown that in fact a fixed deterministic *one-counter* language K as well as a fixed *linear* deterministic context-free language [20] suffices in order to obtain PSPACE. Here "linear" means that the pushdown automaton makes only one turn.

In [8, 36], a tight connection between leaf languages and computational problems for succinct input representations was established. More precisely, it was shown that the membership problem for a language $K \subseteq \{0,1\}^*$ is complete (w.r.t. polynomial time reductions in [8] and projection reductions in [36]) for the leaf language class $\mathsf{LEAF}_b^{\mathsf{P}}(K)$, if the input string x is represented by a Boolean circuit. A Boolean circuit $C(x_1, \ldots, x_n)$ with n inputs represents a string x of length 2^n in the natural way: the i-th position in x carries a 1 if and only if $C(a_1, \ldots, a_n) = 1$, where $a_1 \cdots a_n$ is the n-bit binary representation of i. In this paper we consider another compressed representation for strings, namely *straight-line programs* (SLPs) [33], which is compared to Boolean circuits more amenable to efficient algorithms. A straight-line program is a context-free grammar $\mathbb{A}$ that generates exactly one string val($\mathbb{A}$). In an SLP, repeated subpatterns in a string have to be represented only once by introducing a nonterminal for the pattern. An SLP with n productions can generate a string of length 2^n by repeated doubling. Hence, an SLP can be seen indeed as a compressed representation of the string it generates. Several other dictionary-based compressed representations, like for instance Lempel-Ziv (LZ) factorizations [40], can be converted in polynomial time into SLPs and vice versa [33]. This implies that complexity results can be transferee from SLP-encoded input strings to LZ-encoded

input strings.

Algorithmic problems for SLP-compressed strings were studied e.g. in [6, 25, 26, 28, 29, 32, 33]. A central problem in this context is the *compressed membership problem* for a language K: it is asked whether $\mathrm{val}(\mathbb{A}) \in K$ for a given SLP $\mathbb{A}$. In [26] it was shown that there exists a fixed linear deterministic context-free language with a PSPACE-complete compressed membership problem. A straightforward argument shows that for every language K, the compressed membership problem for K is complete for the logspace leaf language class $\mathsf{LEAF}_a^{\mathsf{L}}(K)$ (Proposition 4). As a consequence, the existence of a linear deterministic context-free language with a PSPACE-complete compressed membership problem [26] can be deduced from the above mentioned $\mathsf{LEAF}_a^{\mathsf{L}}$-characterization of PSPACE from [10], and vice versa. For polynomial time leaf languages, we reveal a more subtle relationship to SLPs. Recall that the *convolution* $u \otimes v$ of two strings $u, v \in \Sigma^*$ is the string over the paired alphabet $\Sigma \times \Sigma$ that is obtained from gluing u and v in the natural way (we cut off the longer string to the length of the shorter one). We define a fixed projection homomorphism $\rho : \{0,1\} \times \{0,1\} \to \{0,1\}$ such that for every language K, the problem of checking $\rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) \in K$ for two given SLPs $\mathbb{A}$, $\mathbb{B}$ is complete for the class $\mathsf{LEAF}_b^{\mathsf{P}}(K)$ (Corollary 6). By combining Corollary 6 with the main result from [18] ($\mathsf{PSPACE} = \mathsf{LEAF}_b^{\mathsf{P}}(K)$ for a certain regular language K), we obtain a regular language L for which it is PSPACE-complete to check whether the convolution of two SLP-compressed strings belongs to L (Corollary 8). Recently, the convolution of SLP-compressed strings was also studied in [6], where for every $n \geq 0$, SLPs $\mathbb{A}_n, \mathbb{B}_n$ of size $n^{O(1)}$ were constructed such that every SLP for the convolution $\mathrm{val}(\mathbb{A}_n) \otimes \mathrm{val}(\mathbb{B}_n)$ has size $\Omega(2^{n/2})$.

From Corollary 8 we obtain a strengthening of one of the above mentioned results from [10] ($\mathsf{PSPACE} = \mathsf{LEAF}_a^{\mathsf{L}}(K)$ for a linear deterministic context-free language K as well as a deterministic one-counter language K) to *visibly pushdown languages* [1]. The latter constitute a subclass of the deterministic context-free languages which received a lot of attention in recent years due to its nice closure and decidability properties. Visibly pushdown languages can be recognized by *deterministic* pushdown automata, where it depends only on the input symbol whether the automaton pushes or pops. Visibly pushdown languages were already introduced in [39] as input-driven languages. In [12] it was shown that every visibly pushdown language can be recognized in NC^1; thus the complexity is the same as for regular languages [2]. In contrast to this, there exist linear deterministic context-free languages as well as de-

terministic one-counter languages with an L-complete membership problem [20]. We show that there exists a linear visibly pushdown language with a PSPACE-complete compressed membership problem (Theorem 9). Together with Proposition 4, it follows that $\mathsf{PSPACE} = \mathsf{LEAF}_a^{\mathsf{L}}(K)$ for a linear visibly pushdown language K (Corollary 10).

In [31], nondeterministic finite automata (instead of polynomial time (resp. logspace) Turing-machines) were used as a device for generating leaf strings. This leads to the definition of the leaf language class $\mathsf{LEAF}^{\mathsf{FA}}(K)$. It was shown that $\mathsf{CFL} \subsetneq \mathsf{LEAF}^{\mathsf{FA}}(\mathsf{CFL}) \subseteq \mathsf{DSPACE}(n^2) \cap \mathsf{DTIME}(2^{O(n)})$, and the question for sharper upper and lower bounds was posed. Here we give a partial answer to this question. For the linear visibly pushdown language mentioned in the previous paragraph, the class $\mathsf{LEAF}^{\mathsf{FA}}(K)$ contains a PSPACE-complete language (Theorem 11).

Another application of the connection between SLP-compression and leaf languages is presented in Section 4.2. The compressed embedding problem (briefly COMPRESSED-EMBEDDING) asks for two given SLPs $\mathbb{A}$ and $\mathbb{B}$ whether $\mathrm{val}(\mathbb{A})$ is a *subsequence* of $\mathrm{val}(\mathbb{B})$, i.e., whether $\mathrm{val}(\mathbb{A})$ can be embedded into $\mathrm{val}(\mathbb{B})$ where consecutive positions in $\mathrm{val}(\mathbb{A})$ can be mapped to non-consecutive positions in $\mathrm{val}(\mathbb{B})$. In [25], it was shown that COMPRESSED-EMBEDDING is hard for $\mathsf{P}_{\parallel}^{\mathsf{NP}}$, which is the class of all problems that can be solved on a deterministic Turing-machine with access to an NP-oracle, where all queries are sent in parallel to the oracle (non-adaptive oracle access). A simplified proof can be found in [28]. Here we will strengthen the lower bound of $\mathsf{P}_{\parallel}^{\mathsf{NP}}$ to PP (Theorem 13). A language L belongs to the class PP (probabilistic polynomial time) if there exists a polynomial time NTM M such that $w \in L$ if and only if on input w the number of accepting computations is larger than the number of rejecting computations. In other words, the acceptance probability has to be larger than $1/2$. It is known that $\mathsf{P}_{\parallel}^{\mathsf{NP}} \subseteq \mathsf{PP}$ [4]. Moreover, Toda's famous theorem [35] states that P^{PP} contains the polynomial time hierarchy. Hence, PP-hardness of COMPRESSED-EMBEDDING implies that COMPRESSED-EMBEDDING is not contained in the polynomial time hierarchy unless the latter collapses. The best known upper bound for COMPRESSED-EMBEDDING is still PSPACE.

Finally, in Section 5 we consider *XML-languages* [5], which constitute a subclass of the visibly pushdown languages. XML-languages are generated by a special kind of context-free grammars (XML-grammars), where every right-hand side of a production is enclosed by a matching pair of brackets. XML-

grammars capture the syntactic features of XML document type definitions (DTDs), see [5]. We prove that, unlike for visibly pushdown languages, for every XML-language the compressed membership problem is in coNP and that there are coNP-complete instances.

A short version of this paper appeared in [27].

2. Preliminaries

Let Γ be a finite alphabet. The *empty word* is denoted by ε. Let $s = a_1 \cdots a_n \in \Gamma^*$ be a word over Γ ($n \geq 0$, $a_1, \ldots, a_n \in \Gamma$). The *length* of s is $|s| = n$. For $1 \leq i \leq n$ let $s[i] = a_i$ and for $1 \leq i \leq j \leq n$ let $s[i:j] = a_i a_{i+1} \cdots a_j$. If $i > j$ we set $s[i:j] = \varepsilon$. Moreover $s[:i] = s[1:i]$. If s is a suffix of the word u, then $u \setminus s$ denotes the unique string v with $u = vs$.

We denote with $\overline{\Gamma} = \{\overline{a} \mid a \in \Gamma\}$ a disjoint copy of Γ. For $\overline{a} \in \overline{\Gamma}$ let $\overline{\overline{a}} = a$. For $w = a_1 \cdots a_n \in (\Gamma \cup \overline{\Gamma})^*$ let $\overline{w} = \overline{a_n} \cdots \overline{a_1}$. For two strings $u, v \in \Gamma^*$ we define the *convolution* $u \otimes v \in (\Gamma \times \Gamma)^*$ as the string of length $\ell = \min\{|u|, |v|\}$ with $(u \otimes v)[i] = (u[i], v[i])$ for all $1 \leq i \leq \ell$.

A sequence $(u_1, \ldots, u_n)$ of natural numbers is *superdecreasing* if $u_i > u_{i+1} + \cdots + u_n$ for all $1 \leq i \leq n$. An instance of the *subsetsum problem* is a tuple $(t, w_1, \ldots, w_k)$ of binary coded natural numbers. It is a positive instance if there are $x_1, \ldots, x_k \in \{0, 1\}$ such that $t = x_1 w_1 + \cdots + x_k w_k$. Subsetsum is a classical NP-complete problem, see e.g. [13]. The *superdecreasing subsetsum* problem is the restriction of subsetsum to instances $(t, w_1, \ldots, w_k)$, where $(w_1, \ldots, w_k)$ is superdecreasing. In [22] it was shown that superdecreasing subsetsum is P-complete.[1] In fact, something more general is shown in [22]: Let $C(x_1, \ldots, x_m)$ be a Boolean circuit with variable input gates $x_1, \ldots, x_m$ (and some additional input gates that are set to fixed Boolean values). Then from $C(x_1, \ldots, x_m)$ an instance $(t(x_1, \ldots, x_m), w_1, \ldots, w_k)$ of superdecreasing subsetsum is constructed. Here, $t(x_1, \ldots, x_m) = t_0 + x_1 t_1 + \cdots + x_m t_m$ is a linear expression such that:

- $t_1 > t_2 > \cdots > t_m$ and the t_i are pairwise distinct powers of 4. Hence also the sequence $(t_1, \ldots, t_m)$ is superdecreasing.
- For all $a_1, \ldots, a_m \in \{0, 1\}$: $C(a_1, \ldots, a_m)$ evaluates to true if and only

[1] In fact, [22] deals with the *superincreasing* subsetsum problem. But this is only a nonessential detail. For our purpose, superdecreasing sequences are more convenient.

if there exist $b_1, \ldots, b_k \in \{0,1\}$ such that $t_0 + a_1 t_1 + \cdots + a_m t_m = b_1 w_1 + \cdots + b_k w_k$.

- $t_0 + t_1 + \cdots + t_m \leq w_1 + \cdots + w_k$

We encode a superdecreasing sequence $(w_1, \ldots, w_k)$ of natural numbers by the string $S(w_1, \ldots, w_k) \in \{0,1\}^*$ of length $w_1 + \cdots + w_k + 1$ such that for all $0 \leq p \leq w_1 + \cdots + w_k$:

$$S(w_1, \ldots, w_k)[p+1] = \begin{cases} 1 & \text{if } \exists x_1, \ldots, x_k \in \{0,1\} : p = x_1 w_1 + \cdots + x_k w_k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Since $(w_1, \ldots, w_k)$ is a superdecreasing sequence, the number of 1's in the string $S(w_1, \ldots, w_k)$ is 2^k.

The lexicographic order on $\mathbb{N}^*$ is denoted by $\preceq$, i.e. $u \preceq v$ if either u is a prefix of v or there exist $w, x, y \in \mathbb{N}^*$ and $i, j \in \mathbb{N}$ such that $u = wix$, $v = wjy$, and $i < j$. A *finite ordered tree* is a finite set $T \subseteq \mathbb{N}^*$ such that for all $w \in \mathbb{N}^*$, $i \in \mathbb{N}$: if $wi \in T$ then $w, wj \in T$ for every $0 \leq j < i$. The set of *children* of $u \in T$ is $u\mathbb{N} \cap T$. A node $u \in T$ is a leaf of T if it has no children. We say that T is a *full binary tree* if (i) every node has at most two children, and (ii) every maximal path in T has the same number of branching nodes (i.e., nodes with exactly two children). A *left initial segment of a full binary tree* is a tree T such that there exists a full binary tree T' and a leaf $v \in T'$ such that $T = \{u \in T' \mid u \preceq v\}$. We can assume that the path from the root ε to the leaf v moves from the first branching node u on the path to its right child $u1$ (otherwise we could replace T by a full binary tree with fewer branching nodes).

Example 1. *The whole tree T in Figure 1 is a full binary tree, whereas the thick part T' is a left initial segment of T, which consists of all nodes $u \preceq 0001000$.*

2.1. Leaf languages

We assume some basic background in complexity theory [30]. In the following, we introduce basic concepts related to leaf languages, more details can be found in [9, 17, 18, 19, 21]. A nondeterministic Turing-machine (NTM) M is *adequate*, if (i) for every input $w \in \Sigma^*$, M does not have an infinite computation on input w and (ii) the set of finitely many transition tuples

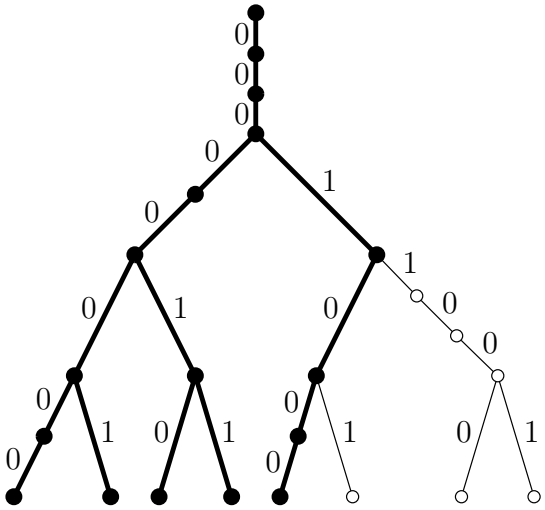

Figure 1: A full binary tree T with a left initial segment T' (the thick part)

of M is linearly ordered. For an input w for M, we define the computation tree by unfolding the configuration graph of M from the initial configuration. By condition (i) and (ii), the computation tree can be identified with a finite ordered tree $T(w) \subseteq \mathbb{N}^*$. For $u \in T(w)$ let $q(u)$ be the M-state of the configuration that is associated with the tree node u. Then, the leaf string $\mathrm{leaf}(M, w)$ is the string $\alpha(q(v_1)) \cdots \alpha(q(v_k))$, where $v_1, \ldots, v_k$ are all leaves of $T(w)$ listed in lexicographic order, and $\alpha(q) = 1$ (resp. $\alpha(q) = 0$) if q is an accepting (resp. rejecting) state.

An adequate NTM M is called *balanced*, if for every input $w \in \Sigma^*$, $T(w)$ is a left initial segment of a full binary tree. With a language $K \subseteq \{0,1\}^*$ we associate the language

$$\mathsf{LEAF}(M, K) = \{w \in \Sigma^* \mid \mathrm{leaf}(M, w) \in K\}.$$

Finally, we associate four complexity classes with $K \subseteq \{0,1\}^*$:

$$\begin{aligned}
\mathsf{LEAF}_a^{\mathsf{P}}(K) &= \{\mathsf{LEAF}(M, K) \mid M \text{ is an adequate polynomial time NTM}\} \\
\mathsf{LEAF}_b^{\mathsf{P}}(K) &= \{\mathsf{LEAF}(M, K) \mid M \text{ is a balanced polynomial time NTM}\} \\
\mathsf{LEAF}_a^{\mathsf{L}}(K) &= \{\mathsf{LEAF}(M, K) \mid M \text{ is an adequate logarithmic space NTM}\} \\
\mathsf{LEAF}_b^{\mathsf{L}}(K) &= \{\mathsf{LEAF}(M, K) \mid M \text{ is a balanced logarithmic space NTM}\}
\end{aligned}$$

The first two (resp. last two) classes are closed under polynomial time (resp. logspace) reductions.

2.2. Straight-line programs

Following [33], 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 variables, Γ is the set of terminals, $S \in V$ is the initial variable, and $P \subseteq V \times (V \cup \Gamma)^*$ is the finite set of productions) such that: (i) for every $A \in V$ there exists exactly one production of the form $(A, \alpha) \in P$ for $\alpha \in (V \cup \Gamma)^*$, and (ii) the relation $\{(A, B) \in V \times V \mid (A, \alpha) \in P, B \text{ occurs in } \alpha\}$ is acyclic. Clearly, the language generated by the SLP $\mathbb{A}$ consists of exactly one word that is denoted by $\mathrm{val}(\mathbb{A})$. More generally, from every variable $A \in V$ we can generate exactly one word that is denoted by $\mathrm{val}_{\mathbb{A}}(A)$ (thus $\mathrm{val}(\mathbb{A}) = \mathrm{val}_{\mathbb{A}}(S)$). We omit the index $\mathbb{A}$ if the underlying SLP is clear from the context. The size of $\mathbb{A}$ is $|\mathbb{A}| = \sum_{(A,\alpha)\in P} |\alpha|$. Every SLP can be transformed in polynomial time into an equivalent SLP in *Chomsky normal form*, i.e., all productions have the form (A, a) with $a \in \Gamma$ or (A, BC) with $B, C \in V$.

Example 2. *Consider the SLP $\mathbb{A}$ over $\{a, b\}$ that consists of the 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 variable is A_7. Then $\mathrm{val}(\mathbb{A}) = abaababaabaab$, which is the 7-th Fibonacci word. The SLP $\mathbb{A}$ is in Chomsky normal form and $|\mathbb{A}| = 12$.*

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 $\lceil \log(i) \rceil$ many ordinary productions.

A *composition system* $\mathbb{A} = (V, \Gamma, S, P)$ is defined analogously to an SLP, but in addition to productions of the form $A \to \alpha$ ($A \in V$, $\alpha \in (V \cup \Gamma)^*$) it may also contain productions of the form $A \to B[i : j]$ for $B \in V$ and $i, j \in \mathbb{N}$ [14]. For such a production we define $\mathrm{val}_{\mathbb{A}}(A) = \mathrm{val}_{\mathbb{A}}(B)[i : j]$.[2] The size of a production $A \to B[i : j]$ is $\lceil \log(i) \rceil + \lceil \log(j) \rceil$. As for SLPs we define $\mathrm{val}(\mathbb{A}) = \mathrm{val}_{\mathbb{A}}(S)$. In [16], Hagenah presented a polynomial time algorithm, which transforms a given composition system $\mathbb{A}$ into an SLP $\mathbb{B}$ with $\mathrm{val}(\mathbb{A}) = \mathrm{val}(\mathbb{B})$.

Let us state some simple algorithmic problems that can easily be solved in polynomial time (but not in deterministic logspace under reasonable com-

[2] In [14], a slightly more restricted formalism, where all productions have the form $A \to a \in \Gamma$ or $A \to B[j :]C[: i]$, was introduced. But this definition is easily seen to be equivalent to our formalism.

plexity theoretic assumptions: problem (a) is #L-complete, problems (b) and (c) are complete for functional P [25]):

(a) Given an SLP $\mathbb{A}$, compute $|\mathrm{val}(\mathbb{A})|$.

(b) Given an SLP $\mathbb{A}$ and a number $i \in \{1, \ldots, |\mathrm{val}(\mathbb{A})|\}$, compute $\mathrm{val}(\mathbb{A})[i]$.

(c) Given an SLP $\mathbb{A}$, a terminal symbol a, and a number i, compute the position in $\mathrm{val}(\mathbb{A})$ of the i-th a in $\mathrm{val}(\mathbb{A})$ (if it exists).

(d) Given an SLP $\mathbb{A}$ and two number $i, j \in \{1, \ldots, |\mathrm{val}(\mathbb{A})|\}$ with $i \leq j$, compute an SLP $\mathbb{B}$ with $\mathrm{val}(\mathbb{B}) = \mathrm{val}(\mathbb{A})[i : j]$.

(e) Given an SLP $\mathbb{A}$ over the terminal alphabet Γ and a homomorphism $\rho : \Gamma^* \to \Sigma^*$, compute an SLP $\mathbb{B}$ such that $\mathrm{val}(\mathbb{B}) = \rho(\mathrm{val}(\mathbb{A}))$.

Algorithms for producing a small SLP for a given input string can be found in [11, 34]. On the other hand, it is NP-complete to decide whether for a given string w and a number n there exists an SLP $\mathbb{A}$ with $\mathrm{val}(\mathbb{A}) = w$ and $|\mathbb{A}| \leq n$ [11]. In [32], Plandowski presented a polynomial time algorithm for testing whether $\mathrm{val}(\mathbb{A}) = \mathrm{val}(\mathbb{B})$ for two given SLPs $\mathbb{A}$ and $\mathbb{B}$. A cubic algorithm can be found in [24]. For a language L, we denote with $\mathrm{CMP}(L)$ (*compressed membership problem* for L) the following computational problem:

INPUT: An SLP $\mathbb{A}$ over the terminal alphabet Σ
QUESTION: $\mathrm{val}(\mathbb{A}) \in L$?

The following result was shown in [3, 21, 29]:

Theorem 3. *For every regular language L, $\mathrm{CMP}(L)$ can be decided in polynomial time. Moreover, there exists a fixed regular language L such that $\mathrm{CMP}(L)$ is P-complete.*

In [25], we constructed in logspace from a given superdecreasing sequence $(w_1, \ldots, w_k)$ an SLP $\mathbb{A}$ over $\{0, 1\}$ such that $\mathrm{val}(\mathbb{A}) = S(w_1, \ldots, w_k)$, where $S(w_1, \ldots, w_k)$ is the string-encoding from (1). Let us briefly repeat the construction. For $1 \leq i \leq k$ let

$$d_i = \begin{cases} w_k - 1 & \text{if } i = k \\ w_i - (w_{i+1} + \cdots + w_k) - 1 & \text{if } 1 \leq i \leq k-1 \end{cases} \tag{2}$$

Moreover define strings $S_1, \ldots, S_k \in \{0,1\}^*$ by the recursion

$$S_k = 10^{d_k}1 \qquad S_i = S_{i+1}0^{d_i}S_{i+1}\ (1 \leq i \leq k-1). \tag{3}$$

Then $S(w_1, \ldots, w_k) = S_1$. Note that the SLP that implements the recursion (3) can be constructed in logspace from the binary encoded sequence $(w_1, \ldots, w_k)$ (in [25] only the existence of an NC-construction is claimed). The only nontrivial step is the calculation of all partial sums $w_{i+1} + \cdots + w_k$ for $1 \leq i \leq k-1$ in (2). This is possible with a logspace transducer. In fact iterated addition (the problem of computing a given sum $n_1 + \cdots + n_\ell$ of binary coded integers n_i) can be accomplished in uniform TC^0, see e.g. [38, Theorem 1.37].

3. Straight-line programs versus leaf languages

In [8, 36], it was shown that the membership problem for a language $K \subseteq \{0,1\}^*$ is complete (w.r.t. polynomial time reductions in [8] and projection reductions in [36]) for the leaf language class $\mathsf{LEAF}_b^{\mathsf{P}}(K)$, if the input string is represented by a Boolean circuit. For SLP-compressed strings, we obtain a similar result:

Proposition 4. *For every language $K \subseteq \{0,1\}^*$, the problem* $\mathrm{CMP}(K)$ *is complete w.r.t. logspace reductions for the class* $\mathsf{LEAF}_a^{\mathsf{L}}(K)$.

Proof. For $\mathrm{CMP}(K) \in \mathsf{LEAF}_a^{\mathsf{L}}(K)$, it suffices to note that for an input SLP $\mathbb{A} = (V, \{0,1\}, S, P)$, an adequate logspace NTM M can behave such that the computation tree on this input is just the derivation tree of $\mathbb{A}$. For hardness, let $L \in \mathsf{LEAF}_a^{\mathsf{L}}(K)$. Hence, there exists an adequate logspace NTM M such that $w \in L$ if and only if $\mathrm{leaf}(M, w) \in K$. Let us take an input w with $|w| = n$ and assume that M operates in space $c \cdot \log(n)$. We construct in logspace an SLP $\mathbb{A} = (V, \{0,1\}, S, P)$ such that $\mathrm{val}(\mathbb{A}) = \mathrm{leaf}(M, w)$. Here, V is the set of all configurations of length $c \cdot \log(n)$ and S is the initial configuration on input w. Finally, the set P consists of the following productions:

- $c \to c_1 \cdots c_k$, where $c \in V$ and $c_1, \ldots, c_k$ are the successor configurations of c in this order.
- $c \to 0$, if c is a rejecting configuration.
- $c \to 1$ if c is an accepting configuration.

The construction ensures that $\mathrm{val}(\mathbb{A}) = \mathrm{leaf}(M, w)$. $\square$

We now prove a more subtle relationship between SLP-compressed strings and polynomial time leaf languages. Let $\rho : (\{0,1\} \times \{0,1\})^* \to \{0,1\}^*$ be the morphism defined by

$$\rho(0,0) = \rho(0,1) = \varepsilon, \quad \rho(1,0) = 0, \quad \rho(1,1) = 1. \tag{4}$$

Theorem 5. *Let M be a balanced polynomial time NTM. From a given input $w \in \Sigma^*$ for M we can construct in polynomial time two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$ and $\mathrm{leaf}(M, w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$.*

Proof. Let w be an input for M. Our construction consists of five steps:

Step 1. By simulating M along the left-most computation path, we can compute in polynomial time the maximal number m of branching nodes along a maximal path in the computation tree $T(w)$. Since M is balanced, there is a full binary T and a leaf $v \in \{0,1\}^*$ of T such that $T(w)$ consists of exactly those nodes of T that are lexicographically less or equal to v. The tree T has exactly m branching nodes on each maximal path. Hence, every leaf of T can be addressed by a bit string $u \in \{0,1\}^m$. Let $r \in \{0,1\}^m$ be the bit string that addresses the leaf v (r results from v by removing those 0's that correspond to non-branching nodes of T). The string r can be computed in polynomial time as follows: We follow the right-most computation path of $T(w)$ and store a number b, which is initialized to m. Each time, we encounter a branching node v in $T(w)$, we compute the number b' of branching nodes along a maximal path in the subtree rooted at $v0$ (the left child of v). Note that the subtree rooted at $v0$ is a full binary tree. Then we print $0^{b-b'-1}1$, set $b := b'$, move to the right child $v1$, and continue. Note that we always print 1 at the first branching node of $T(w)$. When we finally reach a leaf in $T(w)$, we print $0^{b'}$ and terminate.

Note that every leaf of $T(w)$ can be specified by a bit string $u \in \{0,1\}^m$ with $u \preceq r$.

Before we continue with Step 2, let us consider as an example the full binary tree T from Figure 1. The thick part T' is a left initial segment that consists of all nodes $u \preceq v$, where $v = 0001000$. The tree T has $m = 3$ branching nodes on each maximal path. Hence, every leaf of T can be addressed by a bit string of length 3. The leaf v is addressed by $r = 100$. The right-most path of T' (which leads to the leaf $v = 0001000$) has only one

branching node in T' (namely 000). When walking down this path, we print 1 at the branching node 000 and set $b := 2$ (the number of branching nodes along a maximal path in the subtree rooted at 0000). Hence, when we reach the leaf 0001000 we print 00. Thus, in total we output the string $r = 100$. The 5 leafs of T' can be addressed by the bit strings 000, 001, 010, 011, 100.

Step 2. Note that given a bit string $u \in \{0,1\}^m$ with $u \preceq r$, we can compute in polynomial time the leaf of $T(w)$ that is addressed by the bit string u. In particular, we can check whether M accepts in the leaf that is addressed by u. Hence, using the classical Cook-Levin construction (see e.g. [30]), we can compute in logspace a Boolean circuit $C_w(x_1, \ldots, x_m)$ from w such that for all $a_1, \ldots, a_m \in \{0,1\}$: $C_w(a_1, \ldots, a_m)$ evaluates to true if and only if the machine M accepts at the leaf of $T(w)$ that is addressed by the bit string $a_1 \cdots a_m$ (in case $r \prec a_1 \cdots a_m$ the concrete value $C_w(a_1, \ldots, a_m)$ does not matter). The circuit $C_w(x_1, \ldots, x_m)$ has input gates $x_1, \ldots, x_m$ together with some additional input gates that carry fixed input bits.

Step 3. Using the construction from [22] (see Section 2), we now transform the circuit $C_w(x_1, \ldots, x_m)$ in logspace into a superdecreasing subsetsum instance $(t(x_1, \ldots, x_m), w_1, \ldots, w_k)$, where $w_1, \ldots, w_k \in \mathbb{N}$ and $t(x_1, \ldots, x_m) = t_0 + x_1 t_1 + \cdots + x_m t_m$ such that

- $t_1 > t_2 > \cdots > t_m$ and the sequence $(t_1, \ldots, t_m)$ is superdecreasing,
- for all $a_1, \ldots, a_m \in \{0,1\}$: $C_w(a_1, \ldots, a_m)$ evaluates to true if and only if there exist $b_1, \ldots, b_k \in \{0,1\}$ such that $t_0 + a_1 t_1 + \cdots + a_m t_m = b_1 w_1 + \cdots + b_k w_k$,
- $t_0 + t_1 + \cdots + t_m \leq w_1 + \cdots + w_k$.

Step 4. By [25] (see the end of Section 2.2 of this paper), we can construct in logspace from the two superdecreasing sequences $(t_1, \ldots, t_m), (w_1, \ldots, w_k)$ SLPs $\mathbb{A}'$ and $\mathbb{B}$ over $\{0,1\}$ such that $\mathrm{val}(\mathbb{A}') = S(t_1, \ldots, t_m)$ and $\mathrm{val}(\mathbb{B}) = S(w_1, \ldots, w_k)$ (see (1)). Note that $t_0 + |\mathrm{val}(\mathbb{A}')| = t_0 + t_1 + \cdots + t_m + 1 \leq w_1 + \cdots + w_k + 1 = |\mathrm{val}(\mathbb{B})|$.

Step 5. Recall that $r = r_1 \cdots r_m$ addresses the right-most leaf of $T(w)$. Let $p = r_1 t_1 + \cdots + r_m t_m$. Thus, if r is the lexicographically n-th string in $\{0,1\}^m$ (this means that $T(w)$ has exactly n leaves), then $p+1$ is the position of the n-th 1 in $\mathrm{val}(\mathbb{A}')$. From the SLP $\mathbb{A}'$ we can finally compute in polynomial

time an SLP $\mathbb{A}$ with

$$\mathrm{val}(\mathbb{A}) = 0^{t_0}\ S(t_1, \ldots, t_m)[1 : p+1]\ 0^{w_1+\cdots+w_k-t_0-p}.$$

Then $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$ and for all positions $q \in \{0, \ldots, |\mathrm{val}(\mathbb{A})| - 1\}$:

$$\begin{aligned}
\mathrm{val}(\mathbb{A})[q+1] = 1 \iff\ & \exists a_1, \ldots, a_m \in \{0,1\} : a_1 \cdots a_m \preceq r\ \wedge \\
& q = t_0 + a_1 t_1 + \cdots + a_m t_m \\
\mathrm{val}(\mathbb{B})[q+1] = 1 \iff\ & \exists b_1, \ldots, b_k \in \{0,1\} : q = b_1 w_1 + \cdots + b_k w_k
\end{aligned}$$

Due to the definition of the projection ρ in (4), we finally have

$$\rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) = \prod_{x \in \{0,1\}^m,\, x \preceq r} \alpha(x),$$

where $\alpha(x) \in \{0,1\}$ and $\alpha(x_1 \cdots x_m) = 1$ if and only if there exist $b_1, \ldots, b_k \in \{0,1\}$ such that $t_0 + x_1 t_1 + \cdots + x_m t_m = b_1 w_1 + \cdots + b_k w_k$. Thus, $\alpha(x_1 \cdots x_m) = 1$ if and only if M accepts at the leaf addressed by $x_1 \cdots x_m \preceq r$. Hence, $\mathrm{leaf}(M, w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$. □

Corollary 6. *For every language $K \subseteq \{0,1\}^*$, the following problem is complete for the class $\mathsf{LEAF}_b^{\mathsf{P}}(K)$ w.r.t. polynomial time reductions:*

INPUT: Two SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$
QUESTION: $\rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) \in K$?

Proof. Hardness w.r.t. polynomial time reductions for the class $\mathsf{LEAF}_b^{\mathsf{P}}(K)$ follows directly from Theorem 5. For containment in $\mathsf{LEAF}_b^{\mathsf{P}}(K)$, we describe a balanced NTM M such that $\mathrm{leaf}(\langle \mathbb{A}, \mathbb{B} \rangle, M) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$ for two given input SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$:

- M first computes deterministically $\ell = \min\{|\mathrm{val}(\mathbb{A})|, |\mathrm{val}(\mathbb{B})|\}$ and the number m of occurrences of 1 in $\mathrm{val}(\mathbb{A})[1, \ell]$.
- Next, M branches nondeterministically such that the computation tree becomes a left initial segment of a full binary tree with m leaves. In the configuration that corresponds to the i-th leaf we store the binary representation of i.
- From the i-th leaf, M now computes deterministically the position p of the i-th 1 in $\mathrm{val}(\mathbb{A})$.

- Then, M computes deterministically $a = \mathrm{val}(\mathbb{B})[p]$. If $a = 1$ then M accepts, otherwise M rejects. □

In order to get completeness results w.r.t. logspace reductions in the next section, we need a variant of Theorem 5. We say that an NTM is *fully balanced*, if for every input w, $T(w)$ is a full binary tree (and not just a left initial segment of a full binary tree).

Theorem 7. *Let M be a fully balanced polynomial time NTM such that for some polynomial $p(n)$ and for every w, every maximal path in the computation tree $T(w)$ 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* $\mathrm{leaf}(M, w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$ *and* $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$.

Proof. The only steps in the proof of Theorem 5 that cannot be done in logspace (unless $\mathsf{L} = \mathsf{P}$), are step 1 and step 5. Under the additional assumptions of Theorem 7, we have to compute in step 1 only $m = p(|w|)$, which is possible in logspace, since $p(n)$ is a fixed polynomial. In step 5, we have to compute an SLP $\mathbb{A}$ with

$$\mathrm{val}(\mathbb{A}) = 0^{t_0}\ S(t_1, \ldots, t_m)\ 0^{w_1+\cdots+w_k-(t_0+\cdots+t_m)}.$$

This is possible in logspace, since $S(t_1, \ldots, t_m)$ can be constructed in logspace. □

4. Applications

In this section, we apply the techniques developed in the previous section to (i) compressed membership problems and (ii) the compressed embedding problem.

4.1. Compressed membership problems

Corollary 8. *There exists a fixed regular language $L \subseteq (\{0,1\} \times \{0,1\})^*$ such that the following problem is* PSPACE*-complete w.r.t. logspace reductions:*

INPUT: Two SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$
QUESTION: $\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in L$*?*

PROOF. Membership in PSPACE is obvious. Let us prove the lower bound. By [18], there exists a regular language $K \subseteq \{0,1\}^*$ and a balanced polynomial time NTM M such that the language $\mathsf{LEAF}(M,K)$ is PSPACE-complete. Using the padding technique from [21, Proposition 2.3], we can even assume that M is fully balanced and that the number of branching nodes along every maximal path of $T(w)$ is exactly $p(|w|)$ for a polynomial $p(n)$. Let $L = \rho^{-1}(K)$, which is a fixed regular language, since ρ from (4) is a fixed morphism. Let w be an input for M. By Theorem 7, we can construct in logspace two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that $\rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) = \mathrm{leaf}(M,w)$. Hence, the corollary follows from

$$\begin{aligned} w \in \mathsf{LEAF}(M,K) &\iff \mathrm{leaf}(M,w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) \in K \\ &\iff \mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in L \end{aligned}$$

□

From Theorem 7 it follows that even the set of all SLP-pairs $\langle \mathbb{A}, \mathbb{B} \rangle$ with $\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in L$ and $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$ (or $|\mathrm{val}(\mathbb{A})| \leq |\mathrm{val}(\mathbb{B})|$) is PSPACE-complete w.r.t. logspace reductions. We need this technical detail in the proof of the next theorem.

In [26] we have constructed a fixed linear deterministic context-free language with a PSPACE-complete compressed membership problem. As noted in the introduction, this result follows also from PSPACE $= \mathsf{LEAF}^{\mathsf{L}}_a(K)$ for a linear deterministic context-free language K [10] together with Proposition 4. Here, we sharpen this result to linear visibly pushdown 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) [1] 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 the set $Q \times \Sigma^* \times (\Gamma \setminus \{\bot\})^* \bot$. 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:

[3] In [1], 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.

- $a \in \Sigma_c$ 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 exists 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 exists at most one $q \in Q$ with $(p, a, \gamma, q) \in \Delta$.

For every VPA V there exists a deterministic VPA V' with $L(V) = L(V')$ [1]. 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*.

By a classical result from [15], 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 [12] and is therefore of the same complexity as for regular languages [2]. 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 3):

Theorem 9. *There exists a linear visibly pushdown language K such that* $\mathrm{CMP}(K)$ *is* PSPACE*-complete w.r.t. logspace reductions.*

Proof. Membership in PSPACE holds even for an arbitrary context-free language K [33]. For the lower bound, we reduce the problem from Corollary 8 to $\mathrm{CMP}(K)$ for some linear visibly pushdown language K. Let $L \subseteq (\{0,1\} \times \{0,1\})^*$ be the regular language from Corollary 8 and let $A = (Q, \{0,1\} \times \{0,1\}, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) = L$. W.l.o.g. assume that the initial state q_0 has no incoming transitions.

From two given SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$ we can easily construct in logspace an SLP $\mathbb{C}$ over $\Sigma = \{0, 1, \overline{0}, \overline{1}\}$ with $\mathrm{val}(\mathbb{C}) = \overline{\mathrm{val}(\mathbb{B})}\,\mathrm{val}(\mathbb{A})$. Let

$V = (Q, q_0, \{\bot, 0, 1\}, \bot, \Delta, F)$ be the 1-turn VPA over $(\{\overline{0}, \overline{1}\}, \{0, 1\})$ with the following transitions:

$$\Delta = \{(q_0, \overline{x}, q_0, x) \mid x \in \{0, 1\}\} \cup \{(q, x, y, p) \mid x, y \in \{0, 1\}, \delta(q, (x, y)) = p\}.$$

Thus, V can only read words of the form $\overline{v}u$ with $u, v \in \{0, 1\}^*$ and $|v| \geq |u|$ (recall that q_0 has no incoming transitions). When reading such a word $\overline{v}u$, V first pushes the word v (reversed) on the stack and then simulates the automaton A on the string $u \otimes v$ and thereby pops from the stack. Let $K = L(V)$. From the construction of V, we obtain

$$\mathrm{val}(\mathbb{C}) = \overline{\mathrm{val}(\mathbb{B})}\,\mathrm{val}(\mathbb{A}) \in K \quad \Longleftrightarrow \quad \mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in L(A) \wedge |\mathrm{val}(\mathbb{A})| \leq |\mathrm{val}(\mathbb{B})|.$$

By Corollary 8 (and the remark after the proof), this concludes the proof. □

Proposition 4 and Theorem 9 imply:

Corollary 10. $\mathsf{PSPACE} = \mathsf{LEAF}_a^{\mathsf{L}}(K)$ *for some linear visibly pushdown language* K*.*

In [31], a suitable variant of nondeterministic finite automata were used as leaf string generating devices. A *finite leaf automaton* (FLA) is a tuple $A = (Q, \Sigma, \Gamma, \delta, \rho, q_0)$, where Q is a finite set of states, Σ and Γ are finite alphabets, $\delta : Q \times \Sigma \to Q^+$ is the transition mapping, $\rho : Q \to \Gamma$ is the output mapping, and $q_0 \in Q$ is the initial state. For every state $q \in Q$ and every input word $w \in \Sigma^*$, we define by induction the string $\widehat{\delta}(q, w)$ as follows: $\widehat{\delta}(q, \varepsilon) = q$ and $\widehat{\delta}(q, au) = \widehat{\delta}(q_1, u) \cdots \widehat{\delta}(q_n, u)$ if $a \in \Sigma$ and $\delta(q, a) = q_1 \cdots q_n$. Let $\mathrm{leaf}(A, w) = \rho(\widehat{\delta}(q_0, w))$, where $\rho : Q \to \Gamma$ is extended to a morphism on Q^*. For $K \subseteq \Gamma^*$ let $\mathsf{LEAF}(A, K) = \{w \in \Sigma^* \mid \mathrm{leaf}(A, w) \in K\}$. Finally, let $\mathsf{LEAF}^{\mathsf{FA}}(K) = \{\mathsf{LEAF}(A, K) \mid A \text{ is an FLA }\}$.

Theorem 11. *There exists a fixed linear visibly pushdown language* K *and an FLA* A *such that* $\mathsf{LEAF}(A, K)$ *is* PSPACE*-complete w.r.t. logspace reductions.*

Proof. We use the linear visibly pushdown language K from the proof of Theorem 9. Notice that the question whether $\mathrm{val}(\mathbb{C}) \in K$ is already PSPACE-complete for a quite restricted class of SLPs. By tracing the construction of

the SLP $\mathbb{C}$ (starting from the proof of Theorem 7), we see that already the following question is PSPACE-complete w.r.t. logspace reductions for the language K:

INPUT: Two superdecreasing sequences $(t_1, \ldots, t_m)$, $(w_1, \ldots, w_k)$, and a number t_0 (all numbers are encoded binary)
QUESTION: Does the following string belong to K?

$$\overline{S(w_1, \ldots, w_k)}\; 0^{t_0}\; S(t_1, \ldots, t_m)\; 0^{w_1+\cdots+w_k-(t_0+\cdots+t_m)} \tag{5}$$

Here we use again the string encoding of superdecreasing sequences from (1). So, it remains to find a fixed FLA A with the following property: from given input data $t_0, (t_1, \ldots, t_m), (w_1, \ldots, w_k)$ as above we can construct in logspace a string w such that $\mathrm{leaf}(A, w)$ is exactly the string (5).

We only present an FLA A and a logspace construction of a string w from a superdecreasing sequence $(w_1, \ldots, w_k)$ such that

$$\mathrm{leaf}(A, w) = S(w_1, \ldots, w_k).$$

From this FLA, an FLA for producing the leaf string (5) can be easily derived. We use the following (logspace computable) exponent-encoding of a natural number d:

$$e(d) = a^{e_1}\$a^{e_2}\$\cdots a^{e_{m-1}}\$a^{e_m}\widetilde{\$} \in \{a, \$\}^*\widetilde{\$},$$

where the numbers $e_1, e_2, \ldots, e_m$ are uniquely determined by: $e_1 < e_2 < \cdots < e_m$ and $d = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_m}$. Next, we derive (in logspace) from the superdecreasing sequence $(w_1, \ldots, w_k)$ the sequence $(d_1, \ldots, d_k)$ of differences as defined in (2) and encode it by the string

$$e(d_1, \ldots, d_k) = \left(\prod_{i=1}^{k-1} \#e(d_i)\right)\widetilde{\#}e(d_k)$$

over the alphabet $\Sigma = \{a, \$, \widetilde{\$}, \#, \widetilde{\#}\}$. Our fixed FLA is

$$A = (\{q_0, p_r, p_\ell, r_0, r_1\}, \Sigma, \{0, 1\}, \delta, \rho, q_0),$$

where the transition function δ is defined as follows:

$$\begin{aligned}
\delta(q_0, \#) &= q_0 p_r q_0 & \delta(p_r, a) &= p_\ell p_r \\
\delta(q_0, x) &= q_0 \text{ for } x \in \{a, \$, \widetilde{\$}\} & \delta(p_r, \$) &= r_0 p_r \\
\delta(q_0, \widetilde{\#}) &= r_1 p_r r_1 & \delta(p_r, \widetilde{\$}) &= r_0 \\
\delta(p_\ell, a) &= p_\ell p_\ell & \delta(r_i, x) &= r_i \text{ for } x \in \Sigma, i \in \{0, 1\} \\
\delta(p_\ell, x) &= r_0 \text{ for } x \in \{\$, \widetilde{\$}\}
\end{aligned}$$

The δ-values that are not explicitly defined can be defined arbitrarily. Finally, let $\rho(r_0) = 0$ and $\rho(r_1) = 1$; all other ρ-values can be defined arbitrarily. We claim that

$$\text{leaf}(A, e(d_1, \ldots, d_k)) = S(w_1, \ldots, w_k).$$

First note that $\widehat{\delta}(p_r, a^e\$) = r_0^{2^e} p_r$ and $\widehat{\delta}(p_r, a^e\widetilde{\$}) = r_0^{2^e}$. Since $\delta(r_0, x) = r_0$ for all input symbols x, we have $\widehat{\delta}(p_r, e(d)) = r_0^d$ for every number d and therefore:

$$\begin{aligned}
\widehat{\delta}(q_0, \#e(d)) &= \widehat{\delta}(q_0, e(d))\, \widehat{\delta}(p_r, e(d))\, \widehat{\delta}(q_0, e(d)) = q_0 r_0^d q_0 \\
\widehat{\delta}(q_0, \widetilde{\#}e(d)) &= \widehat{\delta}(r_1, e(d))\, \widehat{\delta}(p_r, e(d))\, \widehat{\delta}(r_1, e(d)) = r_1 r_0^d r_1
\end{aligned}$$

Hence, the FLA A realizes the recurrence (3) when reading the input string $e(d_1, \ldots, d_k)$. □

A precise characterization of the class $\bigcup\{\mathsf{LEAF}^{\mathsf{FA}}(K) \mid K \text{ is context-free}\}$ remains open.

4.2. Compressed embedding problem

We say that a string $u = a_1 a_2 \cdots a_n$ ($a_i \in \Sigma$) is a *subsequence* of a string $v \in \Sigma^*$ (or u *embeds* into v, briefly $u \hookrightarrow v$), if $v \in \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots a_{n-1} \Sigma^* a_n \Sigma^*$. The *compressed embedding problem*, briefly COMPRESSED-EMBEDDING, is defined as follows:

INPUT: Two SLPs $\mathbb{A}$ and $\mathbb{B}$ over the alphabet $\{0, 1\}$
QUESTION: $\text{val}(\mathbb{A}) \hookrightarrow \text{val}(\mathbb{B})$?

In [25], it was shown that COMPRESSED-EMBEDDING is hard for $\mathsf{P}_{\parallel}^{\mathsf{NP}}$, which is the class of all problems that can be solved on a deterministic Turing-machine with access to an NP-oracle, where all queries are sent in parallel

to the oracle (non-adaptive oracle access). A simplified proof for this result can be found in [28].

Here we will strengthen the lower bound of $\mathsf{P}_{\parallel}^{\mathsf{NP}}$ to PP. The class PP (probabilistic polynomial time) consists of all languages L, for which there exists a polynomial time NTM M such that $w \in L$ if and only if on input w the number of accepting computations is larger than the number of rejecting computations. The latter means that the acceptance probability has to be larger than $1/2$, when M is viewed as a probabilistic Turing machine. It is known that $\mathsf{P}_{\parallel}^{\mathsf{NP}} \subseteq \mathsf{PP}$ [4]. Moreover, Toda's seminal theorem [35] states that P^{PP} contains the polynomial time hierarchy. To the knowledge of the author, the best known upper bound for COMPRESSED-EMBEDDING is PSPACE.

Clearly, for all strings u, v, w we have $uw \hookrightarrow vw$ if and only if $u \hookrightarrow v$. We will need a simple generalization of this fact. We say that a string $u \in \Sigma^+$ *just fits into* a string $v = av' \in \Sigma^+$ ($a \in \Sigma$) if $u \hookrightarrow v$ and $u \not\hookrightarrow v'$. The proof of the following lemma is straightforward.

Lemma 12. *Let u, v, w, x be strings such that w just fits into x. Then $uw \hookrightarrow vx$ if and only if $u \hookrightarrow v$.*

Theorem 13. *COMPRESSED-EMBEDDING is PP-hard under logspace reductions.*

Proof. It is easy to see that there exists a fixed fully balanced polynomial time NTM M such that the following problem is PP-complete:

INPUT: An input w for M and a binary coded number m.
QUESTION: Is the number of accepting computations of M on input w at least m?

For instance, we can use the fact that checking whether a given CNF-formula has at least m (which is part of the input) satisfying assignments is a classical PP-complete problem.

Let us fix an input w for M and a binary coded number m. By Theorem 7, we can compute in logspace two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that $\mathrm{leaf}(M, w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$ and $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$. Let $n = |\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})| \geq 1$; 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:

$$\phi_1(0) = 0^{n+1} \qquad \phi_1(1) = (10)^n \tag{6}$$
$$\phi_2(0) = 0(10)^n \qquad \phi_2(1) = (10)^{n+1} \tag{7}$$

It is straightforward to compute in logspace SLPs $\mathbb{C}$ and $\mathbb{D}$ such that $\mathrm{val}(\mathbb{C}) = \phi_1(\mathrm{val}(\mathbb{A}))(10)^m$ and $\mathrm{val}(\mathbb{D}) = \phi_2(\mathrm{val}(\mathbb{B}))$. We claim that $\mathrm{val}(\mathbb{C}) \hookrightarrow \mathrm{val}(\mathbb{D})$ if and only if the number of accepting computations of M on input w is at least m. Note that the latter fact means that $\mathrm{leaf}(M, w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$ contains at least m 1's, i.e., that $\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})$ contains at least m occurrences of the symbol $(1, 1)$. Hence, we have to show that $\phi_1(\mathrm{val}(\mathbb{A}))(10)^m \hookrightarrow \phi_2(\mathrm{val}(\mathbb{B}))$ if and only if $\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})$ contains at least m occurrences of $(1, 1)$.

By induction over $i \geq 1$, we will prove that the following two statements are equivalent for every $x \in \{0, \ldots, n\}$:

(A) $\phi_1(\mathrm{val}(\mathbb{A})[: i])(10)^x \hookrightarrow \phi_2(\mathrm{val}(\mathbb{B})[: i])$

(B) $\mathrm{val}(\mathbb{A})[: i] \otimes \mathrm{val}(\mathbb{B})[: i]$ has at least x many occurrences of $(1, 1)$.

An inspection of (6) and (7) shows that this is true for $i = 1$: (A) holds if and only if $x = 0$ or ($x = 1$ and $\mathrm{val}(\mathbb{A})[1] = \mathrm{val}(\mathbb{B})[1] = 1$) if and only if (B) holds. Now, assume that (A) and (B) are equivalent for some $i \geq 1$. We will prove the equivalence for $i + 1$. We can restrict to the case $x > 0$ since for $x = 0$, (A) and (B) are both true (for (A) note that $\phi_1(a) \hookrightarrow \phi_2(b)$ for all $a, b \in \{0, 1\}$). Note that for every $k \geq 1$ and $x \leq n$, $\phi_2(\mathrm{val}(\mathbb{B})[: k])$ has a suffix of the form $(10)^x$. Hence, we get

$$\phi_1(\mathrm{val}(\mathbb{A})[: i + 1])(10)^x \hookrightarrow \phi_2(\mathrm{val}(\mathbb{B})[: i + 1])$$

if and only if

$$\phi_1(\mathrm{val}(\mathbb{A})[: i + 1]) \hookrightarrow \phi_2(\mathrm{val}(\mathbb{B})[: i + 1]) \setminus (10)^x. \tag{8}$$

We make a case distinction on the symbols $\mathrm{val}(\mathbb{A})[i + 1]$ and $\mathrm{val}(\mathbb{B})[i + 1]$.

Case 1. $\mathrm{val}(\mathbb{A})[i + 1] \neq 1$ or $\mathrm{val}(\mathbb{B})[i + 1] \neq 1$. We first show that (8) is equivalent to

$$\phi_1(\mathrm{val}(\mathbb{A})[: i]) \hookrightarrow \phi_2(\mathrm{val}(\mathbb{B})[: i]) \setminus (10)^x. \tag{9}$$

Case 1.1. $\mathrm{val}(\mathbb{A})[i+1] = \mathrm{val}(\mathbb{B})[i+1] = 0$. Then (8) is equivalent to

$$\begin{aligned} \phi_1(\mathrm{val}(\mathbb{A})[:i])0^{n+1} &\hookrightarrow \phi_2(\mathrm{val}(\mathbb{B})[:i])0(10)^{n-x} \\ &= (\phi_2(\mathrm{val}(\mathbb{B})[:i]) \setminus (10)^x)(10)^x 0(10)^{n-x}. \end{aligned} \tag{10}$$

Since 0^{n+1} just fits into $0(10)^{x-1}0(10)^{n-x}$, Lemma 12 implies that (10) is equivalent to

$$\phi_1(\mathrm{val}(\mathbb{A})[:i]) \hookrightarrow (\phi_2(\mathrm{val}(\mathbb{B})[:i]) \setminus (10)^x)1. \tag{11}$$

Finally, since $\phi_1(\mathrm{val}(\mathbb{A})[:i])$ ends with 0, (11) is indeed equivalent to (9).

Case 1.2. $\mathrm{val}(\mathbb{A})[i+1] = 0$ and $\mathrm{val}(\mathbb{B})[i+1] = 1$. Then (8) is equivalent to

$$\begin{aligned} \phi_1(\mathrm{val}(\mathbb{A})[:i])0^{n+1} &\hookrightarrow \phi_2(\mathrm{val}(\mathbb{B})[:i])(10)^{n+1-x} \\ &= (\phi_2(\mathrm{val}(\mathbb{B})[:i]) \setminus (10)^x)(10)^x (10)^{n+1-x}. \end{aligned} \tag{12}$$

The same arguments as in Case 1.1 yield equivalence of (12) and (9).

Case 1.3. $\mathrm{val}(\mathbb{A})[i+1] = 1$ and $\mathrm{val}(\mathbb{B})[i+1] = 0$. Then (8) is equivalent to

$$\begin{aligned} \phi_1(\mathrm{val}(\mathbb{A})[:i])(10)^{n} &\hookrightarrow \phi_2(\mathrm{val}(\mathbb{B})[:i])0(10)^{n-x} \\ &= (\phi_2(\mathrm{val}(\mathbb{B})[:i]) \setminus (10)^x)(10)^x 0(10)^{n-x}. \end{aligned} \tag{13}$$

The string $(10)^n$ just fits into $(10)^x 0(10)^{n-x}$. Hence, by Lemma 12, (13) is indeed equivalent to (9).

In all three subcases, we have shown that (8) is equivalent to (9).

By induction, (9) is equivalent to the fact that $\mathrm{val}(\mathbb{A})[:i] \otimes \mathrm{val}(\mathbb{B})[:i]$ contains at least x occurrences of the symbol $(1,1)$. Since $(\mathrm{val}(\mathbb{A})[i+1], \mathrm{val}(\mathbb{B})[i+1]) \neq (1,1)$, this is equivalent to the fact that $\mathrm{val}(\mathbb{A})[:i+1] \otimes \mathrm{val}(\mathbb{B})[:i+1]$ contains at least x occurrences of $(1,1)$, and we are done.

Case 2. $\mathrm{val}(\mathbb{A})[i+1] = \mathrm{val}(\mathbb{B})[i+1] = 1$. Then (8) is equivalent to

$$\begin{aligned} \phi_1(\mathrm{val}(\mathbb{A})[:i])(10)^{n} &\hookrightarrow \phi_2(\mathrm{val}(\mathbb{B})[:i])(10)^{n+1-x} \\ &= (\phi_2(\mathrm{val}(\mathbb{B})[:i]) \setminus (10)^x)(10)^x (10)^{n+1-x}. \end{aligned}$$

This is equivalent to

$$\phi_1(\mathrm{val}(\mathbb{A})[:i]) \hookrightarrow (\phi_2(\mathrm{val}(\mathbb{B})[:i]) \setminus (10)^x)10. \tag{14}$$

Since $x > 0$, (14) is equivalent to

$$\phi_1(\mathrm{val}(\mathbb{A})[:i]) \hookrightarrow \phi_2(\mathrm{val}(\mathbb{B})[:i]) \setminus (10)^{x-1}.$$

By induction, this is equivalent to the fact that $\mathrm{val}(\mathbb{A})[:i] \otimes \mathrm{val}(\mathbb{B})[:i]$ contains at least $x-1$ occurrences of $(1,1)$. Since $(\mathrm{val}(\mathbb{A})[i+1], \mathrm{val}(\mathbb{B})[i+1]) = (1,1)$, this is equivalent to the fact that $\mathrm{val}(\mathbb{A})[:i+1] \otimes \mathrm{val}(\mathbb{B})[:i+1]$ contains at least x occurrences of $(1,1)$. This concludes the proof. □

Together with Toda's theorem ($\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$), Theorem 13 implies that the problem COMPRESSED-EMBEDDING is not contained in the polynomial time hierarchy unless the latter collapses.

5. Compressed membership in XML languages

In this final section, we consider a subclass of the visibly pushdown languages, which is motivated in connection with XML. Let B be a finite set of opening brackets and let $\overline{B}$ be the set of corresponding closing brackets. An *XML-grammar* [5] is a tuple $G = (B, (R_b)_{b \in B}, a)$ where $a \in B$ (the axiom) and R_b is a regular language over the alphabet $\{X_c \mid c \in B\}$. We identify G with the context-free grammar, where (i) $\{X_b \mid b \in B\}$ is the set of variables, (ii) $B \cup \overline{B}$ is the set of terminals, (iii) X_a is the start variable, and (iv) the (infinite) set of productions is $\{X_b \to b\, w\, \overline{b} \mid b \in B, w \in R_b\}$. Clearly, since R_b is regular, this set is equivalent to a finite set of productions. XML-grammars capture the syntactic features of XML document type definitions (DTDs), see [5] for more details. For every XML-grammar G, the language $L(G)$ is a visibly pushdown language [1]. The main result of this section is:

Theorem 14. *For every XML-grammar G,* $\mathrm{CMP}(L(G))$ *belongs to* coNP. *Moreover, there is an XML-grammar G such that* $\mathrm{CMP}(L(G))$ *is* coNP*-complete w.r.t. logspace reductions.*

For the proof of the upper bound in Theorem 14 we need a few definitions. Let us fix an XML-grammar $G = (B, (R_b)_{b \in B}, a)$ until further notice. The set $D_B \subseteq (B \cup \overline{B})^+$ of all *Dyck primes* over B is the set of all well-formed strings over $B \cup \overline{B}$ that do not have a non-empty proper prefix, which is well-formed as well. Formally, D_B is the smallest set such that $w_1, \ldots, w_n \in D_B$ $(n \geq 0)$ and $b \in B$ imply $b w_1 \cdots w_n \overline{b} \in D_B$. For $b \in B$ let $D_b = D_B \cap b(B \cup \overline{B})^* \overline{b}$. The set of all *Dyck words* over $B \cup \overline{B}$ is D_B^*. Note that $L(G) \subseteq D_a$.

Assume that $w \in D_B^*$ and let $1 \leq i \leq |w|$ be a position in w such that $w[i] \in B$, i.e., the i-th symbol in w is an opening bracket. Since $w \in D_B^*$, there exists a unique position $\gamma(w,i) > i$ such that $w[i,\gamma(w,i)] \in D_B$. The string $w[i+1:\gamma(w,i)-1]$ belongs to D_B^*. Since D_B is a code, there exists a unique factorization $w[i+1:\gamma(w,i)-1] = w_1 w_2 \cdots w_n$ such that $n \geq 0$ and $w_1, w_2, \ldots, w_n \in D_B$. Moreover, for every $1 \leq i \leq n$ let b_i be the unique opening bracket such that $w_i \in D_{b_i}$. Finally, define $\mathrm{surface}(w,i) = X_{b_1} X_{b_2} \cdots X_{b_n}$. We choose the term "surface" here, because this definition is motivated by the surface of $b \in B$ from [5].

Lemma 15. *Let $w \in (B \cup \overline{B})^*$. Then $w \in L(G)$ if and only if (i) $w \in D_a$ and (ii) $\mathrm{surface}(w,j) \in R_b$ for every position $1 \leq j \leq |w|$ such that $w[j] = b \in B$.*

Proof. The only-if direction is easy to see. Let us prove the if-direction. For $b \in B$ let L_b be the set of all $w \in D_b$ that can be generated from the variable X_b of G. It suffices to prove that for every $b \in B$ and every $w \in D_b$:

$$\forall 1 \leq j \leq |w| \, (w[j] \in B \to \mathrm{surface}(w,j) \in R_{w[j]}) \quad \Longrightarrow \quad w \in L_b. \tag{15}$$

We prove (15) by induction over the length of w (simultaneously for all $b \in B$). Hence, let $w \in D_b$ such that $\mathrm{surface}(w,j) \in R_{w[j]}$ for every position $1 \leq j \leq |w|$ with $w[j] \in B$. Moreover, assume that (15) is true for all strings strictly shorter than w. Since $w \in D_b$, we can factorize w uniquely as $w = b w_1 \cdots w_n \overline{b}$ such that $n \geq 0$ and $w_1, \ldots, w_n \in D_B$. Assume that $w_i \in D_{b_i}$ for $1 \leq i \leq n$. Since $|w_i| < |w|$ and w_i satisfies the precondition in (15) (with b replaced by b_i), we obtain $w_i \in L_{b_i}$ for all $1 \leq i \leq n$ by induction. Together with $X_{b_1} \cdots X_{b_n} = \mathrm{surface}(w,1) \in R_{w[1]} = R_b$ we obtain $w = b w_1 \cdots w_n \overline{b} \in L_b$. □

The next lemma was shown in [26, Lemma 5.6]:

Lemma 16. $\mathrm{CMP}(D_B^*)$ *can be solved in polynomial time. Moreover, for a given SLP $\mathbb{A}$ such that $w := \mathrm{val}(\mathbb{A}) \in D_B^*$ and a given (binary coded) position $1 \leq i \leq |w|$ with $w[i] \in B$ one can compute the position $\gamma(w,i)$ in polynomial time.*

Lemma 16 and the fact that $w \in D_B \iff (w \in D_B^* \text{ and } \gamma(w,1) = |w|)$ imply:

Proposition 17. $\mathrm{CMP}(D_B)$ *can be solved in polynomial time.*

For the proof of Theorem 14 we need one more technical lemma:

Lemma 18. *For a given SLP $\mathbb{A}$ such that $w := \mathrm{val}(\mathbb{A}) \in D_B^*$ and a given (binary coded) position $1 \le i \le |w|$ with $w[i] \in B$ one can compute an SLP for the string surface(w, i) in polynomial time.*

Proof. By Lemma 16 we can compute the position $\gamma(w, i)$ in polynomial time. Next, we can compute in polynomial time an SLP $\mathbb{B}$ with $\mathrm{val}(\mathbb{B}) = w[i+1 : \gamma(w,i) - 1] \in D_B^*$.

Consider the following string rewriting system S over the alphabet $B \cup \overline{B}$:

$$S = \{b\, c\, \overline{c} \to b \mid b, c \in B\} \cup \{c\, \overline{c}\, \overline{b} \to \overline{b} \mid b, c \in B\}$$

The system S is terminating and confluent. The latter can easily be checked by considering critical pairs resulting from overlapping left-hand sides of S, see e.g. [7]. Therefore every string $v \in (B \cup \overline{B})^*$ has a unique normal form $\mathrm{NF}_S(v)$ w.r.t. S, i.e. $v \to_S^* \mathrm{NF}_S(v)$ and $\mathrm{NF}_S(v)$ is irreducible w.r.t. S. The main property of S is the following: Let $w_i \in D_{a_i}$ ($a_i \in B$) for $1 \le i \le n$. Then

$$\mathrm{NF}_S(w_1 \cdots w_n) = a_1 \overline{a}_1 a_2 \overline{a}_2 \cdots a_n \overline{a}_n$$

This can easily be shown by induction on the length of the string $w_1 \cdots w_n$. For the set of normal forms of factors of well-formed words, we have:

$$\mathrm{NF}_S(\{w \in (B \cup \overline{B})^* \mid \exists x, y \in (B \cup \overline{B})^* : xwy \in D_B^*\}) = \overline{B}^* \{b\overline{b} \mid b \in B\}^* B^*.$$

From the SLP $\mathbb{B}$ (which, w.l.o.g., is in Chomsky normal form) we compute in polynomial time a composition system $\mathbb{C}$ (see Section 2.2) such that $\mathrm{val}(\mathbb{C}) = \mathrm{NF}_S(\mathrm{val}(\mathbb{B}))$. For this, $\mathbb{C}$ contains for every variable B_i of $\mathbb{B}$ variables C_i, $C_{i,1}$, $C_{i,2}$, and $C_{i,3}$ such that

- $\mathrm{val}(C_i) = \mathrm{NF}_S(\mathrm{val}(B_i))$,
- $\mathrm{val}(C_{i,1}) \in \overline{B}^*$, $\mathrm{val}(C_{i,2}) \in \{b\overline{b} \mid b \in B\}^*$, $\mathrm{val}(C_{i,3}) \in B^*$, and
- $C_i \to C_{i,1} C_{i,2} C_{i,3}$ is a production of $\mathbb{C}$.

We add productions to $\mathbb{C}$ with a bottom-up process. Assume that all productions of $\mathbb{B}$ have the form $B_i \to x \in B \cup \overline{B}$ or $B_i \to B_j B_k$ with $j, k < i$. If $B_i \to b \in B$ is a production of $\mathbb{B}$, then $\mathbb{C}$ contains the productions $C_{i,1} \to \varepsilon$, $C_{i,2} \to \varepsilon$, $C_{i,3} \to b$, and $C_i \to C_{i,1} C_{i,2} C_{i,3}$. For a production $B_i \to \overline{b} \in \overline{B}$ we

add to $\mathbb{C}$ the productions $C_{i,1} \to \overline{b}$, $C_{i,2} \to \varepsilon$, $C_{i,3} \to \varepsilon$, and $C_i \to C_{i,1}C_{i,2}C_{i,3}$. Finally, if $B_i \to B_jB_k$ is a production of $\mathbb{B}$ $(j,k < i)$ then all productions for variables $C_p, C_{p,q}$ $(1 \leq p \leq i-1,\ 1 \leq q \leq 3)$ are already constructed. Since $\mathrm{val}(\mathbb{B})$ is well-formed (and hence, $\mathrm{val}(B_i), \mathrm{val}(B_j), \mathrm{val}(B_k)$ are factors of well-formed words), one of the following four conditions must hold:

$$\mathrm{val}(C_{j,3}) = \mathrm{val}(C_{k,1}) = \varepsilon \tag{16}$$

$$\mathrm{val}(C_{j,3}) = \overline{\mathrm{val}(C_{k,1})} \in B^+ \tag{17}$$

$$\overline{\mathrm{val}(C_{k,1})} \in B^* \text{ is a proper suffix of } \mathrm{val}(C_{j,3}) \in B^+ \tag{18}$$

$$\mathrm{val}(C_{j,3}) \in B^* \text{ is a proper suffix of } \overline{\mathrm{val}(C_{k,1})} \in B^+ \tag{19}$$

Using Plandowski's polynomial time algorithm for testing equality of SLP-represented strings [32], we can determine in polynomial time, which of the four cases holds. Depending on the outcome, we add the following productions to $\mathbb{C}$, where $n_{p,q} = |\mathrm{val}(C_{p,q})|$ for $p \in \{j,k\}$ and $1 \leq q \leq 3$ (these numbers can be computed in polynomial time as well):

- If (16) holds: $C_{i,1} \to C_{j,1}$, $C_{i,2} \to C_{j,2}C_{k,2}$, $C_{i,3} \to C_{k,3}$
- If (17) holds: $C_{i,1} \to C_{j,1}$, $C_{i,2} \to C_{j,2}\, b\, \overline{b}\, C_{k,2}$ with b the first symbol of $\mathrm{val}(C_{j,3})$, $C_{i,3} \to C_{k,3}$
- If (18) holds: $C_{i,1} \to C_{j,1}$, $C_{i,2} \to C_{j,2}$, $C_{i,3} \to C_{j,3}[: n_{j,3} - n_{k,1}]C_{k,3}$
- If (19) holds: $C_{i,1} \to C_{j,1}C_{k,1}[n_{j,3}+1 : n_{k,1}]$, $C_{i,2} \to C_{k,2}$, $C_{i,3} \to C_{k,3}$

Finally, we add the production $C_i \to C_{i,1}C_{i,2}C_{i,3}$ to $\mathbb{C}$. The correctness of this construction follows immediately from the definition of the string rewriting system S. For instance, if (19) holds, then by applying the "cut-operator" $[n_{j,3}+1 : n_{k,1}]$ to the variable $C_{k,1}$, we cut off the part that cancels in the product $\mathrm{val}(C_{j,3})\mathrm{val}(C_{k,1})$ when applying the rewriting system S. Note that by (19) this is exactly the factor $\mathrm{val}(C_{j,3})\mathrm{val}(C_{k,1})[: n_{j,3}]$. In other words, we have $\mathrm{NF}_S(\mathrm{val}(C_{j,3})\mathrm{val}(C_{k,1})) = \mathrm{val}(C_{k,1})[n_{j,3}+1 : n_{k,1}]$. This word only contains closing brackets and it is non-empty. Moreover, since $\mathrm{val}(C_{j,2})$ is a sequence of pairs of an opening bracket followed by the corresponding closing bracket, the word $\mathrm{val}(C_{j,2})\mathrm{val}(C_{k,1})[n_{j,3}+1 : n_{k,1}]$ reduces by S to $\mathrm{val}(C_{k,1})[n_{j,3}+1 : n_{k,1}]$. Finally, we have to merge this word with the word $\mathrm{val}(C_{j,1})$ (which only contains closing brackets as well) in order to obtain $\mathrm{val}(C_{i,1})$. This is the reason for the production $C_{i,1} \to C_{j,1}C_{k,1}[n_{j,3}+1 : n_{k,1}]$.

By Hagenah's algorithm from [16], we can transform $\mathbb{C}$ in polynomial time into an equivalent SLP $\mathbb{D}$. From the SLP $\mathbb{D}$ it is now easy to compute an SLP that generates the string surface(w, i): we just have to replace every occurrence of a terminal $b \in B$ (resp. $\overline{B}$) by X_b (resp. ε). $\square$

Now we are in the position to prove Theorem 14.

Proof of Theorem 14. Let us first prove the coNP upper bound. Let $G = (B, (R_b)_{b\in B}, a)$ be an XML grammar, let $\mathbb{A}$ be an SLP over the terminal alphabet $B \cup \overline{B}$, and let $w = \mathrm{val}(\mathbb{A})$. By Lemma 15 we have to check that:

(a) $w \in D_a = D_B \cap a(B \cup \overline{B})^* \overline{a}$ and

(b) $\mathrm{surface}(w, j) \in R_b$ for every position $1 \le j \le |w|$ with $w[j] = b \in B$.

Condition (a) can be checked in deterministic polynomial time by Proposition 17; condition (b) belongs to coNP by Lemma 18 and Theorem 3.

For the coNP lower bound, let G be the context-free grammar with the only nonterminal X, terminal alphabet $\{0, \overline{0}, 1, \overline{1}, [,]\}$, and the following productions:

$$X \to [\,], \quad X \to [\,0\,\overline{0}\,X\,0\,\overline{0}\,], \quad X \to [\,0\,\overline{0}\,X\,1\,\overline{1}\,], \quad X \to [\,1\,\overline{1}\,X\,1\,\overline{1}\,].$$

Thus, $L(G)$ contains all strings of the form $[u_1[u_2[\cdots u_n[\,]v_n \cdots]v_2]v_1]$, where for all $1 \le i \le n$: $u_i, v_i \in \{0\overline{0}, 1\overline{1}\}$ and ($u_i = 0\overline{0}$ or $v_i = 1\overline{1}$). The language $L(G)$ can easily be generated by an XML-grammar with $B = \{0, 1, [\,\}$.

Let M be a fully balanced polynomial time NTM such that $\mathsf{LEAF}(M, 1^*)$ is coNP-complete (such a machine exists). From a given input w for M we compute in logspace, as described in the proof of Theorem 7, two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that $\mathrm{leaf}(M, w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$ and $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$. Hence, $w \in \mathsf{LEAF}(M, 1^*)$ if and only if $\mathrm{val}(\mathbb{A}) \otimes \overline{\mathrm{val}(\mathbb{B})} \in \{(0,0), (0,1), (1,1)\}^*$. Let $\mathbb{C}$ be an SLP with $\mathrm{val}(\mathbb{C}) = \mathrm{val}(\mathbb{A})\,[\,]\,\overline{\mathrm{val}(\mathbb{B})}$. Finally, let the SLP $\mathbb{D}$ result from $\mathbb{C}$ be applying the following mapping to all terminal symbols:

$$0 \mapsto [\,0\,\overline{0}, \quad 1 \mapsto [\,1\,\overline{1}, \quad \overline{0} \mapsto 0\,\overline{0}\,], \quad \overline{1} \mapsto 1\,\overline{1}\,].$$

Then, $\mathrm{val}(\mathbb{D}) \in L(G)$ if and only if $\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in \{(0,0), (0,1), (1,1)\}^*$ if and only if $w \in \mathsf{LEAF}(M, 1^*)$. $\square$

6. Open problems

The precise complexity of COMPRESSED-EMBEDDING remains open. We conjecture that this problem is PSPACE-complete. Another interesting problem is to obtain a precise characterization of the class $\mathsf{LEAF}^{\mathsf{FA}}(\mathsf{CFL})$. This class is contained in PSPACE and we have shown that it contains PSPACE-complete problems (Theorem 11). Recently, a connection between $\mathsf{LEAF}^{\mathsf{FA}}(\mathsf{CFL})$ and second-order monadic monoidal and groupoidal quantifiers was established [23].

References

[1] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. STOC 2004*, pages 202–211. ACM Press, 2004.

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

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

[4] R. Beigel, L. A. Hemachandra, and G. Wechsung. On the power of probabilistic polynomial time: $\mathrm{P}^{\mathrm{NP[log]}}$ subseteq PP. In *Proc. Structure in Complexity Theory Conference 1989*, pages 225–227, 1989.

[5] J. Berstel and L. Boasson. Formal properties of XML grammars and languages. *Acta Inform.*, 38(9):649–671, 2002.

[6] A. Bertoni, C. Choffrut, and R. Radicioni. Literal shuffle of compressed words. In *Proc. IFIP TCS 2008*, pages 87–100. Springer, 2008.

[7] R. V. Book and F. Otto. *String–Rewriting Systems*. Springer, 1993.

[8] B. Borchert and A. Lozano. Succinct circuit representations and leaf language classes are basically the same concept. *Inform. Process. Lett.*, 59(4):211–215, 1996.

[9] D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoret. Comput. Sci.*, 104(2):263–283, 1992.

[10] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *J. Comput. System Sci.*, 57(2):200–212, 1998.

[11] M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

[12] P. W. Dymond. Input-driven languages are in $\log n$ depth. *Inform. Process. Lett.*, 26(5):247–250, 1988.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–completeness*. Freeman, 1979.

[14] L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. In *Proc. SWAT 1996*, LNCS 1097, pages 392–403. Springer, 1996.

[15] S. Greibach. The hardest context-free language. *SIAM J. Comput.*, 2(4):304–310, 1973.

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

[17] U. Hertrampf. The shapes of trees. In *Proc. COCOON 1997*, LNCS 1276, pages 412–421. Springer, 1997.

[18] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proc. Eighth Annual Structure in Complexity Theory Conference*, pages 200–207. IEEE Computer Society Press, 1993.

[19] U. Hertrampf, H. Vollmer, and K. W. Wagner. On balanced versus unbalanced computation trees. *Math. Systems Theory*, 29(4):411–421, 1996.

[20] M. Holzer and K.-J. Lange. On the complexities of linear LL(1) and LR(1) grammars. In *Proc. FCT 1993*, LNCS 710, pages 299–308. Springer, 1993.

[21] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Inform. and Comput.*, 129(1):21–33, 1996.

[22] H. J. Karloff and W. L. Ruzzo. The iterated mod problem. *Inform. and Comput.*, 80(3):193–204, 1989.

[23] J. Kontinen and H. Vollmer. On second-order monadic monoidal and groupoidal quantifiers. *Logical Methods in Computer Science*, 6(3), 2010. electronic.

[24] Y. Lifshits. Processing compressed texts: A tractability border. In *Proc. CPM 2007*, LNCS 4580, pages 228–240. Springer, 2007.

[25] Y. Lifshits and M. Lohrey. Querying and embedding compressed texts. In *Proc. MFCS 2006*, LNCS 4162, pages 681–692. Springer, 2006.

[26] M. Lohrey. Word problems and membership problems on compressed words. *SIAM J. Comput.*, 35(5):1210 – 1240, 2006.

[27] M. Lohrey. Leaf languages and string compression. In *Proc. FSTTCS 2008*, pages 292–303, 2008.

[28] M. Lohrey. Compressed membership problems for regular expressions and hierarchical automata. to appear in *Internat. J. Found. Comput. Sci.*, 2009.

[29] N. Markey and P. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Inform. Process. Lett.*, 90(1):3–6, 2004.

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

[31] T. Peichl and H. Vollmer. Finite automata with generalized acceptance criteria. *Discrete Math. Theor. Comput. Sci.*, 4(2):179–192 (electronic), 2001.

[32] W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA'94*, LNCS 855, pages 460–470. Springer, 1994.

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

[34] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoret. Comput. Sci.*, 302(1–3):211–222, 2003.

[35] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[36] H. Veith. Succinct representation, leaf languages, and projection reductions. *Inform. and Comput.*, 142(2):207–236, 1998.

[37] N. K. Vereshchagin. Relativizable and nonrelativizable theorems in the polynomial theory of algorithms. *Izv. Ross. Akad. Nauk Ser. Mat.*, 57(2):51–90, 1993.

[38] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.

[39] B. von Braunmühl and R. Verbeek. Input-driven languages are recognized in $\log n$ space. In *Proc. FCT 1983*, LNCS 158, pages 40–51. Springer, 1983.

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