

# Leaf languages and string compression\*

**Markus Lohrey**

Universität Leipzig, Institut für Informatik, Germany  
lohrey@informatik.uni-leipzig.de

**ABSTRACT.** Tight connections between leafs languages and strings compressed via 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, a fixed linear visibly pushdown language with a PSPACE-complete compressed membership problem is obtained. For XML languages, the compressed membership problem is shown to be coNP-complete.

## 1 Introduction

*Leaf languages* were introduced in [7, 25] and became 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 computation tree (i.e. the string obtained by listing all leafs 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  $\text{LEAF}(M, K)$  denote the set of all inputs 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  $\text{LEAF}_a^P(K)$ . 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  $\text{LEAF}_b^P(K)$ , see [13, 16] 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,  $\text{NP} = \text{LEAF}_x^P(0^*1\{0,1\}^*)$  and  $\text{coNP} = \text{LEAF}_x^P(1^*)$  for both  $x = a$  and  $x = b$ . In [14], it was shown that  $\text{PSPACE} = \text{LEAF}_b^P(K)$  for a fixed regular language  $K$ . In [16], logspace leaf language classes  $\text{LEAF}_a^L(K)$  and  $\text{LEAF}_b^L(K)$ , where  $M$  varies over all (resp. all balanced) nondeterministic logspace machines, were investigated. Among other results, a fixed deterministic context-free language  $K$  with  $\text{PSPACE} = \text{LEAF}_a^L(K)$  was presented. In [8], it was shown that in fact a fixed deterministic *one-counter* language  $K$  as well as a fixed *linear* deterministic context-free language [15] suffices in order to obtain PSPACE. Here “linear” means that the pushdown automaton makes only one turn.

In [6, 24], 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 [6] and projection reductions in [24]) for the leaf language class  $\text{LEAF}_b^P(K)$ , if the input string  $x$  is represented by a Boolean circuit. A Boolean circuit  $C(x_1, \dots, x_n)$  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

---

\*This work is supported by the DFG research project ALKODA.

and only if  $C(a_1, \dots, a_n) = 1$ , where  $a_1 \cdots a_n$  is the  $n$ -bit binary representation of  $i$ . In this paper we consider another more practical compressed representation for strings, namely *straight-line programs* (SLPs) [23]. A straight-line program is a context-free grammar  $\mathbb{A}$  that generates exactly one string  $\text{val}(\mathbb{A})$ . 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, can be converted in polynomial time into SLPs and vice versa [23]. This implies that complexity results can be transferred from SLP-encoded input strings to LZ-encoded input strings.

Algorithmic problems for SLP-compressed strings were studied e.g. in [5, 18, 19, 20, 22, 23]. A central problem in this context is the *compressed membership problem* for a language  $K$ : it is asked whether  $\text{val}(\mathbb{A}) \in K$  for a given SLP  $\mathbb{A}$ . In [19] 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  $\text{LEAF}_a^L(K)$  (Prop. 2). As a consequence, the existence of a linear deterministic context-free language with a PSPACE-complete compressed membership problem [19] can be deduced from the above mentioned  $\text{LEAF}_a^L$ -characterization of PSPACE from [8], 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  $\rho : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$  such that for every language  $K$ , the problem of checking  $\rho(\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B})) \in K$  for two given SLPs  $\mathbb{A}, \mathbb{B}$  is complete for the class  $\text{LEAF}_b^P(K)$  (Cor. 4). By combining Cor. 4 with the main result from [14] ( $\text{PSPACE} = \text{LEAF}_b^P(K)$  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  $L$  (Cor. 6). Recently, the convolution of SLP-compressed strings was also studied in [5], 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  $\text{val}(\mathbb{A}_n) \otimes \text{val}(\mathbb{B}_n)$  has size  $\Omega(2^{n/2})$ .

From Cor. 6 we obtain a strengthening of one of the above mentioned results from [8] ( $\text{PSPACE} = \text{LEAF}_a^L(K)$  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 [27] as input-driven languages. In [9] it was shown that every visibly pushdown language can be recognized in  $\text{NC}^1$ ; 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 deterministic one-counter languages with an L-complete membership problem [15]. We show that there exists a linear visibly pushdown language with a PSPACE-complete compressed membership problem (Thm. 7). Together with Prop. 2, it follows that  $\text{PSPACE} = \text{LEAF}_a^L(K)$  for a linear visibly pushdown language  $K$  (Cor. 8).

In [21], 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  $\text{LEAF}^{\text{FA}}(K)$ . It was shown that  $\text{CFL} \subsetneq \text{LEAF}^{\text{FA}}(\text{CFL}) \subseteq \text{DSPACE}(n^2) \cap \text{DTIME}(2^{O(n)})$ , 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  $\text{LEAF}^{\text{FA}}(K)$  contains a PSPACE-complete language (Thm. 9).

Finally, in Sec. 5 we consider *XML-languages* [4], 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 [4]. 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.

Proofs that are omitted due to space restriction will appear in a long version.

## 2 Preliminaries

Let  $\Gamma$  be a finite alphabet. The *empty word* is denoted by  $\varepsilon$ . Let  $s = a_1 \cdots a_n \in \Gamma^*$  be a word over  $\Gamma$  ( $n \geq 0$ ,  $a_1, \dots, a_n \in \Gamma$ ). 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  $s[i, j] = \varepsilon$ . We denote with  $\bar{\Gamma} = \{\bar{a} \mid a \in \Gamma\}$  a disjoint copy of  $\Gamma$ . For  $\bar{a} \in \bar{\Gamma}$  let  $\bar{\bar{a}} = a$ . For  $w = a_1 \cdots a_n \in (\Gamma \cup \bar{\Gamma})^*$  let  $\bar{w} = \bar{a}_n \cdots \bar{a}_1$ . 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  $(u_1, \dots, u_n)$  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  $(w_1, \dots, w_k, t)$  of binary coded natural numbers. It is a positive instance if there are  $x_1, \dots, x_k \in \{0, 1\}$  such that  $t = x_1 w_1 + \cdots + x_k w_k$ . Subsetsum is a classical NP-complete problem. The *superdecreasing subsetsum* problem is the restriction of subsetsum to instances  $(w_1, \dots, w_k, t)$ , where  $(w_1, \dots, w_k)$  is superdecreasing. In [17] it was shown that superdecreasing subsetsum is P-complete ([17] deals with the *superincreasing* subsetsum problem; but the results from [17] can be easily transferred to superdecreasing subsetsum). In fact, something more general is shown in [17]: Let  $C(x_1, \dots, x_m)$  be a Boolean circuit with variable input gates  $x_1, \dots, x_m$  (and some additional input gates that are set to fixed Boolean values). Then from  $C(x_1, \dots, x_m)$  an instance  $(t(x_1, \dots, x_m), w_1, \dots, w_k)$  of superdecreasing subsetsum is constructed. Here,  $t(x_1, \dots, x_m) = t_0 + x_1 t_1 + \cdots + x_m t_m$  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  $(t_1, \dots, t_m)$  is superdecreasing.
- For all  $a_1, \dots, a_m \in \{0, 1\}$ :  $C(a_1, \dots, a_m)$  evaluates to true if and only if  $\exists b_1, \dots, b_k \in \{0, 1\} : t_0 + a_1 t_1 + \cdots + a_m t_m = b_1 w_1 + \cdots + b_k w_k$ .
- $t_0 + t_1 + \cdots + t_m \leq w_1 + \cdots + w_k$

We encode a superdecreasing sequence  $(w_1, \dots, w_k)$  by the string  $S(w_1, \dots, w_k) \in \{0, 1\}^*$  of

length  $w_1 + \dots + w_k + 1$  such that for all  $0 \leq p \leq w_1 + \dots + w_k$ :

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

Since  $(w_1, \dots, w_k)$  is superdecreasing, the number of 1's in  $S(w_1, \dots, w_k)$  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  $T = \{u \in T' \mid u \preceq v\}$ .

## 2.1 Leaf languages

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 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  $\text{leaf}(M, w)$  is the string  $\alpha(q(v_1)) \cdots \alpha(q(v_k))$ , where  $v_1, \dots, v_k$  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 *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  $\text{LEAF}(M, K) = \{w \in \Sigma^* \mid \text{leaf}(M, w) \in K\}$  and the following four complexity classes:

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

The first two (resp. last two) classes are closed under polynomial time (resp. logspace) reductions. More details on leaf languages can be found in [7, 13, 14, 16].

## 2.2 Straight-line programs

Following [23], a *straight-line program (SLP)* over the terminal alphabet  $\Gamma$  is a context-free grammar  $\mathbb{A} = (V, \Gamma, S, P)$  ( $V$  is the set of variables,  $\Gamma$  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  $\text{val}(\mathbb{A})$ . 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  $B, C \in V$ .

As an example, consider the SLP  $\mathbb{A}$  (in Chomsky normal form) that consists of the productions  $A_1 \rightarrow b$ ,  $A_2 \rightarrow a$ , and  $A_i \rightarrow A_{i-1}A_{i-2}$  for  $3 \leq i \leq 7$ . The start variable is  $A_7$ . Then  $\text{val}(\mathbb{A}) = \text{abaababaabaab}$ , which is the 7-th Fibonacci word. We have  $|\mathbb{A}| = 12$ .

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.

Let us state some simple algorithmic problems that can be easily solved in polynomial time (but not in deterministic logspace under reasonable complexity theoretic assumptions: problem (a) is #L-complete, problems (b) and (c) are complete for functional P [18]):

- (a) Given an SLP  $\mathbb{A}$ , calculate  $|\text{val}(\mathbb{A})|$ .
- (b) Given an SLP  $\mathbb{A}$  and a number  $i \in \{1, \dots, |\text{val}(\mathbb{A})|\}$ , calculate  $\text{val}(\mathbb{A})[i]$ .
- (c) Given an SLP  $\mathbb{A}$  and two positions  $1 \leq i \leq j \leq |\text{val}(\mathbb{A})|$ , calculate an SLP for the string  $\text{val}(\mathbb{A})[i, j]$ .

In [22], Plandowski presented a polynomial time algorithm for testing whether  $\text{val}(\mathbb{A}) = \text{val}(\mathbb{B})$  for two given SLPs  $\mathbb{A}$  and  $\mathbb{B}$ . For a language  $L \subseteq \Sigma^*$ , we denote with  $\text{CMP}(L)$  (*compressed membership problem* for  $L$ ) the following computational problem:

INPUT: An SLP  $\mathbb{A}$  over the terminal alphabet  $\Sigma$

QUESTION:  $\text{val}(\mathbb{A}) \in L$ ?

The following result was shown in [3, 16, 20]:

**THEOREM 1.** *For every regular language  $L$ ,  $\text{CMP}(L)$  can be decided in polynomial time. Moreover, there exists a fixed regular language  $L$  such that  $\text{CMP}(L)$  is P-complete.*

In [18], we constructed in logspace from a given superdecreasing sequence  $(w_1, \dots, w_k)$  an SLP  $\mathbb{A}$  over  $\{0, 1\}$  such that  $\text{val}(\mathbb{A}) = S(w_1, \dots, w_k)$ , where  $S(w_1, \dots, w_k)$  is the string-encoding from (1). This construction was used in order to prove P-hardness of the problem (b) above. Let us briefly repeat the construction. For  $1 \leq i \leq k$  let

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

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

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

Then  $S(w_1, \dots, w_k) = S_1$ . Note that the SLP that implements the recursion (3) can be constructed in logspace from the binary encoded sequence  $(w_1, \dots, w_k)$  (in [18] only the existence of an NC-construction is claimed). The only nontrivial step is the calculation of all suffix sums  $w_{i+1} + \dots + w_k$  for  $1 \leq i \leq k - 1$  in (2), see e.g. [26].



### 3 Straight-line programs versus leaf languages

In [6, 24], it was shown that the membership problem for a language  $K \subseteq \{0,1\}^*$  is complete (w.r.t. polynomial time reductions in [6] and projection reductions in [24]) for the leaf language class  $\text{LEAF}_b^P(K)$ , if the input string is represented by a Boolean circuit. For SLP-compressed strings, we obtain a similar result:

**PROPOSITION 2.** *For every language  $K \subseteq \{0,1\}^*$ , the problem  $\text{CMP}(K)$  is complete w.r.t. logspace reductions for the class  $\text{LEAF}_a^L(K)$ .*

The proposition can be easily shown by translating configuration graphs of logspace machines into SLPs and vice versa. We now prove a more subtle relationship between SLP-compressed strings and polynomial time leaf languages. Let  $\rho : (\{0,1\} \times \{0,1\})^* \rightarrow \{0,1\}^*$  be the morphism defined by

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

**THEOREM 3.** *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  $|\text{val}(\mathbb{A})| = |\text{val}(\mathbb{B})|$  and  $\text{leaf}(M, w) = \rho(\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B}))$ .*

**PROOF.** Let  $w$  be an input for  $M$ . Our construction consists of five steps:

*Step 1.* By simulating  $M$  e.g. along the right-most computation path, we can compute in polynomial time the number  $m$  of branching nodes along every maximal path in the computation tree  $T(w)$ . Thus, maximal paths in  $T(w)$  can be represented by strings from  $\{0,1\}^m$ .

*Step 2.* Using the classical Cook-Levin construction, we compute in logspace a Boolean circuit  $C_w(x_1, \dots, x_m)$  from  $w$  such that for all  $a_1, \dots, a_m \in \{0,1\}$ :  $C_w(a_1, \dots, a_m)$  evaluates to true if and only if the machine  $M$  accepts on the computation path that is specified by the bit string  $a_1 \cdots a_m$ . The circuit  $C_w(x_1, \dots, x_m)$  has input gates  $x_1, \dots, x_m$  together with some additional input gates that carry fixed input bits.

*Step 3.* The construction from [17] (see Sec. 2) allows us to compute from  $C_w(x_1, \dots, x_m)$  in logspace a superdecreasing subsetsum instance  $(t(x_1, \dots, x_m), w_1, \dots, w_k)$  with  $w_1, \dots, w_k \in \mathbb{N}$  and  $t(x_1, \dots, x_m) = t_0 + x_1 t_1 + \cdots + x_m t_m$  such that

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

*Step 4.* By [18] (see the end of Sec. 2.2), we can construct in logspace from the two superdecreasing sequences  $(t_1, \dots, t_m), (w_1, \dots, w_k)$  SLPs  $\mathbb{A}'$  and  $\mathbb{B}$  over  $\{0,1\}$  such that  $\text{val}(\mathbb{A}') = S(t_1, \dots, t_m)$  and  $\text{val}(\mathbb{B}) = S(w_1, \dots, w_k)$  (see (1)). Note that  $|\text{val}(\mathbb{A}')| = t_1 + \cdots + t_m + 1 \leq w_1 + \cdots + w_k + 1 = |\text{val}(\mathbb{B})|$ .

*Step 5.* Now, we compute in polynomial time the right-most path of the computation tree  $T(w)$ . Assume that this path is represented by the bit string  $r = r_1 \cdots r_m \in \{0,1\}^m$ . Let  $p = r_1 t_1 + \cdots + r_m t_m$ . Thus, if  $r$  is the lexicographically  $n$ -th string in  $\{0,1\}^m$ , then  $p + 1$  is the position of the  $n$ -th 1 in  $\text{val}(\mathbb{A}')$ . From the SLP  $\mathbb{A}'$  we can finally compute in polynomial

time an SLP  $\mathbb{A}$  with  $\text{val}(\mathbb{A}) = 0^{t_0} S(t_1, \dots, t_m)[1, p+1] 0^{w_1+\dots+w_k-t_0-p}$ . Then  $|\text{val}(\mathbb{A})| = |\text{val}(\mathbb{B})|$  and for all positions  $q \in \{0, \dots, |\text{val}(\mathbb{A})| - 1\}$ :

- $\text{val}(\mathbb{A})[q+1] = 1$  if and only if  $\exists a_1, \dots, a_m \in \{0, 1\} : q = t_0 + a_1 t_1 + \dots + a_m t_m$
- $\text{val}(\mathbb{B})[q+1] = 1$  if and only if  $\exists b_1, \dots, b_k \in \{0, 1\} : q = b_1 w_1 + \dots + b_k w_k$ .

Due to the definition of the projection  $\rho$  in (4), we finally have

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

where  $\alpha(x) \in \{0, 1\}$  and  $\alpha(x_1 \dots x_m) = 1$  if and only if there exist  $b_1, \dots, b_k \in \{0, 1\}$  such that  $t_0 + x_1 t_1 + \dots + x_m t_m = b_1 w_1 + \dots + b_k w_k$ . Hence,  $\alpha(x_1 \dots x_m) = 1$  if and only if  $M$  accepts on the computation path specified by  $x_1 \dots x_m \preceq r$ . Thus,  $\text{leaf}(M, w) = \rho(\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B}))$ . ■

Thm. 3 implies the hardness part in the following corollary. The proof of the upper bound is not difficult and left to the reader.

**COROLLARY 4.** *For every language  $K \subseteq \{0, 1\}^*$ , the following problem is complete for the class  $\text{LEAF}_b^P(K)$  w.r.t. polynomial time reductions:*

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

In order to get completeness results w.r.t. logspace reductions in the next section, we need a variant of Thm. 3. 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 5.** *Let  $M$  be a fully balanced polynomial time NTM such that for some polynomial  $p(n)$ , every maximal path in a 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  $\text{leaf}(M, w) = \rho(\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B}))$  and  $|\text{val}(\mathbb{A})| = |\text{val}(\mathbb{B})|$ .*

**PROOF.** Only step 1 and 5 in the proof of Thm. 3 cannot be done in logspace, unless  $L = P$ . Under the additional assumptions of Thm. 5, 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 just have to compute in logspace an SLP  $\mathbb{A}$  with  $\text{val}(\mathbb{A}) = 0^{t_0} S(t_1, \dots, t_m) 0^{w_1+\dots+w_k-(t_0+\dots+t_m)}$ . ■

## 4 Applications

**COROLLARY 6.** *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:  $\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B}) \in L$ ?*

**PROOF.** Membership in PSPACE is obvious. Let us prove the lower bound. By [14], there exists a regular language  $K \subseteq \{0, 1\}^*$  and a balanced polynomial time NTM  $M$  such that the language  $\text{LEAF}(M, K)$  is PSPACE-complete. Using the padding technique from [16, Prop. 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  $\rho$  from (4) is a fixed morphism. Let  $w$

be an input for  $M$ . By Thm. 5, we can construct in logspace two SLPs  $\mathbb{A}$  and  $\mathbb{B}$  such that  $\rho(\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B})) = \text{leaf}(M, w)$ . Hence, the corollary follows from  $w \in \text{LEAF}(M, K) \iff \text{leaf}(M, w) = \rho(\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B})) \in K \iff \text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B}) \in L$ .  $\blacksquare$

From Thm. 5 it follows that that even the set of all SLP-pairs  $\langle \mathbb{A}, \mathbb{B} \rangle$  with  $\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B}) \in L$  and  $|\text{val}(\mathbb{A})| = |\text{val}(\mathbb{B})|$  (or  $|\text{val}(\mathbb{A})| \leq |\text{val}(\mathbb{B})|$ ) is PSPACE-complete w.r.t. logspace reductions. We need this detail in the proof of the next theorem.

In [19] we constructed a linear deterministic context-free language with a PSPACE-complete compressed membership problem. As noted in the introduction, this result follows also from  $\text{PSPACE} = \text{LEAF}_a^L(K)$  for a linear deterministic context-free language  $K$  [8] together with Prop. 2. We now sharpen this result to linear visibly pushdown languages.

Let  $\Sigma_c$  and  $\Sigma_r$  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  $(\Sigma_c, \Sigma_r)$  is a tuple  $V = (Q, q_0, \Gamma, \perp, \Delta, F)$ , 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,  $\Gamma$  is the finite set of stack symbols,  $\perp \in \Gamma$  is the initial stack symbol, and

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

is the set of transitions. 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 configuration of  $V$  is a triple from  $Q \times \Sigma^* \times \Gamma^*$ . 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

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

The language  $L(V)$  is defined as  $L(V) = \{w \in \Sigma^* \mid \exists f \in F, u \in \Gamma^* : (q_0, w, \perp) \Rightarrow_V^* (f, \varepsilon, u)\}$ . 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  $(p, a, \gamma, q) \in \Delta$ .

For every VPA  $V$  there exists a deterministic VPA  $V'$  with  $L(V) = L(V')$  [1]. A *1-turn VPA* is a VPA  $V$  with  $L(V) \subseteq \Sigma_c^* \Sigma_r^*$ . In this case  $L(V)$  is called a *linear visibly pushdown language*.

By a classical result from [11], 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  $\text{NC}^1$  [9] 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 (Thm. 1):

**THEOREM 7.** *There exists a linear visibly pushdown language  $K$  such that  $\text{CMP}(K)$  is PSPACE-complete w.r.t. logspace reductions.*

**PROOF.** Membership in PSPACE holds even for an arbitrary context-free language  $K$  [23]. For the lower bound, we reduce the problem from Cor. 6 to  $\text{CMP}(K)$  for some linear visibly pushdown language  $K$ . Let  $L \subseteq (\{0, 1\} \times \{0, 1\})^*$  be the regular language from Cor. 6 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  $\Sigma = \{0, 1, \bar{0}, \bar{1}\}$  with  $\text{val}(\mathbb{C}) = \overline{\text{val}(\mathbb{B})} \text{val}(\mathbb{A})$ . Let  $V = (Q, q_0, \{\perp, 0, 1\}, \perp, \Delta, F)$  be the 1-turn VPA over  $(\{\bar{0}, \bar{1}\}, \{0, 1\})$  with the following transitions:

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

Thus,  $V$  can only read words of the form  $\bar{v}u$  with  $u, v \in \{0, 1\}^*$  and  $|v| \geq |u|$  (recall that  $q_0$  has no incoming transitions). When reading such a word  $\bar{v}u$ ,  $V$  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. From the construction of  $V$ , we obtain

$$\text{val}(\mathbb{C}) = \overline{\text{val}(\mathbb{B})} \text{val}(\mathbb{A}) \in L(V) \iff \text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B}) \in L(A) \wedge |\text{val}(\mathbb{A})| \leq |\text{val}(\mathbb{B})|.$$

By Cor. 6 (and the remark after the proof), this concludes the proof.  $\blacksquare$

Prop. 2 and Thm. 7 imply:

**COROLLARY 8.**  $\text{PSPACE} = \text{LEAF}_a^L(K)$  for some linear visibly pushdown language  $K$ .

In [21], 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,  $\Sigma$  and  $\Gamma$  are finite alphabets,  $\delta : Q \times \Sigma \rightarrow Q^+$  is the transition mapping,  $\rho : Q \rightarrow \Gamma$  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  $\hat{\delta}(q, w)$  as follows:  $\hat{\delta}(q, \varepsilon) = q$  and  $\hat{\delta}(q, au) = \hat{\delta}(q_1, u) \cdots \hat{\delta}(q_n, u)$  if  $a \in \Sigma$  and  $\delta(q, a) = q_1 \cdots q_n$ . Let  $\text{leaf}(A, w) = \rho(\hat{\delta}(q_0, w))$ , where  $\rho : Q \rightarrow \Gamma$  is extended to a morphism on  $Q^*$ . For  $K \subseteq \Gamma^*$  let  $\text{LEAF}(A, K) = \{w \in \Sigma^* \mid \text{leaf}(A, w) \in K\}$  and  $\text{LEAF}(K) = \{\text{LEAF}(A, K) \mid A \text{ is an FLA}\}$ .

**THEOREM 9.** *There exists a fixed linear visibly pushdown language  $K$  and an FLA  $A$  such that  $\text{LEAF}(A, K)$  is PSPACE-complete w.r.t. logspace reductions.*

**PROOF.** We use the linear visibly pushdown language  $K$  from the proof of Thm. 7. Notice that the question whether  $\text{val}(\mathbb{C}) \in K$  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 Thm. 5), we see that it is already PSPACE-complete to check for a number  $t_0$  and two superdecreasing sequences  $(t_1, \dots, t_m), (w_1, \dots, w_k)$  (all numbers are encoded binary) whether

$$\overline{S(w_1, \dots, w_k)} 0^{t_0} S(t_1, \dots, t_m) 0^{w_1 + \dots + w_k - (t_0 + \dots + t_m)} \in K. \quad (5)$$

Here we use again the encoding of superdecreasing sequences from (1). So, it remains to find an FLA  $A$  with the following property: from given input data  $t_0, (t_1, \dots, t_m), (w_1, \dots, w_k)$  as above we can construct in logspace a string  $w$  such that  $\text{leaf}(A, w)$  is exactly the string in (5). We only present an FLA  $A$  and a logspace construction of a string  $w$  from a superdecreasing sequence  $(w_1, \dots, w_k)$  such that  $\text{leaf}(A, w) = S(w_1, \dots, w_k)$ . 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  $d = 2^{e_1} + 2^{e_2} + \dots + 2^{e_m}$  ( $e_1 < e_2 < \dots < e_m$ ):

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

Next, we derive in logspace from the superdecreasing sequence  $(w_1, \dots, w_k)$  the sequence  $(d_1, \dots, d_k)$  of differences as defined in (2) and encode it by the string

$$e(d_1, \dots, d_k) = \left( \prod_{i=1}^{k-1} \#e(d_i) \right) \#e(d_k) \in \{a, \$, \tilde{\$}, \#, \tilde{\#}\}^*$$

Our fixed FLA is  $A = (\{q_0, p_r, p_\ell, r_0, r_1\}, \{a, \$, \tilde{\$}, \#, \tilde{\#}\}, \{0, 1\}, \delta, \rho, q_0)$ , where the transition function  $\delta$  is defined as follows:

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

The  $\delta$ -values that are not explicitly defined can be set arbitrarily. Finally, let  $\rho(r_0) = 0$  and  $\rho(r_1) = 1$ ; all other  $\rho$ -values can be defined arbitrarily. We claim that  $\text{leaf}(A, e(d_1, \dots, d_k)) = S(w_1, \dots, w_k)$ . First note that  $\hat{\delta}(p_r, a^e \$) = r_0^e p_r$  and  $\hat{\delta}(p_r, a^e \tilde{\$}) = r_0^{2^e}$ . Since  $\delta(r_0, x) = r_0$  for all input symbols  $x$ , we have  $\hat{\delta}(p_r, e(d)) = r_0^d$  for every number  $d$  and therefore:

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

Hence, the FLA  $A$  realizes the recurrence (3) when reading the input  $e(d_1, \dots, d_k)$ . ▀

## 5 Compressed membership in XML languages

In this 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  $\bar{B}$  be the set of corresponding closing brackets. An *XML-grammar* [4] 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)  $B \cup \bar{B}$  is the set of terminals, (iii)  $X_a$  is the start variable, and (iv) the (infinite) set of productions is  $\{X_b \rightarrow b w \bar{b} \mid b \in B, w \in R_b\}$ . Since  $R_b$  is regular, this set is equivalent to a finite set of productions. One can show that  $L(G)$  is a visibly pushdown language [1]. XML-grammars capture the syntactic features of XML document type definitions (DTDs), see [4] for details.

**THEOREM 10.** *For every XML-grammar  $G$ ,  $\text{CMP}(L(G))$  belongs to  $\text{coNP}$ . Moreover, there is an XML-grammar  $G$  such that  $\text{CMP}(L(G))$  is  $\text{coNP}$ -complete w.r.t. logspace reductions.*

For the proof of the upper bound in Thm. 10 we need a few definitions. Let us fix an XML-grammar  $G = (B, (R_b)_{b \in B}, a)$  for the further considerations. The set  $D_B \subseteq (B \cup \bar{B})^+$  of all *Dyck primes* over  $B$  is the set of all well-formed strings over  $B \cup \bar{B}$  that do not have a non-empty proper prefix, which is well-formed as well. Formally,  $D_B$  is the smallest set such that  $w_1, \dots, w_n \in D_B$  ( $n \geq 0$ ) implies  $b w_1 \dots w_n \bar{b} \in D_B$ . For  $b \in B$  let  $D_b = D_B \cap b(B \cup \bar{B})^* \bar{b}$ . The set of all *Dyck words* over  $B \cup \bar{B}$  is  $D_B^*$ . Note that  $L(G) \subseteq D_a$ .

Let  $w \in D_B^*$ , and let  $1 \leq i \leq |w|$  be a position with  $w[i] \in B$ , i.e. the  $i$ -th symbol in  $w$  is an opening bracket. Since  $w \in D_B^*$ , there exists a unique position  $\gamma(w, i) > i$  with  $w[i, \gamma(w, i)] \in$

$D_B$ . The string  $w[i+1, \gamma(w, i) - 1]$  belongs to  $D_B^*$ . Since  $D_B$  is a code, there exists a unique factorization  $w[i+1, \gamma(w, i) - 1] = w_1 \cdots w_n$  with  $n \geq 0$  and  $w_1, \dots, w_n \in D_B$ . 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  $\text{surface}(w, i) = X_{b_1} X_{b_2} \cdots X_{b_n}$ . The term “surface” is motivated by the surface of  $b \in B$  from [4]. A straightforward induction shows:

**LEMMA 11.** *Let  $w \in (B \cup \bar{B})^*$ . Then  $w \in L(G)$  if and only if (i)  $w \in D_a$  and (ii)  $\text{surface}(w, j) \in R_b$  for every position  $1 \leq j \leq |w|$  such that  $w[j] = b \in B$ .*

The next lemma was shown in [19, Lemma 5.6]:

**LEMMA 12.** *CMP( $D_B^*$ ) can be solved in polynomial time. Moreover, for a given SLP  $\mathbb{A}$  such that  $w := \text{val}(\mathbb{A}) \in D_B^*$  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 12 and the fact  $w \in D_B \iff (w \in D_B^* \text{ and } \gamma(w, 1) = |w|)$  implies:

**PROPOSITION 13.** *CMP( $D_B$ ) can be solved in polynomial time.*

For the proof of Thm. 10 we need one more technical lemma, whose proof has to be omitted in this short version:

**LEMMA 14.** *For a given SLP  $\mathbb{A}$  such that  $w := \text{val}(\mathbb{A}) \in D_B^*$  and a given (binary coded) position  $1 \leq i \leq |w|$  with  $w[i] \in B$  one can compute an SLP for the string  $\text{surface}(w, i)$  in polynomial time.*

Now we can prove Thm. 10: For the coNP upper bound, let  $G = (B, (R_b)_{b \in B}, a)$  be an XML grammar and let  $\mathbb{A}$  be an SLP over the terminal alphabet  $B \cup \bar{B}$  with  $w = \text{val}(\mathbb{A})$ . By Lemma 11 we have to check that (i)  $w \in D_a = D_B \cap a(B \cup \bar{B})^* \bar{a}$  and (ii)  $\text{surface}(w, j) \in R_b$  for all  $1 \leq j \leq |w|$  with  $w[j] = b \in B$ . Condition (i) can be checked in deterministic polynomial time by Prop. 13; condition (ii) belongs to coNP by Lemma 14 and Thm. 1. The proof of the coNP lower bound is similar to the proof of [19, Thm. 5.2] and therefore omitted. ■

## References

- [1] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. STOC 2004*, 202–211. ACM Press, 2004.
- [2] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in  $\text{NC}^1$ . *J. Comput. System Sci.*, 38:150–164, 1989.
- [3] M. Beaudry, P. McKenzie, P. Péladéau, and D. Thérien. Finite monoids: From word to circuit evaluation. *SIAM J. Comput.*, 26(1):138–152, 1997.
- [4] J. Berstel and L. Boasson. Formal properties of XML grammars and languages. *Acta Inform.*, 38(9):649–671, 2002.
- [5] A. Bertoni, C. Choffrut, and R. Radicioni. Literal shuffle of compressed words. In *Proc. IFIP TCS 2008*, 87–100. Springer, 2008.
- [6] 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.
- [7] D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoret. Comput. Sci.*, 104(2):263–283, 1992.

- [8] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic  $NC^1$  computation. *J. Comput. System Sci.*, 57(2):200–212, 1998.
- [9] P. W. Dymond. Input-driven languages are in  $\log n$  depth. *Inform. Process. Lett.*, 26(5):247–250, 1988.
- [10] L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. In *Proc. SWAT 1996*, LNCS 1097, 392–403. Springer, 1996.
- [11] S. Greibach. The hardest context-free language. *SIAM J. Comput.*, 2(4):304–310, 1973.
- [12] C. Hagenah. *Gleichungen mit regulären Randbedingungen über freien Gruppen*. PhD thesis, University of Stuttgart, Institut für Informatik, 2000.
- [13] U. Hertrampf. The shapes of trees. In *Proc. COCOON 1997*, LNCS 1276, 412–421. Springer, 1997.
- [14] 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*, 200–207. IEEE Computer Society Press, 1993.
- [15] M. Holzer and K.-J. Lange. On the complexities of linear LL(1) and LR(1) grammars. In *Proc. FCT 1993*, LNCS 710, 299–308. Springer, 1993.
- [16] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Inform. and Comput.*, 129(1):21–33, 1996.
- [17] H. J. Karloff and W. L. Ruzzo. The iterated mod problem. *Inform. and Comput.*, 80(3):193–204, 1989.
- [18] Y. Lifshits and M. Lohrey. Querying and embedding compressed texts. In *Proc. MFCS 2006*, LNCS 4162, 681–692. Springer, 2006.
- [19] M. Lohrey. Word problems and membership problems on compressed words. *SIAM J. Comput.*, 35(5):1210 – 1240, 2006.
- [20] N. Markey and P. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Inform. Process. Lett.*, 90(1):3–6, 2004.
- [21] T. Peichl and H. Vollmer. Finite automata with generalized acceptance criteria. *Discrete Math. Theor. Comput. Sci.*, 4(2):179–192 (electronic), 2001.
- [22] W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA'94*, LNCS 855, 460–470. Springer, 1994.
- [23] 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*, 262–272. Springer, 1999.
- [24] H. Veith. Succinct representation, leaf languages, and projection reductions. *Inform. and Comput.*, 142(2):207–236, 1998.
- [25] N. K. Vereshchagin. Relativizable and nonrelativizable theorems in the polynomial theory of algorithms. *Izv. Ross. Akad. Nauk Ser. Mat.*, 57(2):51–90, 1993.
- [26] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.
- [27] B. von Braunmühl and R. Verbeek. Input-driven languages are recognized in  $\log n$  space. In *Proc. FCT 1983*, LNCS 158, 40–51. Springer, 1983.