On the Complexity of Computing Strahler Numbers

Moses Ganardi
University of Kaiserslautern-Landau (RPTU), Germany

Markus Lohrey
Universität Siegen, Germany

Abstract

It is shown that the problem of computing the Strahler number of a binary tree given as a term is complete for the circuit complexity class uniform NC^1. For several variants, where the binary tree is given by a pointer structure or in a succinct form by a directed acyclic graph or a tree straight-line program, the complexity of computing the Strahler number is determined as well. The problem, whether a given context-free grammar in Chomsky normal form produces a derivation tree (resp., an acyclic derivation tree), whose Strahler number is at least a given number k is shown to be P-complete (resp., PSPACE-complete).

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Grammars and context-free languages

Keywords and phrases Strahler number, circuit complexity classes, context-free grammars

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.41

Related Version *Full Version*: https://arxiv.org/abs/2512.19060 [29]

1 Introduction

Strahler numbers. The main topic of this paper is the complexity of computing *Strahler numbers* of binary trees. The *Strahler number* of a binary tree t is a parameter $\mathsf{st}(t)$ that can be defined recursively as follows:

- If t consists of a single node then $\mathsf{st}(t) = 0$.
- If the root of t has the left (resp., right) subtree t_1 (resp., t_2) then

$$\mathsf{st}(t) = \begin{cases} \mathsf{st}(t_1) + 1 & \text{if } \mathsf{st}(t_1) = \mathsf{st}(t_2), \\ \max\{\mathsf{st}(t_1), \mathsf{st}(t_2)\} & \text{if } \mathsf{st}(t_1) \neq \mathsf{st}(t_2). \end{cases} \tag{1}$$

The Strahler number is sometimes also called the *Horton-Strahler number* and first appeared in the area of hydrology, where Horton used it in a paper from 1945 [37] to define the order of a river. The correspondence to binary trees comes from the fact that a system of joining rivers can be viewed as a binary tree (unless there are bifurcations, where a river splits into two streams). In 1952, Strahler [53] (also a hydrologist) further developed Horton's ideas.

There are numerous applications of Strahler numbers in computer science, where they appeared also under different names (e.g., register function, tree dimension). Ershov [23] showed that the minimal number of registers needed to evaluate an arithmetic expression is exactly the Strahler number of the syntax tree of the arithmetic expression. Another area, where Strahler numbers found many applications, is formal language theory [12, 20, 33, 51]. For context-free grammars, the relation comes from the following fact: Let G be a context-free grammar in Chomsky normal form and let t be a derivation tree of G. Then $\mathsf{st}(t)+1$ is exactly the minimal index among all derivations corresponding to t, where the index of a derivation $S = w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \cdots \Rightarrow_G w_n$ is the maximal number of nonterminals in one of the w_i [33]. Finite-index context-free grammars [3], i.e., grammars where every produced word has a derivation of bounded index, play an important role in the recent decidability proof of the reachability problem in one-dimensional pushdown VASS [7]. Strahler numbers

43rd International Symposium on Theoretical Aspects of Computer Science (STACS 2026).
Editors: Meena Mahajan, Florin Manea, Annabelle McIver, and Nguyễn Kim Thắng
Article No. 41; pp. 41:1–41:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

have been also investigated in the context of Newton iteration [24, 46], parity games [16], and social networks [1]. The distribution of the Strahler number of a random tree has been studied by several authors [17, 18, 26, 39, 42]. For more information on Strahler numbers and their applications in computer science, the reader may consult the surveys [25, 56].

The above mentioned applications naturally lead to the question for the precise complexity of computing the Strahler number of a given tree. This problem has a straightforward linear time algorithm: simply compute the Strahler number for all subtrees bottom-up using the definition of the Strahler number. A straightforward recursive algorithm can be implemented on a deterministic Turing machine, running in logspace and polynomial time and equipped with an auxiliary stack, which puts the problem in the class $\mathsf{LogDCFL} \subseteq \mathsf{NC}^2 \subseteq \mathsf{DSPACE}(\log^2 n)$ [54]. In particular, Strahler numbers can be computed in polylogarithmic time with polynomially many processors. Alternatively, one can implement a recursive evaluation in $\mathcal{O}(\log n \log\log n)$ space. To do so, one uses the fact that the Strahler number of a binary tree with n leaves is bounded by $\log_2(n)$, and hence its bit length is $\mathcal{O}(\log\log n)$. To the best of our knowledge, the precise complexity of computing Strahler numbers has not been pinpointed yet.

Contributions. Our first goal is to pinpoint the precise parallel complexity of computing the Strahler number. As explained above, the problem belongs to NC, but the existence of a logspace algorithm for instance is by no means obvious. Formally, we consider the decision problem, asking whether $\mathsf{st}(t) \geq k$ for a given tree t and a given number k. We show that this problem is NC^1-complete if t is given in term representation.[1] Recall that NC^1 is the class of all problems that can be solved by a uniform[2] family of bounded fan-in circuits of polynomial-size and logarithmic depth, which is a subclass of deterministic logarithmic space (L for short). If the term t is given as a pointer structure, i.e., by an adjacency list or matrix, then checking $\mathsf{st}(t) \geq k$ is complete for deterministic logspace (L for short).

As a corollary, one can compute in NC^1 from a given arithmetic expression e (given in term representation) an optimal straight-line code, i.e., a sequence of statements $x := y \circ z$ for registers x, y, z and an elementary arithmetic operation $\circ$. Here, optimal means that the number of used registers is minimal. For this, one has to compute the Strahler number of every subexpression of e; see also [25, Section 2].

Let us give a high level idea of the NC^1-membership proof. The first step is to "balance" the input tree t by computing a so-called *tree straight-line program* (TSLP) for t, whose depth is logarithmic in the size of t. This can be done in TC^0 by a result from [28]. Roughly speaking, a TSLP is a recursive decomposition of a tree into subtrees and so-called contexts (subtrees, where a smaller subtree is removed). Originally, TSLPs were introduced as a formalism for grammar-based tree compression; see [44] for more details. The next step is to convert the TSLP for t into a bounded fan-in Boolean circuit, that decides whether $\mathsf{st}(t) \geq k$. Furthermore, the polynomial size and logarithmic depth of the TSLP should be preserved to obtain an NC^1 upper bound. A straightforward construction only leads to such a Boolean circuit with *unbounded* fan-in OR-gates. We obtain a bounded fan-in circuit by carefully analyzing the unary linear term functions computed by contexts when binary nodes are interpreted according to (1).

[1] For instance, $bbaabaa$ (or $b(b(a,a), b(a,a))$ with brackets) is the term representation of a complete binary tree of height 2, where b denotes an inner node and a denotes a leaf.

[2] All circuit complexity classes refer to their uniform variants in this paper. In Section 2.2 we will say more about uniformity.

For the NC^1-hardness, we show that the Boolean formula problem, which is one of the best known NC^1-complete problems [9], can be reduced to the problem of computing the Strahler number of a tree given in term representation. A similar reduction from the monotone circuit value problem shows that the computation of the Strahler number is P-complete when the input tree is given succinctly by a directed acyclic graph (DAG) or a TSLP.

We also consider the problem of checking $\mathsf{st}(t) \geq k$ for a fixed value k that is not part of the input (the input only consists of the tree t). If t is given in term representation (resp., pointer representation) then this problem is TC^0-complete for all $k \geq 4$ (resp., L-complete for all $k \geq 3$). Moreover, if t is given by a DAG, then this problem belongs to $\mathsf{UL} \cap \mathsf{coUL}$ for all k (UL is unambiguous logspace), whereas for TSLP-represented trees the problem is NL-complete for all $k \geq 2$.

In Section 4 we briefly report on some results concerning the maximal Strahler number of derivation trees of a given context-free grammar in Chomsky normal form (CNF). It is known to be undecidable, whether every word produced by a given context-free grammar has a derivation tree of Strahler number at most a given bound k [34, Theorem 5]. Here, we are interested in the question, whether a given CNF-grammar G produces at least one derivation tree t with $\mathsf{st}(t) \geq k$ for a given number k. In the long version [29] we show that this problem is P-complete. Finally, we also consider the restriction to *acyclic derivation trees*. A derivation tree is called acyclic if there is no nonterminal that appears more than once on a path in the derivation tree. The motivation for this restriction comes from the recent paper [45], where it was shown that the intersection non-emptiness problem for a given list of group DFA[3] plus a single context-free grammar is PSPACE-complete. For general DFA, this problem is $\mathsf{EXPTIME}$-complete [55]. Moreover, if the context-free grammar G is such that for some constant k, all acyclic derivation trees of G have Strahler number at most k, then the intersection problem (with the finite automata restricted to group DFA) is NP-complete. In [45], it was shown that the problem whether a given CNF-grammar has an acyclic derivation tree of Strahler number at least k is in NP, when k is a fixed constant. In the long version [29] we show that the problem is NP-complete already for $k = 2$. Finally, when k is part of the input, we show that the problem becomes PSPACE-complete.

Broader context: tree evaluation and tree balancing. The problem of computing the Strahler number of a given tree is a special instance of a *tree evaluation problem*: The input is a rooted tree where each leaf is labelled with a value from a domain A, and each inner node carries a (suitably specified) function $f\colon A^r \to A$ where r is the number of its children. The goal is to compute the value of the root, obtained by evaluating the functions at each node from bottom to top. For the case of Strahler numbers we have $A = \mathbb{N}$, every leaf is labelled with 0 and there is only one binary operation implicitly defined by (1) (or explicitly by (2) on page 6). The corresponding algebra will be called the *Strahler algebra*.

Other prominent examples are the evaluation problems for Boolean formulas such as $(1 \vee 0) \wedge 1$ and arithmetic expressions over the natural numbers (or other rings) such as $(1 + 2) \times (3 + 4)$. Boolean formula evaluation is NC^1-complete [9]. In fact, the acceptance problem of a fixed tree automaton or, equivalently, evaluating an expression over a fixed finite algebra is known to be in NC^1 for every finite algebra [28, 43]. By an algebra, we simply mean a set equipped with a set of finitary operations. More surprisingly, arithmetic expressions can be evaluated in deterministic logspace [6, 10, 11] despite the fact that the value of an expression may have polynomially many bits in terms of the size of the expression.

[3] A group DFA is a deterministic finite automaton, where for every input letter a the a-labelled transitions induce a permutation of the set of states.

Any algorithm that performs a bottom-up computation over a tree can be seen as an instance of tree evaluation, assuming that the local computation at each node is sufficiently simple. A classical example is Courcelle's theorem, stating that any monadic second-order (MSO) definable graph property Φ can be checked in linear time over graphs of bounded tree-width [15]. The standard proof of Courcelle's theorem compiles the MSO formula Φ into a tree automaton $\mathcal{A}_\Phi$ that, given a tree decomposition of a graph G, verifies whether Φ holds in G. As remarked above, tree automata can be simulated in $\mathsf{NC}^1 \subseteq \mathsf{L}$, and therefore Courcelle's theorem also holds when linear time is replaced by logspace [21] (assuming the logspace version of Bodlaender's theorem for computing small-width tree decompositions; see [21]). In fact, [21] proves a more powerful *solution histogram* version of Courcelle's theorem, which, in the end, reduces to evaluating arithmetic expressions.

Very recently, the tree evaluation problem attracted new attention due to a surprising result by Cook and Mertz [13]. They presented an algorithm that evaluates a complete binary tree of height h, whose inner nodes are labelled with binary operations over $\{1, \ldots, k\}$ and whose leaves are labelled with elements from $\{1, \ldots, k\}$, in space $\mathcal{O}(h \log\log k + \log k)$. A straightforward evaluation takes $\mathcal{O}(h \log k)$ space. Since the height h is logarithmic in the total input size n, the Cook-Mertz algorithm uses $\mathcal{O}(\log n \log\log n)$ space, which comes very close to $\mathcal{O}(\log n)$ space. It is also a key ingredient in Ryan Williams' recent proof that any t-time bounded Turing machine can be simulated in $\mathcal{O}(\sqrt{t \log t})$ space [58]. Notice that the Cook-Mertz algorithm does not give any nontrivial space bounds for the computation of the Strahler number of a tree t, since the height of t can be linearly large in its size.

A standard strategy to evaluate a tree t of size n using small space or in parallel polylogarithmic time is to first *balance* t, i.e., to transform it into an equivalent tree of depth $\mathcal{O}(\log n)$ and size $\mathsf{poly}(n)$. In a second step, the reduced depth can often be exploited to evaluate the tree in parallel or in small space. For example, to evaluate a balanced arithmetic expression in logspace, one can use a result by Ben-Or and Cleve [6] that transforms an arithmetic expression of depth d into a product of 4^d many (3×3)-matrices such that the value of the arithmetic expression appears as a particular entry in the matrix product. The matrix product can in turn be evaluated in logspace using results from [11].

Balancing algorithms were first presented by Spira [52] for Boolean formulas and by Brent [8] for arithmetic expressions. Later work showed that arithmetic expressions can be balanced in NC^1 (observed implicitly in [10]) and in fact in TC^0 [28]. A generic framework for evaluating trees was presented in [41], which implicitly balances the input tree in NC^1. The above mentioned logspace version of Courcelle's theorem was improved to NC^1 [22] (under an appropriate input form) in subsequent work. The first step of that algorithm is to balance a given tree decomposition in TC^0.

In general, not every algebra admits such a depth-reduction result, if one requires that the balanced tree is over the *same* algebra [40, Theorem 1]. The core of most tree balancing approaches is a purely syntactic recursive decomposition of the input tree into subtrees and contexts (subtrees where a subtree was removed) and the depth of this decomposition is bounded logarithmically in the size of the input tree. Formally, this decomposition is a tree straight-line program of logarithmic depth. While subtrees evaluate to elements, contexts describe unary linear term functions over the algebra. For example, over a commutative semiring a context computes an affine function $x \mapsto ax + b$, and can be represented by the parameters a, b. Furthermore, the composition of two affine functions can be implemented using semiring operations on these parameters. The main challenge towards efficient tree balancing and tree evaluation over a particular algebra is understanding the structure of its unary linear term functions (called the *functional algebra* in [41]). In general this can be difficult, as can be seen from the example of a finite algebra: Given a tree automaton with

k states, the contexts can induce up to k^k many state transformations. In particular, the space bound of $\mathcal{O}(h \log\log k + \log k)$ achieved by the Cook-Mertz algorithm for an algebra of size k cannot be immediately extended to unbalanced trees by applying the Cook-Mertz algorithm to a balanced tree straight-line program for the original tree, since k would blow up to k^k. For the special case of the Strahler algebra we provide a characterization of the unary linear term functions computed by contexts in Section 3.

2 Preliminaries

We assume some familiarity with formal language theory, in particular with context-free grammars; see e.g. [36] for details. The set of all finite words over an alphabet Γ is denoted with Γ^*; it includes the empty word ε. The length of a word $w \in \Gamma^*$ is $|w|$ and the number of occurrences of $a \in \Gamma$ in the word w is denoted with $|w|_a$.

2.1 Directed acyclic graphs, trees, contexts

Directed acyclic graph. We have to deal with node-labelled directed acyclic graphs (DAGs). Let us fix a ranked alphabet Σ (possibly infinite), where every $a \in \Sigma$ has a rank in $\mathbb{N}$. Let $\Sigma_i \subseteq \Sigma$ be the set of symbols of rank $i \in \mathbb{N}$. A Σ-*labelled DAG* is a tuple $\mathcal{D} = (V, v_0, \lambda, \gamma)$ with the following properties:

- V is the finite set of nodes.
- $v_0 \in V$ is a distinguished root.
- $\lambda\colon V \to \Sigma$ is a mapping that assigns to every node $v \in V$ its label $\lambda(v)$.
- $\gamma\colon V \to V^*$ is a function such that $\lambda(v) \in \Sigma_{|\gamma(v)|}$. It assigns to every node v the list $\gamma(v)$ of v's children (a node may occur more than once in this list).
- We require that the directed graph $(V, \{(u,v) : v \text{ appears in } \gamma(u)\})$ is acyclic.

Sometimes we do not need the labelling function λ, in which case we omit λ from the description of the DAG.

We also write $d(v) = |\gamma(v)|$ for the *degree* of the node $v \in V$. Nodes of degree zero are also called *leaves*. For every $v \in V$ and $1 \le i \le d(v)$ we define $v \cdot i$ as the i^{th} node in the word $\gamma(v)$. This notation can be extended to words $\alpha \in \mathbb{N}^*$ (so-called *address strings*) inductively: $v \cdot \varepsilon = v$ and if $\alpha = \beta i$, $v \cdot \beta$ is defined and $1 \le i \le d(v \cdot \beta)$ then $v \cdot \alpha = (v \cdot \beta) \cdot i$. We define the *size* $|\mathcal{D}|$ of $\mathcal{D}$ as $|\mathcal{D}| = \sum_{v \in V}(d(v)+1)$.

A path in $\mathcal{D}$ can be specified by its start node v and an address string $\alpha = i_1 i_2 \cdots i_n \in \mathbb{N}^*$. The corresponding path consists of the nodes $v, v \cdot i_1, v \cdot i_1 i_2, \ldots, v \cdot \alpha$. For a node $v \in V$ we define $\mathsf{height}_{\mathcal{D}}(v) = \max\{|\alpha| : \alpha \in \mathbb{N}^*, v \cdot \alpha \text{ is defined}\}$. Moreover, the *height* (or *depth*) of $\mathcal{D}$ is $\max\{\mathsf{height}_{\mathcal{D}}(v) : v \in V\}$.

In the following, we will mainly consider *binary* DAGs where $d(v) \le 2$ for every $v \in V$.

Trees. A Σ-*labelled tree* can be defined as a Σ-labelled DAG $t = (V, v_0, \lambda, \gamma)$ as above such that in addition for every $v \in V$ there is a unique address string α such that $v = v_0 \cdot \alpha$. The node v_0 is the *root* of the tree. For a tree t and a node v we write $t(v)$ for the *subtree* of t rooted in v. It is the tree $(V', v, \lambda{\restriction_{V'}}, \gamma{\restriction_{V'}})$ where $V' = \{v \cdot \alpha : \alpha \in \mathbb{N}^*, v \cdot \alpha \text{ is defined}\}$.

From a DAG $\mathcal{D} = (V, v_0, \lambda, \gamma)$ one can define a tree $\mathsf{unfold}(\mathcal{D}) = (V', \varepsilon, \lambda', \gamma')$ (the *unfolding* of $\mathcal{D}$) as follows: The set of nodes of V' contains all address strings α such that $v_0 \cdot \alpha$ is defined and the empty string ε is the root. If $\alpha \in V'$ is such that $v = v_0 \cdot \alpha$, then $\lambda'(\alpha) = \lambda(v)$ and $\gamma'(\alpha) = (\alpha 1)(\alpha 2) \cdots (\alpha d(v))$. Figure 1(right) shows a DAG, whose unfolding is the tree on the left. The edge from a node to its i^{th} child in the DAG is labelled with i. Clearly, the size of $\mathsf{unfold}(\mathcal{D})$ can be exponential in the size of $\mathcal{D}$. This shows the potential of DAGs as a compact tree representation; see also [19, 27].

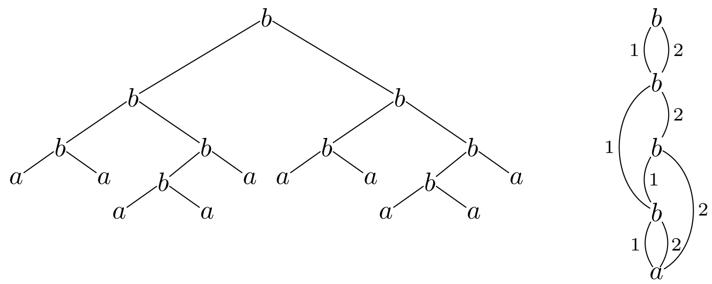

Figure 1 A binary tree.

Most trees in this paper are (unlabelled) *binary trees*, in which case we have $d(v) \in \{0, 2\}$ for all nodes v, $\Sigma_0 = \{a\}$, and $\Sigma_2 = \{b\}$. So, internal nodes are labelled with b and leaves are labelled with a. Thus, the node labels do not carry any information and can be omitted.

Input representation of trees and DAGs. When it comes to circuit complexity (see the next section), the input representation of DAGs has a big influence on complexity. The representation as a tuple $(V, v_0, \lambda, \gamma)$ is also called *pointer representation.* In the pointer representation the edges of the DAG are given by adjacency list (namely the lists $\gamma(v)$). In the case of trees, another well-known representation of a tree t is the *term representation*, where t is represented by a term formed from the symbols in Σ. For instance, the string *bbbaabbaaabbaabbaaa* (which is written as $b(b(b(a, a), b(b(a, a), a)), b(b(a, a), b(b(a, a), a)))$ for better readability) is the term representation of the binary tree shown in Figure 1. It is obtained by listing the node labels of the binary tree in preorder. With Bin we denote the set of all $x \in \{a, b\}^*$ that are the term representation of a binary tree. It can be produced by the context-free grammar with the productions $S \to a \mid bSS$.

For binary DAGs, we also use the so-called extended connection representation, which extends the pointer representation by a further relation; see also [28, 48] and [57, Definition 2.43]. Consider a binary DAG $\mathcal{D} = (V, v_0, \lambda, \gamma)$ as above. The *extended connection representation*, briefly *ec-representation*, of $\mathcal{D}$, denoted by $\mathrm{ec}(\mathcal{D})$, is the tuple $(V, v_0, \lambda, \gamma, \mathsf{ec}_{\mathcal{D}})$, where the set $\mathsf{ec}_{\mathcal{D}}$ consists of all so-called *ec-triples* $(v, \alpha, v \cdot \alpha)$, where $v \in V$, $\alpha \in \{1, 2\}^*$ is an address string such that $v \cdot \alpha$ is defined and $|\alpha| \le \log_2 |\mathcal{D}|$. Note that since $\mathcal{D}$ is binary, the number of address strings with $|\alpha| \le \log_2 |\mathcal{D}|$ is bounded by $\mathcal{O}(|\mathcal{D}|)$.

Contexts. Fix a so-called placeholder symbol $x \notin \{a, b\}$. A *binary context* is a binary tree t, where exactly one leaf $v \in V_0$ is labelled with x. All other leaves are labelled with a and internal nodes are labelled with b. Given a binary context t and a binary tree (resp., context) t' we define the binary tree (resp., context) $t[x/t']$ by replacing the unique occurrence of x in t by t'. For instance, we have $b(b(a, a), b(x, a))[x/b(a, a)] = b(b(a, a), b(b(a, a), a))$.

Strahler numbers. Let s be the binary operation on $\mathbb{N}$ with

$$s(x, y) = \begin{cases} x + 1 & \text{if } x = y, \\ \max(x, y) & \text{if } x \neq y. \end{cases} \tag{2}$$

The algebraic structure $\mathcal{S} = (\mathbb{N}, s, 0)$ is also called the *Strahler algebra* in the following. The Strahler number $\mathsf{st}(t)$ of a binary tree $t \in \mathsf{Bin}$ is defined as follows:

$$\mathsf{st}(a) = 0, \qquad \mathsf{st}(b(t_1, t_2)) = s(\mathsf{st}(t_1), \mathsf{st}(t_2)).$$

In other words: $\mathsf{st}(t)$ is obtained by evaluating t in the Strahler algebra, where the binary symbol b is interpreted by s and the leaf symbol a is interpreted by 0. The tree from Figure 1 has Strahler number 3.

It is well-known that if t has n leaves then $\mathsf{st}(t) \le \log_2 n$: Let $m = \mathsf{st}(t)$. The case $m = 0$ is clear. If $m > 0$ then the root of t must have at least two descendants with Strahler number $m - 1$. By induction it follows that t has at least 2^i many nodes with Strahler number $m - i$. Thus, t has at least 2^m many leaves (= nodes with Strahler number 0). Moreover, the Strahler number of a tree t is the largest k such that a complete binary tree t_k of depth k can be embedded into t (thereby, edges of t_k can be mapped to non-empty paths in t).

2.2 Computational complexity

We assume that the reader is familiar with the complexity classes L (deterministic logspace), NL (nondeterministic logspace), P, NP and PSPACE; see e.g. [2] for details. The class UL (unambiguous logspace) is the class of all languages that can be recognized by a nondeterministic logspace Turing machine that has on each input word at most one accepting computation. It is conjectured that $\mathsf{UL} = \mathsf{NL}$. In the nonuniform setting this has been shown in [47].

In the rest of this section we briefly introduce some well-known concepts from circuit complexity, more details can be found in the monograph [57].

A *(Boolean) circuit* with n inputs can be defined as a Σ-labelled DAG $\mathcal{B} = (V, v_0, \lambda, \gamma)$, where the set of node labels Σ consists of the symbols $x_1, \ldots, x_n$ of arity 0 (the input variables) and additional Boolean functions of arbitrary arity (we identify here a k-ary Boolean function with a k-ary node label). The set of these Boolean functions is also called the *Boolean base* of $\mathcal{B}$. Nodes of $\mathcal{B}$ are usually called *gates* and the degree $d(v)$ of a gate v is called its *fan-in*.

A Boolean circuit $\mathcal{B} = (V, v_0, \lambda, \gamma)$ as above defines a mapping $\eta_{\mathcal{B}} : \{0,1\}^n \to \{0,1\}$ in the natural way: Let $w = a_1 a_2 \cdots a_n \in \{0,1\}^n$. First define $\eta_v(w)$ for every gate v inductively: $\eta_v(w) = a_i$ if $\lambda(v) = x_i$ and $\eta_v(w) = f(\eta_{v_1}(w), \eta_{v_2}(w), \ldots, \eta_{v_d}(w))$ if $\gamma(v) = v_1 v_2 \cdots v_d$ and $\lambda(v) = f$ (a Boolean function of arity d). Finally, we set $\eta_{\mathcal{B}}(w) = \eta_{v_0}(w)$.

The complexity class NC^1 contains all languages $L \subseteq \{0,1\}^*$ such that there exists a circuit family $(\mathcal{B}_n)_{n \in \mathbb{N}}$ where

- $\mathcal{B}_n$ is a Boolean circuit with n inputs over the Boolean base consisting of the unary function $\neg$ (negation) and the binary functions $\wedge$ (conjunction) and $\vee$ (disjunction),
- $\mathcal{B}_n$ has size $n^{\mathcal{O}(1)}$ and depth $\mathcal{O}(\log n)$ and
- for every $w \in \{0,1\}^n$, $\eta_{\mathcal{B}_n}(w) = 1$ if and only if $w \in L$.

Important subclasses of NC^1 are AC^0 and TC^0. The class AC^0 is defined similarly to NC^1 with the following modifications:

- The Boolean base of $\mathcal{B}_n$ consists of $\neg$ and disjunctions and conjunctions of any arity.
- The depth of the circuit $\mathcal{B}_n$ is bounded by a fixed constant.

If one includes in the first point also majority functions of any arity in the Boolean base, then one obtains the class TC^0. The m-ary majority function returns 1 if and only if more than $m/2$ many input bits are 1.

We only use the $\mathsf{DLOGTIME}$-uniform variants of AC^0, TC^0 and NC^1. For AC^0 and TC^0, $\mathsf{DLOGTIME}$-uniformity means that for a given tuple $(1^n, u, v)$, where 1^n is the unary encoding of $n \in \mathbb{N}$ and u and v are binary encoded gates of the n-th circuit $\mathcal{B}_n$, one can

(i) compute the label of gate u in time $\mathcal{O}(\log n)$ and
(ii) check in time $\mathcal{O}(\log n)$ whether u is an input gate for v.

Note that since the number of gates of $\mathcal{B}_n$ is polynomially bounded in n, the gates of $\mathcal{B}_n$ can be encoded by bit strings of length $\mathcal{O}(\log n)$.

The definition of DLOGTIME-uniform NC^1 is similar, but instead of (ii) one requires that for given 1^n, u, v as above and an address string $\alpha \in \{1,2\}^*$ with $|\alpha| \leq \log_2 |\mathcal{B}_n|$ one can check in time $\mathcal{O}(\log n)$ whether $u = v \cdot \alpha$ [4, 48, 57]. In other words, the relations from the ec-representation of $\mathcal{B}_n$ can be verified in time $\mathcal{O}(\log n)$. We denote the DLOGTIME-uniform variants of AC^0, TC^0 and NC^1 with uAC^0, uTC^0 and uNC^1, respectively. It is known that uNC^1 coincides with ALOGTIME (logarithmic time on an alternating random access Turing machine). The following inclusions hold between the complexity classes introduced above:

$$\mathsf{uAC}^0 \subsetneq \mathsf{uTC}^0 \subseteq \mathsf{uNC}^1 = \mathsf{ALOGTIME} \subseteq \mathsf{L} \subseteq \mathsf{UL} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}.$$

The definitions of the above circuit complexity classes can be easily extended to functions $f\colon \{0,1\}^* \to \{0,1\}^*$. This can be done by encoding f by the language $L_f = \{1^i 0 w : w \in \{0,1\}^*, \text{ the } i\text{-th bit of } f(w) \text{ is } 1\}$.

Hardness for uNC^1 (resp., uTC^0) is always understood with respect to uTC^0-computable (resp., uAC^0-computable) many-one reductions.

Typical problems in uTC^0 are the computation of the integer quotient of binary encoded integers, and the sum and product of an arbitrary number of binary encoded integers [35]. The canonical uTC^0-complete language is $\mathsf{Majority} = \{x \in \{0,1\}^* : |x|_1 > |x|/2\}$. Also the language Bin from Section 2.1 is uTC^0-complete. Membership in uTC^0 was shown in [43], and uTC^0-hardness can be easily shown by a reduction from the uTC^0-complete language $\{w \in \{0,1\}^* : |w|_0 = |w|_1\}$.

A famous uNC^1-complete problem is the Boolean formula problem: the input is a binary tree t in term representation using the binary symbols $\wedge$ and $\vee$ and the constant symbols 0 (for false) and 1 (for true), and the question is whether t evaluates to 1 in the Boolean algebra. Buss has shown the following theorem (note that the negation operator $\neg$ is not needed for uNC^1-hardness in [9]):

▶ **Theorem 2.1** ([9]). *The Boolean formula problem is complete for* uNC^1*.*

The following results are well-known and easy to show. Let t be an arbitrary binary tree.

- From the term representation of t one can compute in uTC^0 its pointer representation.
- From the pointer representation of t one can compute in logspace its term representation. This transformation cannot be done in uNC^1 unless $\mathsf{L} = \mathsf{uNC}^1$ holds [5].

The following lemma has been shown in [28].

▶ **Lemma 2.2** ([28, Lemma 3.4]). *For any $c > 0$ there exists a* uTC^0*-computable function, which maps the ec-representation of a DAG $\mathcal{D}$ of size n and depth at most $c \cdot \log_2 n$ to the term representation of the tree* $\mathsf{unfold}(\mathcal{D})$*.*

2.3 Tree straight-line programs

In this section, we introduce *tree straight-line programs (TSLPs)*, which have been studied mainly as a compressed representation of trees; see [44] for a survey. Here, we define tree straight-line programs only for unlabelled binary trees. A tree straight-line program (TSLP) is a tuple $\mathcal{G} = (N_0, N_1, S, \rho)$ with the following properties:

- N_0 is a finite set of *tree variables.* Tree variables are considered as symbols of rank 0.
- N_1 is a finite set of *context variables.* Context variables are considered as symbols of rank 1. Let $N = N_0 \cup N_1$. We assume that $N_0 \cap N_1 = \emptyset$.
- $S \in N_0$ is the *start variable.*

- ρ maps every $A \in N_0$ to an expression $\rho(A)$ that has one of the following three forms, where $B, C \in N_0$ and $D \in N_1$: a, $b(B, C)$, $D(C)$ (recall from Section 2.1 that a labels the leaves of a binary tree and b labels internal nodes).
- ρ maps every $A \in N_1$ to an expression $\rho(A)$ that has one of the following three forms, where $B \in N_0$ and $C, D \in N_1$: $b(x, B)$, $b(B, x)$, $D(C(x))$ (here, x is the placeholder symbol from contexts; see Section 2.1).
- The binary relation $\{(A, B) \in N \times N : B \text{ occurs in } \rho(A)\}$ must be acyclic.

For a TSLP $\mathcal{G} = (N_0, N_1, S, \rho)$ one should see the function ρ as a set of term rewrite rules $A \to \rho(A)$ for $A \in N$. With these rewrite rules, we can derive from every $A \in N_0$ (resp., $A \in N_1$) a binary tree (resp., a binary context; see Section 2.1) $\mathsf{val}_{\mathcal{G}}(A)$ (the value of A). We omit the index $\mathcal{G}$ if it is clear from the context. Formally, we define $\mathsf{val}_{\mathcal{G}}(A)$ as follows:

- if $A \in N_0$ and $\rho(A) = a$ then $\mathsf{val}_{\mathcal{G}}(A) = a$,
- if $A \in N_0$ and $\rho(A) = b(B, C)$ then $\mathsf{val}_{\mathcal{G}}(A) = b(\mathsf{val}_{\mathcal{G}}(B), \mathsf{val}_{\mathcal{G}}(C))$,
- if $A \in N_0$ and $\rho(A) = D(C)$ then $\mathsf{val}_{\mathcal{G}}(A) = \mathsf{val}_{\mathcal{G}}(D)[x/\mathsf{val}_{\mathcal{G}}(C)]$,
- if $A \in N_1$ and $\rho(A) = b(x, B)$ then $\mathsf{val}_{\mathcal{G}}(A) = b(x, \mathsf{val}_{\mathcal{G}}(B))$,
- if $A \in N_1$ and $\rho(A) = b(B, x)$ then $\mathsf{val}_{\mathcal{G}}(A) = b(\mathsf{val}_{\mathcal{G}}(B), x)$,
- if $A \in N_1$ and $\rho(A) = D(C(x))$ then $\mathsf{val}_{\mathcal{G}}(A) = \mathsf{val}_{\mathcal{G}}(D)[x/\mathsf{val}_{\mathcal{G}}(C)]$.

Finally, we define the binary tree $\mathsf{val}(\mathcal{G}) = \mathsf{val}_{\mathcal{G}}(S)$ (recall that $S \in N_0$).

▶ **Example 2.3.** Consider the TSLP $\mathcal{G}$ with $N_0 = \{S, A, B, C, D\}$, $N_1 = \{E\}$ and the following rules: $S \to b(A, A)$, $A \to b(B, C)$, $C \to E(B)$, $B \to E(D)$, $E(x) \to b(x, D)$, $D \to a$. Then $\mathsf{val}(\mathcal{G})$ is the tree from Figure 1.

A TSLP $\mathcal{G} = (N_0, N_1, S, \rho)$ can be encoded by a Σ-labelled DAG $(N_0 \cup N_1, S, \lambda, \gamma)$ with $\Sigma_0 = \{a\}$, $\Sigma_1 = \{b_1, b_2\}$ and $\Sigma_2 = \{b, \circ_0, \circ_1\}$ in the following way:

- if $A \in N_0$ and $\rho(A) = a$ then $\lambda(A) = a$ and $\gamma(A) = \varepsilon$,
- if $A \in N_0$ and $\rho(A) = b(B, C)$ then $\lambda(A) = b$ and $\gamma(A) = BC$,
- if $A \in N_0$ and $\rho(A) = D(C)$ then $\lambda(A) = \circ_0$ and $\gamma(A) = DC$,
- if $A \in N_1$ and $\rho(A) = b(x, B)$ then $\lambda(A) = b_1$ and $\gamma(A) = B$,
- if $A \in N_1$ and $\rho(A) = b(B, x)$ then $\lambda(A) = b_2$ and $\gamma(A) = B$,
- if $A \in N_1$ and $\rho(A) = D(C(x))$ then $\lambda(A) = \circ_1$ and $\gamma(A) = DC$.

In particular, we can speak about the ec-representation of a TSLP or the height of a variable in a TSLP. We define the size $|\mathcal{G}|$ of the TSLP $\mathcal{G}$ as the size of the corresponding DAG, which is bounded by $3|N|$. It is easy to see that the tree $\mathsf{val}(\mathcal{G})$ has at most $2^{\mathcal{O}(|\mathcal{G}|)}$ many nodes.

Note that for a TSLP $\mathcal{G}$, where $N_1 = \emptyset$ (hence, every $\rho(A)$ is either a or $b(B, C)$ for $B, C \in N_0$), the unfolding of the above DAG is $\mathsf{val}(\mathcal{G})$. In general, TSLPs can be more succinct than DAGs: take for instance a caterpillar tree $t = b(b(\ldots b(a, a), a), \ldots, a)$ of size n. It can be represented by a TSLP of size $\mathcal{O}(\log n)$, whereas every DAG that unfolds into t has size $\Omega(n)$. The following result from [28] will be important in the next section.

▶ **Theorem 2.4** ([28, Theorem 5.6])**.** *From a binary tree t of size n given in term representation one can compute in* uTC^0 *the ec-representation of a TSLP $\mathcal{G}$ of depth $\mathcal{O}(\log n)$ and size $\mathcal{O}(n)$ such that* $\mathsf{val}(\mathcal{G}) = t$.

The size bound $\mathcal{O}(n)$ for the TSLP $\mathcal{G}$ in Theorem 2.4 can be even replaced by $\mathcal{O}(n/\log n)$ [28, Theorem 5.6], but this is not important for our purpose.

3 Complexity of computing the Strahler number

In this section we consider the problem of checking whether the Strahler number of a given binary tree is at least a given threshold. The problem $\mathsf{St}^{\geq}$ is defined as follows:

- Input: a binary tree t and a number k.
- Question: Is $\mathsf{st}(t) \geq k$?

If we fix the value $k \geq 1$, then we obtain the following problem $\mathsf{St}^{\geq k}$:

- Input: a binary tree t.
- Question: Is $\mathsf{st}(t) \geq k$?

These problem descriptions are actually incomplete, since we did not fix the input encoding of t, which influences the complexity of the problems. We obtain the following variations: In $\mathsf{St}^{\geq}_{\mathsf{term}}$ (resp., $\mathsf{St}^{\geq}_{\mathsf{pointer}}$) the tree t is given by its term (resp., pointer) representation. In $\mathsf{St}^{\geq}_{\mathsf{dag}}$ (resp., $\mathsf{St}^{\geq}_{\mathsf{tslp}}$) the tree t is given succinctly by a binary DAG $\mathcal{D}$ (resp., a TSLP $\mathcal{G}$) such that $t = \mathsf{unfold}(\mathcal{D})$ (resp., $t = \mathsf{val}(\mathcal{G})$). The problems $\mathsf{St}^{\geq k}_{\mathsf{term}}$, $\mathsf{St}^{\geq k}_{\mathsf{pointer}}$, $\mathsf{St}^{\geq k}_{\mathsf{dag}}$, and $\mathsf{St}^{\geq k}_{\mathsf{tslp}}$ are defined analogously. Our main result is:

▶ **Theorem 3.1.** $\mathsf{St}^{\geq}_{\mathsf{term}}$ *is* uNC^1*-complete.*

As a gentle introduction into the problem, we first present a weaker result, namely that one can calculate the Strahler number of a tree with n leaves in $\mathcal{O}(\log n \log\log n)$ space, and then show how to reduce the space complexity to $\mathcal{O}(\log n)$. We only sketch the proof, since these space bounds are subsumed by the uNC^1 upper bound, proven later in this section.

The idea is to compute the Strahler number recursively by traversing the tree, in a depth-first order. We perform a depth-first traversal through the tree, maintaining a single pointer to the current node using $\mathcal{O}(\log n)$ space, and a constant-sized information, indicating the direction of the next traversal step. Additionally, we store a list of the Strahler numbers $s_1, \dots, s_k$ of the inclusion-wise maximal subtrees $t_1, \dots, t_k$ that have been completely traversed in that order (i.e., s_i has been traversed before s_j for $i < j$). Whenever both subtrees of a node have been traversed, we can combine their Strahler numbers to obtain the Strahler number of the parent node. Since each Strahler number s_i is bounded by $\log n$ where n is the number of leaves in the input tree, it can be stored in $\mathcal{O}(\log\log n)$ bits. However, if the tree is traversed in an arbitrary order, the number k of subtrees could be up to linear in n. The solution is to visit *heavy* subtrees first, i.e. if a node is visited for the first time, the next step moves to the larger subtree of the current node (if both subtrees have the same size, one moves to the left subtree). Note that the size of a subtree can be computed in logspace. This ensures that $|t_i| \geq |t_{i+1}| + \cdots + |t_k|$ and therefore $|t_i| \geq 2|t_{i+2}|$ for $i \leq k-2$. In particular, k is bounded by $\mathcal{O}(\log n)$ and the total space complexity is $\mathcal{O}(k \log\log n) = \mathcal{O}(\log n \log\log n)$.

To shave off the $\log\log n$ factor, we store the sequence of Strahler numbers $s_1, \dots, s_k$ using a delta encoding. Let us say that a number s_i is *dominated* if there exists $j > i$ such that $s_j > s_i$. Such a dominated number s_i can be replaced by 0, which does not change the Strahler number of the tree. The subsequence of undominated numbers $s_{i_1}, s_{i_2}, \dots, s_{i_\ell}$ is monotonically decreasing and can be encoded by its delta encoding $s_{i_1} - s_{i_2}, \dots, s_{i_{\ell-1}} - s_{i_\ell}, s_{i_\ell}$. Each number s in this sequence will be represented in unary encoding by $1^s\#$. As an example, the sequence $s_1, \dots, s_k = 3, 2, 5, 3, 4, 4, 2, 1$ is encoded by $001\#0\#11\#1\#1\#$. The resulting word over the alphabet $\{0, 1, \#\}$ has length $k + s_{i_1} \leq \mathcal{O}(\log n)$.

Now we turn to proving the uNC^1 bound from Theorem 3.1. To this end, we will first compute from the input tree t a TSLP using Theorem 2.4. To make use of the TSLP-representation of t, we need a simple description of unary linear term functions in the Strahler algebra. For this, we start with some preparations.

Consider a TSLP $\mathcal{G} = (N_0, N_1, S, \rho)$ as defined in Section 2.3. For a tree variable $A \in N_0$ we write st_A for the Strahler number $\mathsf{st}(\mathsf{val}(A))$. For a context variable $B \in N_1$ we define a function $\mathsf{st}_B : \mathbb{N} \to \mathbb{N}$ as follows: Consider the binary context $t = \mathsf{val}(B)$. Take an integer $n \in \mathbb{N}$ and take a binary tree t' with $\mathsf{st}(t') = n$. The concrete choice of t' is not important. Then we define $\mathsf{st}_B(n) = \mathsf{st}(t[x/t'])$. Intuitively speaking, the evaluation of the binary context t in the Strahler algebra yields the unary linear term function st_B. If we substitute the placeholder x by a number n then we can evaluate the resulting expression in the Strahler algebra and the result is $\mathsf{st}_B(n)$. The functions st_B can be described by two integers in the following way: For $\ell, h \in \mathbb{N}$ with $0 \le \ell \le h$ we define the function $[\ell, h] : \mathbb{N} \to \mathbb{N}$ as follows:

$$[\ell,h](x) = \begin{cases} h & \text{if } x < \ell \\ h+1 & \text{if } \ell \le x \le h \\ x & \text{if } x > h \end{cases} \tag{3}$$

▶ **Lemma 3.2.** *The functions $[\ell, h]$ are closed under composition. More precisely, for all $\ell \le h$, $m \le i$ and $x \in \mathbb{N}$ we have the following:*

$$[m,i]([\ell,h](x)) = \begin{cases} [m,i](x) & \text{if } h+2 \le m \\ [\ell,i](x) & \text{if } h+1 = m \\ [0,i](x) & \text{if } m \le h \le i \\ [\ell,h](x) & \text{if } i < h \end{cases} \tag{4}$$

Proof. For all $x \in \mathbb{N}$ we have:

$$[m,i]([\ell,h](x)) = \begin{cases} [m,i](h) & \text{if } x < \ell \\ [m,i](h+1) & \text{if } \ell \le x \le h \\ [m,i](x) & \text{if } x > h \end{cases} \tag{5}$$

We now distinguish the four cases from (4):

Case 1: $h + 2 \le m$. We have to show that $[m,i]([\ell,h](x)) = [m,i](x)$ for all x. Since $h + 1 < m$ we obtain from (5):

$$[m,i]([\ell,h](x)) = \begin{cases} i = [m,i](x) & \text{if } x \le h \\ [m,i](x) & \text{if } x > h \end{cases}$$

Case 2: $h + 1 = m$. We have to show $[m,i]([\ell,h](x)) = [\ell,i](x)$ for all x. With (5) we get

$$[m,i]([\ell,h](x)) \;=\; \left.\begin{cases} [h+1,i](h) = i & \text{if } x < \ell \\ [h+1,i](h+1) = i+1 & \text{if } \ell \le x \le h \\ [h+1,i](x) = i+1 & \text{if } h < x \le i \\ [h+1,i](x) = x & \text{if } x > i \end{cases}\right\} = [\ell,i](x).$$

Case 3: $m \le h \le i$. We have to show $[m,i]([\ell,h](x)) = [0,i](x)$ for all x. Equation (5) yields

$$[m,i]([\ell,h](x)) = \begin{cases} i+1 = [0,i](x) & \text{if } x \le i, \\ x = [0,i](x) & \text{if } x > i. \end{cases}$$

Case 4: $i < h$. We have to show that $[m,i]([\ell,h](x)) = [\ell,h](x)$ for all x. Equation (5) simplifies for $i < h$ to

$$[m,i]([\ell,h](x)) = \left.\begin{cases} h & \text{if } x < \ell \\ h+1 & \text{if } \ell \leq x \leq h \\ x & \text{if } x > h \end{cases}\right\} = [\ell,h](x).$$

This concludes the proof of the lemma. ◀

▶ **Lemma 3.3.** *Let $\mathcal{G} = (N_0, N_1, S, \rho)$ be a TSLP and let $A \in N_1$. Then there exist numbers ℓ_A and h_A such that $\mathsf{st}_A = [\ell_A, h_A]$.*

Proof. Every context $\mathsf{val}(A)$ for $A \in N_1$ can be obtained from composing contexts of the form $b(x,t)$ and $b(t,x)$ where $t = \mathsf{val}(B)$ for some $B \in N_0$. By Lemma 3.2 it therefore suffices to show that for every $m \in \mathbb{N}$ the mapping $x \mapsto s(x,m)$ (where s is from (2)) is of the form $[\ell,h]$ (then, since s is commutative, the same holds for the mapping $x \mapsto s(m,x)$). It is straightforward to check that $s(x,m) = [m,m](x)$ for all $x \in \mathbb{N}$. ◀

We are now in the position to prove Theorem 3.1.

Proof of Theorem 3.1. We start with the uNC^1 upper bound. Let $t \in \mathsf{Bin}$ be a binary tree given in term representation. Let n be the number of leaves of t. Hence, we must have $\mathsf{st}(t) \leq \log_2 n$. Our goal is to compute in uTC^0 from t and an integer $0 \leq k \leq \log_2 n$ a Boolean circuit $\mathcal{B}_{t,k}$ of depth $\mathcal{O}(\log n)$ such that $\mathcal{B}_{t,k}$ evaluates to true if and only if $\mathsf{st}(t) \geq k$. The Boolean circuit $\mathcal{B}_{t,k}$ is represented in ec-representation, which ensures that it can be unfolded in uTC^0 into an equivalent Boolean formula (Lemma 2.2) and then evaluated in uNC^1 by Theorem 2.1.

In a first step, we use Theorem 2.4 to compute in uTC^0 the ec-representation of a TSLP $\mathcal{G} = (N_0, N_1, S, \rho)$ of depth $\mathcal{O}(\log n)$ and size $\mathcal{O}(n)$ such that $\mathsf{val}(\mathcal{G}) = t$. Let $N = N_0 \cup N_1$. Since the ec-representation of $\mathcal{G}$ is available and the depth of $\mathcal{G}$ is bounded by $\mathcal{O}(\log n)$, one can ensure in uTC^0 that all variables in N can be reached from the start variable S (this property is actually satisfied when $\mathcal{G}$ is constructed according to [28]). In particular, every variable $A \in N_0$ produces a subtree of t and every variable $A \in N_1$ produces a subcontext of t. Hence, every number st_A is bounded by $\log_2 n$ and we will see in a moment that the same holds for the numbers ℓ_A and h_A from Lemma 3.3.

In the following we consider the following set of formal integer variables:

$$\Delta(\mathcal{G}) = \{A_{\mathsf{st}} : A \in N_0\} \cup \{\, A_\ell, A_h : A \in N_1\}.$$

For $X \in \Delta(\mathcal{G})$ we define the integer $v(X)$ by $v(A_{\mathsf{st}}) = \mathsf{st}_A$, $v(A_\ell) = \ell_A$ and $v(A_h) = h_A$.

We will define a Boolean circuit $\mathcal{B}_{t,k}$ that contains for all $i \in \mathbb{Z}$ with $|i| \leq \log_2 n$ and all $X, Y \in \Delta(\mathcal{G})$ a gate $[X \leq Y + i]$ with the obvious meaning: the gate evaluates to true if and only if $v(X) \leq v(Y) + i$. Let V be the set of all such gates $[X \leq Y + i]$. It is convenient to allow also gates $[X \leq i]$ and $[X \geq i]$. Formally, they can be replaced by $[X \leq A_{\mathsf{st}} + i]$ and $[A_{\mathsf{st}} \leq X - i]$, where $A \in N_0$ is a variable with $\rho(A) = a$ (so that $\mathsf{st}_A = 0$). The output gate of $\mathcal{B}_{t,k}$ is $[S_{\mathsf{st}} \geq k]$.

The number i is called the *offset* of the gate $g = [X \leq Y + i]$ and we define $N(g) = \{A, B\}$ if $X \in \{A_{\mathsf{st}}, A_\ell, A_h\}$ and $Y \in \{B_{\mathsf{st}}, B_\ell, B_h\}$. For gates $g_1, g_2 \in V$, we write $g_1 \succ g_2$ if every variable $B \in N(g_2)$ appears in $\rho(A)$ for some $A \in N(g_1)$. Then the length of every chain $g_1 \succ g_2 \succ g_3 \succ \cdots \succ g_m$ is bounded by the depth of $\mathcal{G}$, which is $\mathcal{O}(\log n)$.

To define the wires of $\mathcal{B}_{t,k}$, first note that the numbers $v(X)$ $(X \in \Delta(\mathcal{G}))$ are computed according to the following rules:

(i) if $A \in N_0$ and $\rho(A) = a$ then $\mathsf{st}_A = 0$,

(ii) if $A \in N_0$ and $\rho(A) = b(B, C)$ then

$$\mathsf{st}_A = \begin{cases} \mathsf{st}_B & \text{if } \mathsf{st}_B > \mathsf{st}_C, \\ \mathsf{st}_C & \text{if } \mathsf{st}_B < \mathsf{st}_C, \\ \mathsf{st}_B + 1 & \text{if } \mathsf{st}_B = \mathsf{st}_C, \end{cases}$$

(iii) if $A \in N_0$ and $\rho(A) = B(C)$ then, since $\mathsf{st}_B = [\ell_B, h_B]$,

$$\mathsf{st}_A = \begin{cases} h_B & \text{if } \mathsf{st}_C < \ell_B, \\ h_B + 1 & \text{if } \ell_B \le \mathsf{st}_C \le h_B, \\ \mathsf{st}_C & \text{if } \mathsf{st}_C > h_B, \end{cases}$$

(iv) if $A \in N_1$ and $\rho(A) = b(x, B)$ or $\rho(A) = b(B, x)$ then $\ell_A = h_A = \mathsf{st}_B$, and

(v) if $A \in N_1$ and $\rho(A) = B(C(x))$ then by Lemma 3.2 we have:

$$\ell_A = \begin{cases} \ell_B & \text{if } h_C + 2 \le \ell_B, \\ \ell_C & \text{if } h_C + 1 = \ell_B, \\ 0 & \text{if } \ell_B \le h_C \le h_B, \\ \ell_C & \text{if } h_B < h_C, \end{cases} \qquad h_A = \begin{cases} h_B & \text{if } h_C + 2 \le \ell_B, \\ h_B & \text{if } h_C + 1 = \ell_B, \\ h_B & \text{if } \ell_B \le h_C \le h_B, \\ h_C & \text{if } h_B < h_C. \end{cases} \tag{6}$$

Points (iv) and (v) imply that all numbers ℓ_A and h_A for $A \in N_1$ are equal to some st_B ($B \in N_0$) and therefore bounded by $\log_2 n$.

From the equalities in (i)–(v), it is now straightforward to construct for every gate $g \in V$ a Boolean circuit $\mathcal{B}_g$ of constant size with output gate g. All input gates g' of $\mathcal{B}_g$ satisfy $g \succ g'$. Let us consider for instance the gate $[A_h \le D_\ell + i]$ for $A, D \in N_1$ and assume that $\rho(A) = B(C(x))$ and $\rho(D) = b(x, E)$. Then, $\ell_D = \mathsf{st}_E$ and the equation for h_A in (6) implies

$$h_A \le \ell_D + i \iff (h_B \le \mathsf{st}_E + i \wedge h_C \le h_B) \vee (h_C \le \mathsf{st}_E + i \wedge h_B < h_C).$$

This equivalence directly yields the Boolean circuit for $[A_h \le D_\ell + i]$. Its input gates are: $[B_h \le E_{\mathsf{st}} + i]$, $[C_h \le B_h]$, $[C_h \le E_{\mathsf{st}} + i]$, and $[B_h \le C_h - 1]$.

When constructing a Boolean circuit $\mathcal{B}_g$ one may obtain gates, where the absolute value of the offset i is larger than $\log_2 n$. Such gates can be replaced by true or false: $[X \le Y + i]$ with $i > \log_2 n$ can be replaced by true (since $v(X) \le \log_2 n$ and $v(Y) \ge 0$) and $[X \le Y - i]$ with $i > \log_2 n$ can be replaced by false.

The circuit $\mathcal{B}_{t,k}$ results from the union of the above constant-size circuits. Since the depth of $\mathcal{G}$ is bounded by $\mathcal{O}(\log n)$, it follows that the depth of $\mathcal{B}$ is also bounded by $\mathcal{O}(\log n)$. Moreover, the ec-representation of $\mathcal{B}$ can be easily computed in uTC^0 from the ec-representation of the TSLP $\mathcal{G}$. This shows the upper bound from Theorem 3.1.

For the lower bound we give a reduction from the uNC^1-complete Boolean formula value problem; see Theorem 2.1. Binary conjunction $\wedge$ is simulated by the operation

$$f_\wedge(x, y) = s(x + 1, y + 1) = s(s(x, x), s(y, y)), \tag{7}$$

where s from (2), and binary disjunction $\vee$ is simulated by the operation

$$f_\vee(x, y) = s(s(x + 1, y), s(x, y + 1)) = s(s(s(x, x), y), s(x, s(y, y))). \tag{8}$$

We obtain for every $a \geq 0$:

$$f_\wedge(a,a) = f_\wedge(a,a+1) = f_\wedge(a+1,a) = a+2 \text{ and } f_\wedge(a+1,a+1) = a+3, \tag{9}$$
$$f_\vee(a,a) = a+2 \text{ and } f_\wedge(a,a+1) = f_\wedge(a+1,a) = f_\wedge(a+1,a+1) = a+3. \tag{10}$$

A given Boolean formula (built from binary operators $\wedge$ and $\vee$; negation is not needed in [9]) can be transformed in uTC^0 into an equivalent Boolean formula Φ of depth $d \leq \mathcal{O}(\log|\Phi|)$; see [28]. We can also assume that every path from the root to a leaf has the same length d. By replacing in Φ every $\wedge$ (resp., $\vee$) by $f_\wedge$ (resp., $f_\vee$) and replacing every occurrence of the truth value true (resp., false) by $1 = s(0,0)$ (resp., 0), we obtain an expression that evaluates in the Strahler algebra to $2d+1$ (resp., $2d$) if the Boolean formula Φ evaluates to true (resp., false). Note that replacing $f_\wedge(x,y)$ and $f_\vee(x,y)$ by their left-hand sides from (7) and (8) yields a DAG (since x and y appear more than once in (7) and (8)) whose ec-representation can be computed in uTC^0 from Φ. Since the depth of this DAG is $\mathcal{O}(\log|\Phi|)$ it can be unfolded into the term representation of a tree in uTC^0 by Lemma 2.2. ◀

For $\mathsf{St}^{\geq k}_{\mathsf{term}}$ with $k \geq 4$ we can show uTC^0-completeness via a reduction from $\mathsf{Majority}$:

▶ **Theorem 3.4.** *The problem* $\mathsf{St}^{\geq k}_{\mathsf{term}}$ *is* uTC^0*-complete for every* $k \geq 4$*. In particular, there is a* uAC^0*-computable function* $t : \{0,1\}^* \to \mathsf{Bin}$ *such that the following holds for every* $w \in \{0,1\}^*$*: if* $w \in \mathsf{Majority}$ *then* $\mathsf{st}(t(w)) = 4$*, otherwise* $\mathsf{st}(t(w)) = 3$*.*

Proof of Theorem 3.4. We first show that every problem $\mathsf{St}^{\geq k}_{\mathsf{term}}$ (for a fixed k) is in uTC^0. Let $t \in \mathsf{Bin}$. We have to check whether the complete binary tree t_k of depth k embeds into t. For this, we have to check whether there exist $2^{k+1}-1$ different positions in t such that the corresponding nodes are in the correct descendant relations in order to yield an embedding of t_k. Hence, it suffices to show that in uTC^0 one can check whether for two positions $i < j$ in t (that are identified with the corresponding tree nodes), j is a proper descendant of i. This holds if and only if there exists a position $k \geq j$ in t such that $t[i,k]$ (the substring of t starting in position i and ending in position k) belongs to Bin. Since Bin belongs to uTC^0 [43], this can be checked in uTC^0 as well.

For the hardness part it suffices to consider the case $k = 4$. We start with the morphism $d : \{0,1\}^* \to \{0,1\}^*$ that doubles each symbol: $d(0) = 00$ and $d(1) = 11$. Clearly, $w \in \mathsf{Majority}$ if and only if $d(w) \in \mathsf{Majority}$ and $d(w)$ can be computed in uAC^0. Hence it suffices to consider strings in the image of d ($\mathsf{im}(d)$ for short) in the following. Every $w \in \mathsf{im}(d)$ can be uniquely factorized as $w = 0^{2k_0}10^{2k_1}10^{2k_2}\cdots 10^{2k_m}$, where $m = |w|_1 \geq 0$ (which is even) and $k_1,\ldots,k_m \geq 0$ (actually, every second k_i is zero due to the factors 11 but we do not need this fact in the following). Then we define

$$f(w) \;=\; b^{k_0}a^{k_0}b^{1+k_1}a^{k_1}b^{1+k_2}a^{k_2}\cdots b^{1+k_m}a^{k_m}.$$

In other words: every 1 in w is replaced by b and every maximal block 0^{2k} of zeros in w is replaced by $b^k a^k$.

▷ Claim 3.5. The function f can be computed in uAC^0.

Proof. To see this, note that $f(w)$ is obtained from w by replacing every 1 by b and every 0 by either b or a according to the following rule: Assume that an occurrence of 0 is the i^{th} 0 in a maximal block of $2k$ zeros. Then this occurrence of 0 is replaced by b if $i \leq k$ and otherwise by a. This case distinction can be easily implemented by a bounded depth Boolean circuit of unbounded fan-in. ◁

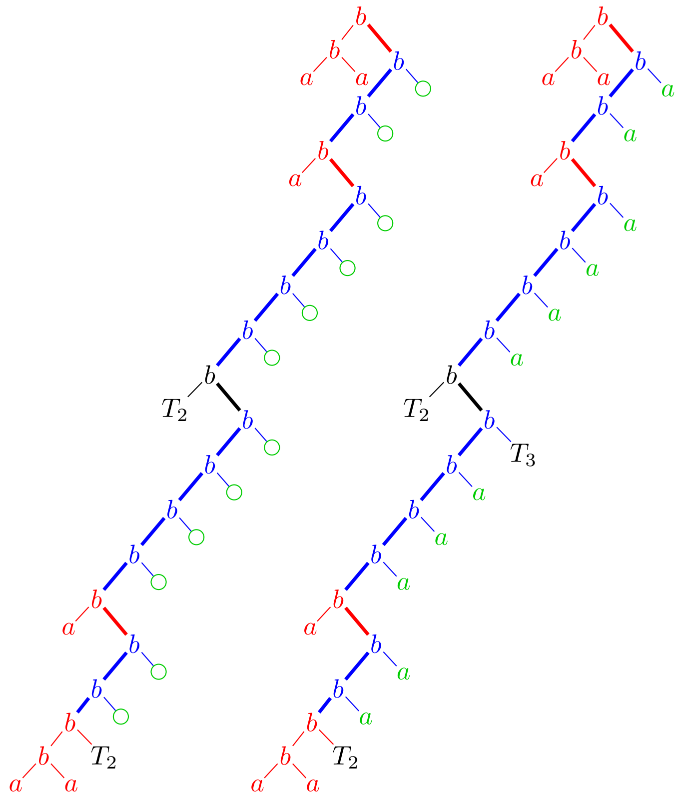

Figure 2 The tree fragment $f(w)\,b\,T_2\,f(\bar{w})\,T_2$ for $w = 000011001111$ on the left and $t(w)$ on the right.

Finally, for a bit string $w \in \mathsf{im}(d)$ of length $2n$ we define

$$t(w) \;=\; f(w)\,b\,T_2\,f(\bar{w})\,T_2\,a^{n-1}\,T_3\,a^n,$$

where $\bar{w} \in \mathsf{im}(d)$ is obtained by flipping every bit in w, $T_2 = bbaabaa$ (the term representation of a tree of Strahler number 2) and $T_3 = bT_2T_2$ (the term representation of a tree of Strahler number 3). Since f can be computed in uAC^0, the same holds for t.

The string $t(w)$ is the term representation of a binary tree and satisfies the following:

▷ Claim 3.6. If $|w|_0 \geq n$ then $\mathsf{st}(t(w)) = 3$, otherwise $\mathsf{st}(t(w)) = 4$.

Proof. Let us look at the example where $w = d(001011) = 000011001111$ has length $2n = 12$, which satisfies $|w|_0 \geq n = 6$. On the right of Figure 2, the tree

$$t(w) = f(w)\,b\,T_2\,f(\bar{w})\,T_2\,a^{n-1}\,T_3\,a^n = bbaa\,bb\,ba\,bbbb\,b\,T_2\,bbbb\,ba\,bb\,bbaa\,T_2\,a^5\,T_3\,a^6$$

is shown. Note that a string $b^k a^k$ produces a caterpillar tree of depth k branching off from the root to the left and leaving a "hole" at the position right below the root. These are the red patterns in Figure 2. The b's (replacing the 1's when applying f) yield the blue nodes and edges in Figure 2.

Figure 2 (left) shows the fragment of $t(w)$ that is produced by the prefix $f(w)\,b\,T_2\,f(\bar{w})\,T_2$. The green circles represent holes. The first $|w|_1$ holes are produced by $f(w)$ followed by $|w|_0$ holes produced by $b\,T_2\,f(\bar{w})\,T_2$ (note that $|w|_0 + |w|_1 = 2n$). These $2n$ holes are then filled bottom-up by the $2n$ trees from the suffix $a^{n-1}\,T_3\,a^n$, which finally yields $t(w)$. All $2n$ holes are filled with a except the n^{th} hole from bottom, which is filled by T_3. Note that the tree $t(w)$ has a main spine that is highlighted by the thick edges in Figure 2. The nodes on this spine are called the spine nodes below. All subtrees that are attached to the spine have Strahler number at most 2 except for the unique occurrence of T_3. The crucial observation now is the following:

- If $|w|_0 \geq n$ then T_3 is attached to a spine node that is below the spine node to which the upper occurrence of T_2 is attached (this is the case in Figure 2). This implies $\mathsf{st}(t(w)) = 3$.
- If $|w|_0 < n$ then T_3 is attached to a spine node that is above the spine node to which the upper occurrence of T_2 is attached. This implies that $\mathsf{st}(t(w)) = 4$.

This proves Claim 3.6. ◁

Claim 3.6 implies the second statement of the theorem: if $w \in \mathsf{Majority}$ then $|w|_1 > n$, i.e., $|w|_0 = 2n - |w|_1 < n$, and Claim 3.6 gives $\mathsf{st}(t(w)) = 4$. Similarly, if $|w|_1 \leq n$ then $|w|_0 \geq n$ and Claim 3.6 gives $\mathsf{st}(t(w)) = 3$. ◀

It is easy to see that the problem $\mathsf{St}^{\geq 2}_{\mathsf{term}}$ belongs to uAC^0: if $t \in \mathsf{Bin}$ then $\mathsf{st}(t) \geq 2$ if and only if the string t contains at least two occurrences of baa, which can be tested in uAC^0. We do not know whether $\mathsf{St}^{\geq 3}_{\mathsf{term}}$ still belongs to uAC^0.

For input trees given in pointer representation, we show the following results in [29]:

▶ **Theorem 3.7.** *$\mathsf{St}^{\geq}_{\mathsf{pointer}}$ and $\mathsf{St}^{\geq k}_{\mathsf{pointer}}$ for every $k \geq 3$ are L-complete.*

Finally, we also considered the cases where the input tree is given in a compressed form by either a binary DAG or a TSLP. In the long version [29] we show the following:

▶ **Theorem 3.8.** *The following hold:*

(i) *$\mathsf{St}^{\geq}_{\mathsf{dag}}$ and $\mathsf{St}^{\geq}_{\mathsf{tslp}}$ are P-complete.*

(ii) *For every fixed $k \geq 1$, $\mathsf{St}^{\geq k}_{\mathsf{dag}}$ is in $\mathsf{UL} \cap \mathsf{coUL}$.*

(iii) *For every fixed $k \geq 2$, $\mathsf{St}^{\geq k}_{\mathsf{tslp}}$ is NL-complete.*

The best lower bound for the problem in Theorem 3.8(ii) that we are aware of is L-hardness (for $k \geq 3$), coming from Theorem 3.7.

4 Strahler number of derivation trees

In this section, we briefly report on our results for the Strahler numbers of derivation trees of context-free grammars. Since we want to obtain binary trees and since the derived words have no relevance for us, we consider context-free grammars $G = (N, S, P)$, where N is the set of nonterminals, $S \in N$ is the start nonterminal and P is the set of productions such that each of them has the form $A \to \varepsilon$ or $A \to BC$ for $A, B, C \in N$. Slightly abusing standard terminology, we call such a grammar a *Chomsky normal form grammar* or CNF-grammar for short. The notion of a derivation tree is defined as usual: a *derivation tree for $A \in N$* is an N-labelled binary tree such that (i) the root is labelled with A, (ii) if an internal node v is labelled with $B \in N$ then there is a production $B \to CD$ such that the left (resp., right) child of v is labelled with C (resp., D) and (iii) if v is a B-labelled leaf then $(B \to \varepsilon) \in P$. A *derivation tree of G* is a derivation for the start nonterminal S. A derivation tree t is called acyclic if there is no nonterminal that appears twice along a path from the root to a leaf. The motivation for considering acyclic derivations trees and their Strahler numbers comes from [45]; see the discussion in the introduction. In this section we consider the following problem $\mathsf{CNF}^{\geq}$ (resp., $\mathsf{acCNF}^{\geq}$):

- Input: a CNF-grammar G and a number k (given in unary encoding).
- Question: Is there a derivation tree (resp., acyclic derivation tree) t of G with $\mathsf{st}(t) \geq k$?

If the number k is fixed and not part of the input, we obtain the problems $\mathsf{CNF}^{\geq k}$ and $\mathsf{acCNF}^{\geq k}$. The following results pinpoint the complexity of these problems:

▶ **Theorem 4.1.** *The following holds:*
(i) $\mathsf{CNF}^{\geq}$ *and* $\mathsf{CNF}^{\geq k}$ *for every* $k \geq 1$ *are* P*-complete.*
(ii) $\mathsf{acCNF}^{\geq k}$ *is* NP*-complete for every* $k \geq 2$*.*
(iii) $\mathsf{acCNF}^{\geq}$ *is* PSPACE*-complete.*

The full proof of Theorem 4.1 can be found in [29]. Here, we only prove statement (ii).

Proof of Theorem 4.1(ii). Membership in NP was shown in [45]. It suffices to show NP-hardness for $k = 2$. For this, we present a reduction from *exact 3-hitting set* (X3HS):
- Input: a finite set M and a non-empty set $\mathcal{B} \subseteq 2^M$ of subsets of M, all of size 3.
- Question: Is there a subset $S \subseteq M$ such that $|S \cap C| = 1$ for all $C \in \mathcal{B}$?

X3HS is the same problem as positive 1-in-3-SAT, which is NP-complete [31, Problem LO4].

Fix the set M and a subset $\mathcal{B} \subseteq 2^M$ with all $C \in \mathcal{B}$ of size 3. W.l.o.g. assume that $M = \{1, 2, \dots, n\}$ and fix an arbitrary ordering $C_1, C_2, \dots, C_m$ of the subsets in $\mathcal{B}$. We will construct a CNF-grammar G such that there is a derivation tree of G with Strahler number at least two if and only if there is a subset $S \subseteq \{1, \dots, n\}$ such that $|S \cap C| = 1$ for all $C \in \mathcal{B}$.

In order to make the grammar more readable we use the following notation below. If we write in a right-hand side $[AB]$ for nonterminals A and B, then $[AB]$ is another nonterminal with the unique production $[AB] \to AB$ and this production is not explicitly listed. Moreover, this notation will be nested, i.e., A and B can be also of the form $[CD]$. In Figure 3 such nonterminals are depicted as filled circles. With this notation, the productions of our CNF-grammar G are as follows:

$$
\begin{aligned}
E &\to \varepsilon \\
A_k &\to I_k E \mid O_k E \text{ whenever } 1 \leq k \leq n \\
I_k &\to A_{k+1} E \mid \varepsilon \text{ whenever } 1 \leq k \leq n-1 \\
I_n &\to B_1 E \mid \varepsilon \\
O_k &\to A_{k+1} E \mid \varepsilon \text{ whenever } 1 \leq k \leq n-1 \\
O_n &\to B_1 E \mid \varepsilon \\
B_j &\to B_{j+1}[[O_a[I_b I_c]] \text{ whenever } 1 \leq j \leq m \text{ and } C_j = \{a, b, c\} \\
B_{m+1} &\to EE
\end{aligned}
$$

The start nonterminal is A_1. Note that there are six productions of the form $B_j \to B_{j+1}[[O_a[I_b I_c]]$ corresponding to the six permutations of the set $C_j = \{a, b, c\}$ (we could restrict to three productions since the order between I_b and I_c is not important for the following arguments).

Consider first an acyclic derivation tree t rooted in A_1 with Strahler number at least two. The top part of every derivation tree rooted in A_1 must have one of the two shapes shown in Figure 3 (left and middle tree), where $X_k \in \{I_k, O_k\}$. A left tree is a complete (acyclic) derivation tree with Strahler number 1. Hence, the top part of t must have the middle shape from Figure 3. It defines the subset $S = \{k : 1 \leq k \leq n, X_k = I_k\}$. From the leaf B_1 we have to expand the derivation tree t further and this results in a bottom part tree as shown in Figure 3 (right tree), where for every $1 \leq j \leq m$ we have $C_j = \{x_j, y_j, z_j\}$. Since the tree t (obtained by merging the top part from Figure 3 (middle tree) with the bottom part from Figure 3 (right tree), where the merging is done by identifying the B_1-labelled nodes) is an acyclic derivation tree we must have $x_m \in S$, $y_m \notin S$, and $z_m \notin S$ for every $j \in \{1, \dots, m\}$. Therefore, our X3HS-instance is positive.

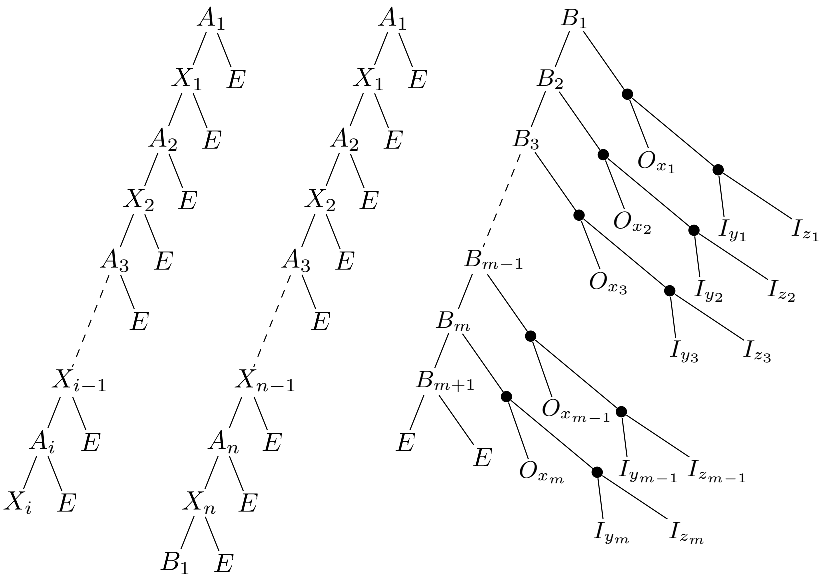

Figure 3 An acyclic derivation tree for the grammar G (proof of Theorem 4.1(ii)) has either the form shown on the left, or it results from merging the tree shown in the middle with the tree shown on the right in the B_1-labelled node. Every X_k is either I_k or O_k.

Vice versa, if there is a subset $S \subseteq \{1, \dots, n\}$ such that $|S \cap C| = 1$ for every $C \in \mathcal{B}$, then we obtain an acyclic derivation tree t with Strahler number two by merging the middle and right tree from Figure 3, where we set $X_k = I_k$ if $k \in S$ and $X_k = O_k$ if $k \notin S$ in the middle tree. Moreover, if $C_j = \{a, b, c\}$ with $\{a\} = S \cap C$ then we set $x_j = a$, $y_j = b$ and $z_j = c$ (or $y_j = c$ and $z_j = b$) in the right tree. ◀

The P-hardness of $\mathsf{CNF}^{\geq k}$ in Theorem 4.1 comes solely from the fact that emptiness for CNF-grammars is P-hard. One can avoid this difficulty by adding to the input grammar a certificate, ensuring that all nonterminals are *productive*. Formally, this certificate is defined in [29], where it is shown that the problem $\mathsf{CNF}^{\geq k}$ for $k \geq 2$ becomes NL-complete when such a certificate is part of the input, whereas $\mathsf{CNF}^{\geq}$ stays P-complete.

5 Open problems

We conclude the paper with some open problems.

Computing Strahler numbers for unranked trees. Strahler numbers have been also defined for unranked trees (trees, where nodes can have any number of children): Consider an unranked tree t, where $t_1, \dots, t_k$ ($k \geq 0$) are the subtrees rooted in the children of the root of t. We define the Strahler number $\mathsf{st}(t)$ of t inductively as follows: if $k = 0$ then $\mathsf{st}(t) = 0$. If $k \geq 1$ then let $n_i = \mathsf{st}(t_i)$ and $n = \max\{n_1, \dots, n_k\}$. If n has a unique occurrence in the list $n_1, \dots, n_k$, then $\mathsf{st}(t) = n$, otherwise $\mathsf{st}(t) = n + 1$. We conjecture that our NC^1-algorithm for computing the Strahler number of a binary tree can be adapted to unranked trees, but this seems to be not obvious. Tree straight-line programs could be replaced by forest straight-line programs [32] that work for unranked trees. For this one has to prove a variant of Theorem 2.4 for forest straight-line programs. In addition, one needs a variant of Lemma 3.2 for unranked trees, which is not obvious.

Expression evaluation for the max-plus semiring. Closely related but slightly different to the computation of Strahler numbers is the problem of evaluating expressions over the max-plus semiring $(\mathbb{N}, \max, +)$. If the input numbers are given in unary encoding, then the problem is logspace reducible to the evaluation of arithmetic expressions over $(\mathbb{N}, +, \times)$ and hence belongs to L [38]. The complexity is open if the input numbers are encoded in binary. Let us also mention that the longest (or shortest [38, Lemma 3.3]) path problem in a directed graph with binary encoded weights is in AC^1, using matrix powering over the max-plus semiring, but it is a longstanding open problem whether it lies in a smaller complexity class [14, p. 13], see also [30, 50] for recent applications.

Computing path width of trees. A problem that is related to the computation of the Strahler number of a tree t is the computation of the path width of an undirected tree. Let us define $\mathsf{st}(t)$ for an undirected tree t as the minimal Strahler number of a rooted tree that can be obtained by choosing a root in t. Then the following relationship is stated in [25]: $\mathsf{pathwidth}(t) - 1 \leq \mathsf{st}(t) \leq 2 \cdot \mathsf{pathwidth}(t)$. It is shown in [49] that the path width of an undirected tree t can be computed in linear time. It would be interesting to know whether it can be computed in logspace.

References

1 Alex Arenas, Leon Danon, Albert Díaz-Guilera, Pablo M. Gleiser, and Roger Guimerá. Community analysis in social networks. *European Physical Journal B*, 38(2):373–380, 2004. doi:10.1140/epjb/e2004-00130-1.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. doi:10.1017/CBO9780511804090.

3 Mohamed Faouzi Atig and Pierre Ganty. Approximating Petri net reachability along context-free traces. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011*, volume 13 of *LIPIcs*, pages 152–163. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.152.

4 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990. doi:10.1016/0022-0000(90)90022-D.

5 Martin Beaudry and Pierre McKenzie. Circuits, matrices, and nonassociative computation. *Journal of Computer and System Sciences*, 50(3):441–455, 1995. doi:10.1006/JCSS.1995.1035.

6 Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992. doi:10.1137/0221006.

7 Clotilde Bizière and Wojciech Czerwinski. Reachability in one-dimensional pushdown vector addition systems is decidable. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 1851–1862. ACM, 2025. doi:10.1145/3717823.3718149.

8 Richard P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, 1974. doi:10.1145/321812.321815.

9 Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proceedings of the 19th Annual Symposium on Theory of Computing, STOC 1987*, pages 123–131. ACM Press, 1987. doi:10.1145/28395.28409.

10 Samuel R. Buss, Stephen A. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21(4):755–780, 1992. doi:10.1137/0221046.

11 Andrew Chiu, George I. Davida, and Bruce E. Litow. Division in logspace-uniform NC^1. *RAIRO – Theoretical Informatics and Applications*, 35(3):259–275, 2001. doi:10.1051/ITA:2001119.

12 Michal P. Chytil and Burkhard Monien. Caterpillars and context-free languages. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1990*, volume 415 of *Lecture Notes in Computer Science*, pages 70–81. Springer, 1990. doi:10.1007/3-540-52282-4.

13 James Cook and Ian Mertz. Tree evaluation is in space $O(\log n \cdot \log \log n)$. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 1268–1278. ACM, 2024. doi:10.1145/3618260.3649664.

14 Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–21, 1985. doi:10.1016/S0019-9958(85)80041-3.

15 Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 193–242. Elsevier and MIT Press, 1990. doi:10.1016/B978-0-444-88074-1.50010-X.

16 Laure Daviaud, Marcin Jurdziński, and K. S. Thejaswini. The Strahler number of a parity game. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 123:1–123:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.123.

17 Luc Devroye and Paul Kruszewski. A note on the Horton-Strahler number for random trees. *Information Processing Letters*, 56(2):95–99, 1995. doi:10.1016/0020-0190(95)00114-R.

18 Luc Devroye and Paul Kruszewski. On the Horton-Strahler number for random tries. *RAIRO – Theoretical Informatics and Applications*, 30(5):443–456, 1996. doi:10.1051/ita/1996300504431.

19 Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4):758–771, 1980. doi:10.1145/322217.322228.

20 Andrzej Ehrenfeucht, Grzegorz Rozenberg, and Dirk Vermeir. On ETOL systems with finite tree-rank. *SIAM Journal on Computing*, 10(1):40–58, 1981. doi:10.1137/0210004.

21 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 143–152. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.21.

22 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012*, volume 14 of *LIPIcs*, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.66.

23 Andrey Petrovich Ershov. On programming of arithmetic operations. *Communications of the ACM*, 1(8):3–6, 1958. doi:10.1145/368892.368907.

24 Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Newtonian program analysis. *Journal of the ACM*, 57(6), 2010. doi:10.1145/1857914.1857917.

25 Javier Esparza, Michael Luttenberger, and Maximilian Schlund. A brief history of Strahler numbers. In *Proceedings of the 8th International Conference on Language and Automata Theory and Applications, LATA 2014*, volume 8370 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2014. doi:10.1007/978-3-319-04921-2_1.

26 Philippe Flajolet, Jean-Claude Raoult, and Jean Vuillemin. The number of registers required for evaluating arithmetic expressions. *Theoretical Computer Science*, 9(1):99–125, 1979. doi:10.1016/0304-3975(79)90009-4.

27 Philippe Flajolet, Paolo Sipala, and Jean-Marc Steyaert. Analytic variations on the common subexpression problem. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming, ICALP 1990*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 1990. doi:10.1007/BFB0032034.

28 Moses Ganardi and Markus Lohrey. A universal tree balancing theorem. *ACM Transactions on Computation Theory*, 11(1):1:1–1:25, 2019. doi:10.1145/3278158.

29 Moses Ganardi and Markus Lohrey. On the complexity of computing Strahler numbers, 2025. arXiv:2512.19060.

30 Moses Ganardi, Irmak Saglam, and Georg Zetzsche. Directed regular and context-free languages. In *Proceedings of the 41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024*, volume 289 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.STACS.2024.36.

31 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–completeness*. Freeman, 1979.

32 Adrià Gascón, Markus Lohrey, Sebastian Maneth, Carl Philipp Reh, and Kurt Sieber. Grammar-based compression of unranked trees. *Theory of Computing Systems*, 64(1):141–176, 2020. doi:10.1007/S00224-019-09942-Y.

33 Seymour Ginsburg and Edwin H. Spanier. Derivation-bounded languages. *Journal of Computer and System Sciences*, 2(3):228–250, 1968. doi:10.1016/S0022-0000(68)80009-1.

34 Jozef Gruska. A few remarks on the index of context-free grammars and languages. *Information and Control*, 19(3):216–223, 1971. doi:10.1016/S0019-9958(71)90095-7.

35 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.

36 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison–Wesley, Reading, MA, 1979.

37 Robert E. Horton. Erosional development of streams and their drainage basins: hydro-physical approach to quantitative morphology. *Geological Society of America Bulletin*, 56(3):275–370, 1945. doi:10.1130/0016-7606(1945)56[275:EDOSAT]2.0.CO;2.

38 Andreas Jakoby and Till Tantau. Computing shortest paths in series-parallel graphs in logarithmic space. In *Complexity of Boolean Functions*, volume 06111 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. doi:10.4230/DagSemProc.06111.6.

39 Rainer Kemp. The average number of registers needed to evaluate a binary tree optimally. *Acta Informatica*, 11(4):363–372, 1979. doi:10.1007/BF00289094.

40 S. Rao Kosaraju. On parallel evaluation of classes of circuits. In *Proceedings of the 10th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 472 of *Lecture Notes in Computer Science*, pages 232–237. Springer, 1990. doi:10.1007/3-540-53487-3_48.

41 Andreas Krebs, Nutan Limaye, and Michael Ludwig. A unified method for placing problems in polylogarithmic depth. In *Proceedings of the 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017*, volume 93 of *LIPIcs*, pages 36:36–36:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.36.

42 Paul Kruszewski. A note on the Horton-Strahler number for random binary search trees. *Information Processing Letters*, 69(1):47–51, 1999. doi:10.1016/S0020-0190(98)00192-6.

43 Markus Lohrey. On the parallel complexity of tree automata. In *Proceedings of the 12th International Conference on Rewrite Techniques and Applications, RTA 2001*, number 2051 in Lecture Notes in Computer Science, pages 201–215. Springer, 2001. doi:10.1007/3-540-45127-7_16.

44 Markus Lohrey. Grammar-based tree compression. In *Proceedings of the 19th International Conference on Developments in Language Theory, DLT 2015*, number 9168 in Lecture Notes in Computer Science, pages 46–57. Springer, 2015. doi:10.1007/978-3-319-21500-6_3.

45 Markus Lohrey, Andreas Rosowski, and Georg Zetzsche. Membership problems in finite groups. *Journal of Algebra*, 675:23–58, 2025. doi:10.1016/j.jalgebra.2025.03.011.

46 Carine Pivoteau, Bruno Salvy, and Michèle Soria. Algorithms for combinatorial structures: Well-founded systems and Newton iterations. *Journal of Combinatorial Theory, Series A*, 119(8):1711–1773, 2012. doi:10.1016/j.jcta.2012.05.007.

47 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000. doi:10.1137/S0097539798339041.

48 Walter L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22(3):365–383, 1981. doi:10.1016/0022-0000(81)90038-6.

49 Petra Scheffler. A linear algorithm for the pathwidth of trees. In Rainer Bodendiek and Rudolf Henn, editors, *Topics in Combinatorics and Graph Theory: Essays in Honour of Gerhard Ringel*, pages 613–620. Physica-Verlag HD, 1990. doi:10.1007/978-3-642-46908-4_70.

50 Yousef Shakiba, Henry Sinclair-Banks, and Georg Zetzsche. A complexity dichotomy for semilinear target sets in automata with one counter. In *Proceedings of the 40th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2025*, pages 594–608. IEEE, 2025. doi:10.1109/LICS65433.2025.00051.

51 Ekaterina Shemetova, Alexander Okhotin, and Semyon Grigorev. Rational index of languages defined by grammars with bounded dimension of parse trees. *Theory of Computing Systems*, 68(3):487–511, 2023. doi:10.1007/s00224-023-10159-3.

52 Philip M. Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proceedings of the 4th Hawaii Symposium on System Sciences*, pages 525–527, 1971.

53 Arthur N. Strahler. Hypsometric (area-altitude) analysis of erosional topology. *Geological Society of America Bulletin*, 63(11):1117–1142, 1952. doi:10.1130/0016-7606(1952)63[1117:HAAOET]2.0.CO;2.

54 Ivan Hal Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, 1978. doi:10.1145/322077.322083.

55 Joseph Swernofsky and Michael Wehar. On the complexity of intersecting regular, context-free, and tree languages. In *Proceedings of the 42nd International Colloquium Automata, Languages, and Programming, Part II, ICALP 2015*, volume 9135 of *Lecture Notes in Computer Science*, pages 414–426. Springer, 2015. doi:10.1007/978-3-662-47666-6_33.

56 Xavier Viennot. Trees. In *Mots, mélanges offert á M.P. Schützenberger*. Hermes, Paris, 1990. Available online at http://www.xavierviennot.org.

57 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. doi:10.1007/978-3-662-03927-4.

58 R. Ryan Williams. Simulating time with square-root space. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 13–23. ACM, 2025. doi:10.1145/3717823.3718225.