ENTROPY BOUNDS FOR GRAMMAR-BASED TREE COMPRESSORS

DANNY HUCKE, MARKUS LOHREY, AND LOUISA SEELBACH BENKNER

Abstract. The definition of k^{th}-order empirical entropy of strings is extended to node-labelled binary trees. A suitable binary encoding of tree straight-line programs (that have been used for grammar-based tree compression before) is shown to yield binary tree encodings of size bounded by the k^{th}-order empirical entropy plus some lower order terms. This generalizes recent results for grammar-based string compression to grammar-based tree compression.

Keywords. Grammar-based compression, binary trees, empirical entropy, lossless compression

1. Introduction

Grammar-based string compression. The idea of grammar-based compression is based on the fact that in many cases a word w can be succinctly represented by a context-free grammar that produces exactly w. Such a grammar is called a *straight-line program* (SLP) for w. In the best case, one gets an SLP of size $\Theta(\log n)$ for a word of length n, where the size of an SLP is the total length of all right-hand sides of the rules of the grammar. A grammar-based compressor is an algorithm that produces for a given word w an SLP $\mathcal{G}_w$ for w, where, of course, $\mathcal{G}_w$ should be smaller than w. Grammar-based compressors can be found at many places in the literature. Probably the best known example is the classical LZ78-compressor of Lempel and Ziv [39]. Indeed, it is straightforward to transform the LZ78-representation of a word w into an SLP for w. Other well-known grammar-based compressors are Bisection [23], Sequitur [32], and Repair [26], just to mention a few.

Recently, several upper bounds on the compression perfomance of grammar-based compressors in terms of higher order empirical entropy have been shown. For this, the choice of a concrete binary encoding $B(\mathcal{G})$ of an SLP $\mathcal{G}$ is crucial. Kieffer and Yang [22] came up with such a binary encoding B and proved that under certain assumptions on the grammar-based compressor $w \mapsto \mathcal{G}_w$, the combined compressor $w \mapsto B(\mathcal{G}_w)$ yields a universal code with respect to the family of finite-state information sources over finite alphabets. More precisely, it is needed that the size of the SLP $\mathcal{G}_w$ is bounded by $\mathcal{O}(|w|/\log_{\hat{\sigma}} |w|)$ where σ is the size of the underlying alphabet and $\hat{\sigma} = \max\{2, \sigma\}$. This upper bound is met by all grammar-based compressors that produce so-called irreducible SLPs [22], which is the case for e.g. LZ78, Bisection, and Repair after a small modification of the latter. In their recent paper [33], Navarro and Ochoa used the binary encoding $B(\mathcal{G}_w)$ from

A short version of this paper appeared in the Proceedings of ISIT 2019 [19].
This work has been supported by the DFG research project LO 748/10-1 (QUANT-KOMP).

[22] in order to prove for every word w over an alphabet of size σ the upper bound

$$|B(\mathcal{G}_w)| \leq |w|H_k(w) + o(|w| \log \hat{\sigma}) \tag{1}$$

for every $k \in o(\log_{\hat{\sigma}} |w|)$. Here, $H_k(w)$ is the k^{th}-order empirical entropy of w, and the grammar-based compressor $w \mapsto \mathcal{G}_w$ must satisfy the upper bound $|\mathcal{G}_w| \leq \mathcal{O}(|w|/\log_{\hat{\sigma}} |w|)$. Similar but weaker upper bounds for more practical binary SLP-encodings have been shown in [15, 31]. Finally, beyond grammar-based compression, universal encodings for strings have been proposed for example based on the Burrows-Wheeler transform [2, 7], Lempel–Ziv compressors [34], or the context-tree weighting method [37].

Grammar-based tree compression. Grammar-based compression has been generalized from strings to trees by means of linear context-free tree grammars generating exactly one tree [4]. Such grammars are also known as tree straight-line programs, TSLPs for short, see [28] for a survey. TSLPs can be seen as a proper generalization of SLPs and DAGs (directed acyclic graphs, which are a widely used compact representation of trees). Whereas DAGs only have the ability to share repeated subtrees of a tree, TSLPs can also share repeated tree patterns with a hole (so-called contexts). In [12], the authors presented a linear time algorithm that computes for a given binary tree t of size[1] n a TSLP $\mathcal{G}_t$ of size $\mathcal{O}(n/\log_{\hat{\sigma}} n)$ where σ is the size of the underlying set of node labels and $\hat{\sigma} = \max\{2, \sigma\}$. An alternative algorithm with the same asymptotic size bound can be found in [13]. TSLPs have been also extended to so-called forest straight-line programs (FSLPs) which allow to compress unranked node-labelled trees [17]. FSLPs are very similar to top DAGs [3] and also meet the size bound $\mathcal{O}(n/\log_{\hat{\sigma}} n)$ for unranked trees of size n. The reader should notice that the $\mathcal{O}(n/\log_{\hat{\sigma}} n)$-bound cannot be achieved by DAGs: the smallest DAG for an unlabelled binary tree of size n may still contain n edges.

Entropy bounds for grammar-based tree compressors. In this paper we first consider node-labelled binary trees: every node has a label from a finite set Σ of size σ and every non-leaf node has a left and a right child. For unlabelled binary trees the results of Kieffer and Yang on universal grammar-based string compressors have been extended to trees in [14, 38]. Whereas the universal tree encoder from [38] is based on DAGs (and needs a certain assumption on the average DAG size with respect to the input distribution), the encoder from [14] uses TSLPs of size $\mathcal{O}(n/\log n)$. For this, a binary encoding of TSLPs similar to the one for SLPs from [22] is proposed. In this paper we extend the binary TSLP-encoding from [14] to node-labelled binary trees (which is straightforward) and prove an entropy bound similar to the one from [33] for strings. To do this, we first have to come up with a reasonable higher order entropy for binary node-labelled trees (we just speak of binary trees in the following). Several notions of tree entropy can be found in the literature, but all are tailored towards unranked trees and do not yield nontrivial results for the special case of unlabelled binary trees (which is an indication that these entropy measures only deal with some partial aspects of trees).

- The k^{th}-order label entropy of a tree t [8] is the expected uncertainty about the label of a node v, given the k first node labels on the path from the parent node of v to the root of t.

[1]We define the size of a binary tree as its number of leaves.

- The degree entropy from [21] is the 0^{th}-order entropy of the node degrees (in [21] it is called the tree entropy, but this term could be misleading since here we talk about several entropy measures for trees).
- Recently, two combinations of degree entropy and k^{th}-order label entropy were proposed in [16]. The first one is the k^{th}-order degree-label entropy of a tree t: it is the expected uncertainty about the label of a node v of t, given (i) the k first labels on the path from the parent node of v to the root and (ii) the degree of v. The second combination is the k^{th}-order label-degree entropy of t: it is the expected uncertainty about the degree of a node v, given (i) the k first labels on the path from the parent node of v to the root and (ii) the label of v. In [16], the following two upper bounds (in bits) for the space to represent t are shown: (a) the k^{th}-order degree-label entropy of t plus the degree entropy of t and (b) the k^{th}-order label-degree entropy of t plus the k^{th}-order label entropy of t.

Degree entropy [21] is not useful in the context of binary trees, since a binary tree with n leaves has $n-1$ nodes of degree 2, which shows that the degree entropy divided by the number of nodes $(2n-1)$ converges to 1 when n increases. A binary unlabelled tree can be encoded with one bit per node; so the degree entropy only yields a trivial upper bound for the encoding length. Node labels are completely ignored by the degree entropy, therefore it cannot be used for node-labelled trees. On the other hand, k^{th}-order label entropy [8] is not useful for unlabelled trees since it ignores the shape of the tree. For the special case of unlabelled binary trees, also the combinations of [16] do not lead to useful entropy measures: for an unlabelled binary tree t, the above two entropy sums (a) and (b) are both equal to the degree entropy of t, which by the above remark only yields a trivial upper bound for the encoding length.

Our first contribution is the definition of a reasonable entropy measure for binary trees that can be also used for the unlabelled case. For this we define the k-history of a node v in a binary tree t by taking the last k edges on the unique path from the root to v. For each edge (v_1, v_2) traversed on this path we write down the node label of v_1 and a 0 (resp., 1) if v_2 is a left (resp., right) child of v_1. Thus, the k-history of a node is a word of length $2k$ that alternatingly consists of symbols from Σ and directions that are encoded by 0 or 1. For nodes at depth smaller than k we pad the history with 0's and a default node label $\square \in \Sigma$ in order to get length exactly k.[2] For each k-history h we then consider the joint probability distribution P_h^t of the node degree (either 0 or 2) and the node label, conditioned on the history h. Thus, $P_h^t(a, i)$ is the probability that a randomly chosen node among the nodes with history h is labelled with the symbol a and has $i \in \{0, 2\}$ children. The k^{th}-order empirical entropy of t, $H_k(t)$ for short, is then the sum of the entropies of these distributions P_h^t (the sum is taken over all histories h) weighted with the number of nodes with history h. This definition is similar to the definition of the k^{th}-order empirical entropy of a string. A detailed quantitative as well as experimental comparison of our k^{th}-order empirical entropy of trees with the above mentioned tree entropies from [8, 16, 21] can be found in [20].

[2]This is an ad hoc decision to make the definitions easier. In the appendix we discuss different approaches of how to deal with nodes of depth smaller than k, and prove that they asymptotically lead to the same entropy measure.

Our main result states that

$$|B(\mathcal{G}_t)| \leq H_k(t) + \mathcal{O}\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + \mathcal{O}\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right) + \sigma, \tag{2}$$

where t is a binary tree with n leaves, the grammar-based compressor $t \mapsto \mathcal{G}_t$ produces TSLPs of size $\mathcal{O}(n/\log_{\hat{\sigma}} n)$ for binary trees of size n with σ many node labels and $\hat{\sigma} = \max(2, \sigma)$. Moreover, B is an extension of the binary TSLP-encoding described in [14] from unlabelled binary trees to labelled binary trees (Section 3.3). If $k \leq o(\log_{\hat{\sigma}} n)$ then this bound can be simplified to $|B(\mathcal{G}_t)| \leq H_k(t) + o(n \log \hat{\sigma})$. The assumption $k \leq o(\log_{\hat{\sigma}} n)$ can be also found in [33]. In fact, Gagie argued in [11] that the k^{th}-order empirical entropy for strings stops being a reasonable complexity measure for almost all strings of length n over alphabets of size σ when $k \geq \log_{\hat{\sigma}} n$. Our bound (2) can be seen as an extension of the bound (1) [33] from strings to trees.

Our definition of k^{th}-order empirical entropy does not capture all regularities that can be exploited in grammar-based tree compression: Take for instance a complete unlabelled binary tree t_n of height n (all paths from the root to a leaf have length n). This tree has 2^n leaves and is well compressible: its minimal DAG has only $n + 1$ nodes, hence there also exists a TSLP of size $n + 1$ for t_n. But for every fixed k the k^{th}-order empirical entropy of t_n divided by n converges to 2 (the trivial upper bound) for $n \to \infty$. If $n \gg k$ then for every k-history z the number of leaves with k-history z is roughly the same as the number of internal nodes with k-history z. Hence, although t_n is highly compressible with TSLPs (and even DAGs), its k^{th}-order empirical entropy is close to the maximal value. This phenomenon is also known for strings: in contrast to grammar-based compression, k^{th}-order empirical string entropy does not capture repetitiveness in strings. In [24, Lemma 2.6] it is shown that the unnormalized empirical k^{th}-order entropy of a string ww is at least twice the unnormalized empirical k^{th}-order entropy of w (as long as $k \leq |w|$). In Section 6 we prove that the gap between grammar-based compression and empirical k^{th}-order entropy can be extreme in the following sense: there exists a family of strings S_n of length $\Theta(2^n)$ such that S_n has an SLP of size $\mathcal{O}(n)$ whereas the empirical k^{th}-order entropy of S_n is of size $\Omega(2^{n-k})$ for $k \in o(n)$.

Unranked trees. In Section 5 we present a simple extension of our entropy notion to node-labelled unranked trees. In an unranked tree the number of children of a node is arbitrary. Unranked trees are important in the area of XML, where the hierarchical structure of a document is represented by a node-labelled unranked tree.[3] For such a tree t we define the k^{th}-order empirical entropy as the k^{th}-order empirical entropy of the *first-child next-sibling* (fcns for short) encoding of t. The fcns-encoding of t is a binary tree which contains all nodes of t. If a node v of t has the first (i.e., left-most) child v_1 and the right sibling v_2 then v_1 (resp., v_2) is the left (resp., right) child of v in the fcns-encoding of t. If v has no child or no right sibling then one adds dummy leaves to the fcns-encoding in order to obtain a full binary tree. Our choice of defining the k^{th}-order empirical entropy of an unranked tree via the fcns-encoding is motivated by the fact that in XML document trees the label of a node v usually depends on the labels of the ancestors and the labels

[3]Being able to apply our tree entropy to XML trees is the main reason for considering node-labelled binary trees in this paper.

of the left siblings of v. This information is contained in the history of v in the fcns-encoding.

We present experimental results with real XML document trees showing that in these cases the k^{th}-order empirical entropy is indeed very small compared to the worst-case bit size. An unranked tree with n nodes and σ node labels can be encoded with $2n+\log_2(\sigma)n$ bits [18]. Up to low order terms, this is optimal. Table 1 shows the values of the k^{th}-order empirical entropy (for $k = 1, 2, 4, 8$) divided by $2n + \log_2(\sigma)n$ for several real XML trees (that were also used in other experiments for XML compression [29, 30]). For $k = 4$, these quotients never exceed 20% and for $k = 8$ all quotients are bounded by 13.5%.

Our experimental results combined with our entropy bound (2) for grammar-based compression are in accordance with the fact that grammar-based tree compressors yield excellent compression ratios for XML document trees, see e.g. [29]. Some of the XML documents from our experiments were also used in [29], where the performance of TreeRePair (currently the best grammar-based tree compressor from a practical point of view) on XML document trees was tested. An interesting observation is that those XML trees, for which our k-th order empirical entropy is large are indeed those XML trees with the worst compression ratio for TreeRePair in [29] (this is in particular the Treebank document from Table 1).

Proof ideas and comparison with related work. In this section we explain some of the main ideas behind the proof of our main inequality (2). In particular, we explain the connections to [14, 33], which are the main inspirations for this work.

In our work as well as in [14] probability distributions on trees play a crucial role. In our paper, we deal with so-called k^{th}-order tree processes. For a fixed set of node labels Σ, a k^{th}-tree process $\mathcal{P}$ specifies for every possible k-history z a probability distribution $P_z : \Sigma \times \{0, 2\} \to [0, 1]$ where $P_z(q, i)$ (for $a \in \Sigma$ and $i \in \{0, 2\}$) is the probability that in a tree a node with k-history z is labelled with a and has i children (here, we only deal with binary trees). Using these distributions P_z we can assign a probability $\mathsf{Prob}_{\mathcal{P}}(t)$ to every binary tree; see (10) in Section 2.2.3. One can show that the sum over all $\mathsf{Prob}_{\mathcal{P}}(t)$ for t a *finite* binary Σ-labelled tree is at most one; see Lemma 2 (the sum of all probabilities for finite trees can be smaller than one because a tree process may also produce infinite trees with non-zero probability). Our technical main result (Theorem 3, from which (2) easily follows) states that for every k^{th}-order tree process $\mathcal{P}$ with $\mathsf{Prob}_{\mathcal{P}}(t) > 0$ we have

$$|B(\mathcal{G}_t)| \leq -\log_2 \mathsf{Prob}_{\mathcal{P}}(t) + \mathcal{O}\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + \mathcal{O}\left(\frac{n \log \log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right) + \sigma. \tag{3}$$

Here, t, n, σ, $\mathcal{G}_t$ and B have the same meaning as in (2). This result can be also stated as follows: The tree encoder $t \mapsto B(\mathcal{G}_t)$ is universal for the class of all k^{th}-order tree processes (assuming that $k \leq o(\log_{\hat{\sigma}} n)$) so that $|B(\mathcal{G}_t)| \leq -\log_2 \mathsf{Prob}_{\mathcal{P}}(t) + o(n \log \hat{\sigma})$. Note that $-\log_2 \mathsf{Prob}_{\mathcal{P}}(t)$ is the self-information of t with respect to $\mathcal{P}$.

Also in [14], universality results are shown, but the underlying probability distributions are different. In [14] the focus is on so-called leaf-centric and depth-centric binary tree sources, which were introduced in [38]. A leaf-centric binary tree source $\mathcal{P}$ specifies for every $n \geq 1$ a probability distribution on the set of all binary trees with exactly n leaves. A depth-centric binary tree source $\mathcal{P}$ specifies for every $h \geq 0$ a probability distribution on the set of all binary trees of height h. In this

way, we again a assign a probability $\mathsf{Prob}_{\mathcal{P}}(t)$ to every binary tree (in [14, 38] only unlabelled trees are considered), but the sum of all these probabilities is of course infinity. It is shown in [14] that for certain classes $\mathcal{C}$ of leaf-centric (resp., depth-centric) binary tree sources the tree encoder $t \mapsto B(\mathcal{G}_t)$ is again universal in the sense that $|B(\mathcal{G}_t)| \leq -\log_2 \mathsf{Prob}_{\mathcal{P}}(t) + o(n \log \hat{\sigma})$ for all $\mathcal{P} \in \mathcal{C}$. The tree encoder $t \mapsto B(\mathcal{G}_t)$ used in [14] is almost the same as in this paper, except for the fact that [14] only deals with unlabelled binary trees.

In contrast to our work, the universality results from [14] do not seem to imply an entropy bound of the form (2). For this one would first have to come up with a suitable entropy notion that is related to the tree sources from [14]. For our notion of k^{th}-order empirical tree entropy $H_k(t)$ the relation to k^{th}-order tree processes is established by Theorem 1. It states that $H_k(t)$ is the minimal self-information of t with respect to a k^{th}-order tree process $\mathcal{P}$, where the minimum is taken over all $\mathcal{P}$. This result is exactly the link between (2) and (3).

Theorem 1 extends an analogous result for k^{th}-order empirical string entropy (instead of k^{th}-order empirical tree entropy) and k^{th}-order Markov processes (instead of k^{th}-order tree processes) by Gagie [11, Theorem 1].[4] Gagie's result is in fact one of the key ingredients for the proof of the entropy bound (1) of Ochoa and Navarro [33] for strings. We remark that Gagie's result is at the basis of the definition of pointwise redundancy [35] and that further similar and related results can be found in early papers on universal coding (see e.g. [25]).

The general dictionary for translating the proof of (1) [33] into our proof of (2) is:

strings	$\rightarrow$	node-labelled binary trees
k^{th}-order empirical string entropy	$\rightarrow$	k^{th}-order empirical tree entropy
k^{th}-order Markov processes	$\rightarrow$	k^{th}-order tree processes
string straight-line programs	$\rightarrow$	tree straight-line programs

The details of this extension from strings to trees turn out to be quite technical.

2. Preliminaries

In this section, we introduce some basic definitions concerning information theory (Section 2.1) and binary trees (Section 2.2).

With $\mathbb{N}$ we denote the natural numbers including 0. We use the standard $\mathcal{O}$-notation. If $b > 0$ is a constant, then we just write $\mathcal{O}(\log n)$ for $\mathcal{O}(\log_b n)$. We make the convention that $0 \cdot \log(0) = 0$ and $0 \cdot \log(x/0) = 0$ for $x \geq 0$. For the unit interval $\{r \in \mathbb{R} \mid 0 \leq r \leq 1\}$ we write $[0, 1]$.

Let $w = a_1 a_2 \cdots a_l \in \Sigma^*$ be a word over an alphabet Σ. With $|w| = l$ we denote the length of w. The empty word is denoted by ε. For $a \in \Sigma$ we denote with $|w|_a = |\{i \mid 1 \leq i \leq l, a_i = a\}|$ the number of occurrences of a in w.

2.1. Empirical distributions and empirical entropy.

Let A be a finite set. A probability distribution on A is a mapping $p : A \to [0, 1]$ such that $\sum_{a \in A} p(a) = 1$. For a probability distribution p on A we define its *Shannon entropy*

$$H(p) = \sum_{a \in A} -p(a) \log_2 p(a) = \sum_{a \in A} p(a) \log_2(1/p(a)).$$

[4] In fact, one could derive [11, Theorem 1] from Theorem 1 by considering chain-like trees.

We have $0 \leq H(p) \leq \log_2 |A|$. A well-known generalization of Shannon's inequality states that for every probability distribution p on A and any mapping $q : A \to [0,1]$ such that $\sum_{a \in A} q(a) \leq 1$ we have

$$H(p) = \sum_{a \in A} -p(a) \log_2 p(a) \leq \sum_{a \in A} -p(a) \log_2 q(a); \tag{4}$$

see [1] for a proof. Shannon's inequality is the special case where q is a probability distribution as well. The Kullback-Leibler divergence between two probability distributions p, q on A (see [6, Section 2.3]) is defined as

$$D(p \,\|\, q) = \sum_{a \in A} p(a) \cdot \log_2(p(a)/q(a)). \tag{5}$$

It is known that $D(p \,\|\, q) \geq 0$ for all p, q (this follows from Shannon's inequality) and $D(p \,\|\, q) = 0$ if and only if $p = q$.

Let $\overline{a} = (a_1, a_2, \ldots, a_l)$ be a tuple of elements that are from some (not necessarily finite) set S. The *empirical distribution* $p_{\overline{a}} : \{a_1, a_2, \ldots, a_l\} \to [0,1]$ of $\overline{a}$ is defined by

$$p_{\overline{a}}(a) = \frac{|\{i \mid 1 \leq i \leq l,\ a_i = a\}|}{l}.$$

We use this (and the following) definition also for words over some alphabet by identifying a word $w = a_1 a_2 \cdots a_l$ with the tuple $(a_1, a_2, \ldots, a_l)$. The *unnormalized empirical entropy* of $\overline{a}$ is

$$H(\overline{a}) = l \cdot H(p_{\overline{a}}) = -\sum_{i=1}^{l} \log_2 p_{\overline{a}}(a_i). \tag{6}$$

From (4) it follows that for a tuple $\overline{a} = (a_1, a_2, \ldots, a_l)$ with $a_1, \ldots, a_l \in S$ and real numbers $q(a) \geq 0$ ($a \in S$) with $\sum_{a \in \{a_1, \ldots, a_l\}} q(a) \leq 1$ we have

$$\sum_{i=1}^{l} -\log_2 p_{\overline{a}}(a_i) \leq \sum_{i=1}^{l} -\log_2 q(a_i). \tag{7}$$

We also need the famous log-sum inequality, see e.g. [6, Theorem 2.7.1] (recall our conventions $0 \cdot \log(0) = 0$ and $0 \cdot \log(x/0) = 0$ for $x \geq 0$):

Lemma 1. *Let $a_1, a_2, \ldots, a_l, b_1, b_2, \ldots, b_l \geq 0$ be real numbers. Moreover, let $a = \sum_{i=1}^{l} a_i$ and $b = \sum_{i=1}^{l} b_i$. Then*

$$a \log_2 \left(\frac{b}{a}\right) \geq \sum_{i=1}^{l} a_i \log_2 \left(\frac{b_i}{a_i}\right).$$

2.2. **Trees, tree processes, and tree entropy.**

2.2.1. *Trees and contexts.* Let Σ denote a finite non-empty alphabet of size $|\Sigma| = \sigma$. Later, we will need a fixed distinguished symbol from Σ that we will denote with $\square \in \Sigma$. We will also need the value $\hat{\sigma} = \max\{2, \sigma\}$. With $\mathcal{T}(\Sigma)$ we denote the set of *labelled binary trees* over the alphabet Σ. Formally, it is inductively defined as the smallest set of terms over Σ such that

- $\Sigma \subseteq \mathcal{T}(\Sigma)$ and
- if $t_1, t_2 \in \mathcal{T}(\Sigma)$ and $a \in \Sigma$, then $a(t_1, t_2) \in \mathcal{T}(\Sigma)$.

If e.g. $\Sigma = \{a, b\}$, then $a \in \mathcal{T}(\Sigma)$ is the binary tree with a single node labelled by a and $a(b(b(a, b), a), a(b, a)) \in \mathcal{T}(\Sigma)$ is the binary tree depicted on the left of Figure 1.

A *tree encoder* is an injective mapping $E : \mathcal{T}(\Sigma) \to \{0, 1\}^*$ such that the range $E(\mathcal{T}(\Sigma))$ is prefix-free, i.e., there do not exist $t, t' \in \mathcal{T}(\Sigma)$ with $t \neq t'$ such that $E(t)$ is a prefix of $E(t')$.

It is convenient to define the size $|t|$ of $t \in \mathcal{T}(\Sigma)$ as the number of leaves of t, which can be inductively defined by $|a| = 1$ and $|a(t_1, t_2)| = |t_1| + |t_2|$ for $a \in \Sigma$ and $t_1, t_2 \in \mathcal{T}(\Sigma)$. Note that $2|t| - 1$ is the number of occurrences of symbols from Σ in t. Let $\mathcal{T}_n(\Sigma) = \{t \in \mathcal{T}(\Sigma) \mid |t| = n\}$ for $n \geq 1$. Note that $\mathcal{T}_1(\Sigma) = \Sigma$. We have $|\mathcal{T}_n(\Sigma)| = \sigma^{2n-1} C_{n-1}$, where C_k is the k^{th} Catalan number. These numbers satisfy the following well-known asymptotic estimate

$$C_k \sim \frac{4^k}{\sqrt{\pi} k^{\frac{3}{2}}}, \tag{8}$$

see e.g. [10]. In fact, we have $C_k \leq 4^k$ for all $k \geq 0$ and hence $|\mathcal{T}_n(\Sigma)| \leq (2\sigma)^{2n}$.

A *context* is a labelled binary tree, where exactly one leaf is labelled with the special symbol $x \notin \Sigma$ (called the *parameter*); all other nodes are labelled with symbols from Σ. Formally, the set of contexts $\mathcal{C}(\Sigma)$ is the smallest set such that

- $x \in \mathcal{C}(\Sigma)$ and
- if $a \in \Sigma$, $c \in \mathcal{C}(\Sigma)$ and $t \in \mathcal{T}(\Sigma)$ then also $a(c, t), a(t, c) \in \mathcal{C}(\Sigma)$.

If e.g. $\Sigma = \{a, b\}$, then $x \in \mathcal{C}(\Sigma)$ is the context with a single node labelled by the parameter x and $a(b(b(a, b), x), a(b, a)) \in \mathcal{C}(\Sigma)$ is the context depicted on the right of Figure 1.

For a tree or context $t \in \mathcal{T}(\Sigma) \cup \mathcal{C}(\Sigma)$ and a context $c \in \mathcal{C}(\Sigma)$, we denote by $c[t]$ the tree or context which results from c by replacing the unique occurrence of the parameter x by t. For example $c = a(a, x)$ and $t = b(a, a)$ yield $c[t] = a(a, b(a, a))$ (with $\Sigma = \{a, b\}$). For a context c we define $|c|$ inductively by $|x| = 0$ and $|a(c, t)| = |a(t, c)| = |t| + |c|$ for $c \in \mathcal{C}(\Sigma)$ and $t \in \mathcal{T}(\Sigma)$. In other words, $|c|$ is the number of leaves of c, where the unique occurrence of the parameter x is not counted. Note that $|c| = |c[a]| - 1$, where $a \in \Sigma$ is arbitrary. We define $\mathcal{C}_n(\Sigma) = \{c \in \mathcal{C}(\Sigma) \mid |c| = n\}$ for $n \in \mathbb{N}$. Since the set Σ will not change in this paper, we use the abbreviations $\mathcal{T}$, $\mathcal{T}_n$, $\mathcal{C}$, and $\mathcal{C}_n$ for $\mathcal{T}(\Sigma)$, $\mathcal{T}_n(\Sigma)$, $\mathcal{C}(\Sigma)$, and $\mathcal{C}_n(\Sigma)$, respectively. Contexts will be needed for the definition of tree straight-line programs in Section 3.

Occasionally, we will consider a binary tree or context as a graph with nodes and edges in the usual way, where each node is labelled with a symbol from Σ (or x in the case of a context). Note that $t \in \mathcal{T}_n$ has $2n - 1$ nodes in total (n leaves and $n - 1$ internal nodes) and $c \in \mathcal{C}_n$ has $2n + 1$ nodes in total ($n + 1$ leaves and n internal nodes).

It is convenient to define a node v of $s \in \mathcal{T} \cup \mathcal{C}$ as a bit string that describes the path from the root to the node (0 means left, 1 means right). Formally, we define the node set $V(s) \subseteq \{0, 1\}^*$ of $s \in \mathcal{T} \cup \mathcal{C}$ by

- $V(a) = \{\varepsilon\}$ for every $a \in \Sigma$,
- $V(x) = \emptyset$ and
- $V(a(s_0, s_1)) = \{iw \mid i \in \{0, 1\}, w \in V(s_i)\} \cup \{\varepsilon\}$ for every $a \in \Sigma$.

Note that for a context $c \in \mathcal{C}$, the set $V(c)$ does not contain the unique node in c labelled with the parameter x. We use this definition due to better readability of the paper since we mostly need the set of nodes without the parameter node.

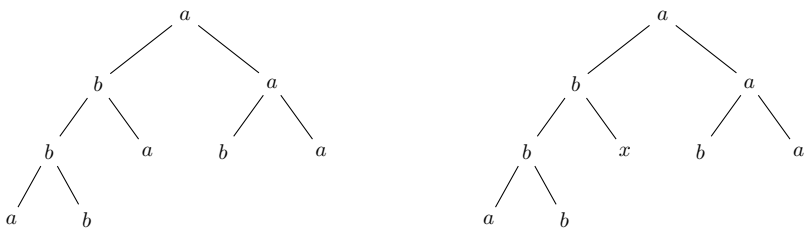

FIGURE 1. A tree (left) and a context (right).

Also, it is still possible to uniquely determine from $V(c)$ the path to the parameter x due to the following properties: For a tree $t \in \mathcal{T}$ we have $w0 \in V(t)$ if and only if $w1 \in V(t)$ for all $w \in \{0,1\}^*$ since each node has zero or two children. The only context c which fulfills this property is $c = x$, i.e., the parameter node is the only node of c and $V(c) = \emptyset$. For all other contexts $c \in \mathcal{C}$ this property is violated since there exists a unique $w \in \{0,1\}^*$ such that $w0 \in V(c)$ (respectively, $w1 \in V(c)$) and $w1 \notin V(c)$ (respectively, $w0 \notin V(c)$). In this case the parameter node is $w1$ (respectively, $w0$). Alternatively, the parameter node of a context c is the single node in the set $V(c[a]) \setminus V(c)$ for a symbol $a \in \Sigma$. We denote this node with $\omega(c) \in \{0,1\}^*$. In other words: $V(c[a]) \setminus V(c) = \{\omega(c)\}$.

Example 1. *Consider the tree $t = a(b(b(a,b),a),a(b,a))$ with $\Sigma = \{a,b\}$ depicted on the left of Figure 1. We have $V(t) = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001\}$. For the context $c = a(b(b(a,b),x),a(b,a))$ depicted on the right of Figure 1, we have $t = c[a]$ and $\omega(c) = 01$.*

Consider a tree or context s and let $v \in V(s)$. The leaves of s are those strings in $V(s)$ that are maximal with respect to the prefix relation. The length $|v|$ is the depth of the node v in s and the depth of s is the maximal depth of a node in $V(s)$ (the depth of $s = x$ is not defined but also not needed). Let $\lambda_s : V(s) \to \Sigma \times \{0,2\}$ denote the function mapping a node v to the pair (a,i) where $a \in \Sigma$ is the label of v and $i \in \{0,2\}$ is the number of children of v. We can define this function inductively as follows:

- $\lambda_a(\varepsilon) = (a,0)$ for $a \in \Sigma$,
- $\lambda_s(\varepsilon) = (a,2)$ for $s = a(s_0,s_1)$ with $a \in \Sigma$ and $s_0, s_1 \in \mathcal{T} \cup \mathcal{C}$,
- $\lambda_s(iw) = \lambda_{s_i}(w)$ for $s = a(s_0,s_1)$ with $a \in \Sigma$, $s_0, s_1 \in \mathcal{T} \cup \mathcal{C}$ and $iw \in V(s)$.

Note that in the last case, if s is a context, we cannot have $s_i = x$ because we must have $w \in V(s_i)$. In the following, we will omit the subscript s in $\lambda_s(v)$ if s is clear from the context.

2.2.2. *Histories.* We now come to the crucial notion of the history of a node v in a tree or context. Intuitively, the history of v records all information that can be obtained by walking from the root of the tree/context straight down to the node v. This information consists of the directions (0 or 1) along the path from the root to v and the node labels along this path.

First, we define the set of *histories* as

$$\mathcal{L} = (\Sigma\{0,1\})^* = \{a_1 i_1 \cdots a_n i_n \mid n \geq 0, a_k \in \Sigma, i_k \in \{0,1\} \text{ for all } 1 \leq k \leq n\}.$$

For an integer $k \geq 0$, let

$$\mathcal{L}_k = \{w \in \mathcal{L} \mid |w| = 2k\}$$

be the set of *k-histories* and let $\ell_k : \mathcal{L} \to \mathcal{L}_k$ denote the partial function mapping a history $z \in \mathcal{L}$ with $|z| \geq 2k$ to the suffix of z of length $2k$, i.e., $\ell_k(a_1 i_1 \cdots a_n i_n) = a_{n-k+1} i_{n-k+1} \cdots a_n i_n$ (the function ℓ_0 maps a string to the empty string).

For a tree t and a node $v \in V(t)$ (resp., a context c and a node $v \in V(c) \cup \{\omega(c)\}$), we inductively define its *history* $h(v) \in \mathcal{L}$ (in t) by

- $h(\varepsilon) = \varepsilon$ and
- $h(wi) = h(w)ai$ for $i \in \{0,1\}$ and $wi \in V(t)$ (resp., $wi \in V(c) \cup \{\omega(c)\}$).

Here, a is the symbol that labels the node w, i.e., $\lambda(w) = (a, 2)$. That is, in order to obtain $h(v)$, while walking downwards in the tree from the root node to the node v we alternately concatenate symbols from Σ with binary numbers in $\{0,1\}$ such that the symbol from Σ corresponds to the label of the current node and the binary number 0 (resp., 1) states that we move on to the left (resp. right) child node. Note that the symbol that labels v is not part of the history of v.

The k-history of a tree node $v \in V(t)$ is

$$h_k(v) = \ell_k((\Box 0)^k h(v)) \in \mathcal{L}_k,$$

i.e., the suffix of length $2k$ of the word $(\Box 0)^k h(v)$, where $\Box$ is a fixed dummy symbol in Σ (the choice is arbitrary). This means that if $|v| \geq k$ then $h_k(v)$ describes the last k directions and node labels along the path from the root to node v. If $|v| < k$, we pad the history of v with $\Box$'s and zeros such that $h_k(v) \in \mathcal{L}_k$. In the appendix, we discuss other reasonable approaches of how to deal with nodes of depth smaller than k. It will turn out that those choices have only a minor influence on our main results. The k-history of a tree node v is the natural extension of the k preceding symbols of a string position.

For $z \in \mathcal{L}_k$ we denote with

$$V_z(t) = \{v \in V(t) \mid h_k(v) = z\} \tag{9}$$

the set of nodes in t with k-history z.

Example 2. *Consider the tree $t = a(b(b(a,b),a),a(b,a))$ from Example 1 and let $\Box = a \in \Sigma$. Then, $h(001) = h_3(001) = a0b0b1$ and $h_4(10) = a0a0a1a0$.*

2.2.3. *Tree processes.* A *tree process* is an infinite tuple $\mathcal{P} = (P_z)_{z \in \mathcal{L}}$ where every P_z is a probability distribution on $\Sigma \times \{0,2\}$. A pair $(a,i) \in \Sigma \times \{0,2\}$ represents the information that a certain tree node v is labelled with a and has i children. The probability $P_z(a,i)$ is the probability that the tree node with history z (if it exists it is of course unique) is labelled with a and has i children. Tree processes will be used in Section 2.2.4 for the definition of our k^{th}-order empirical entropy for trees. Thereby, we will restrict to so-called k^{th}-order tree process that can be seen as an extension of k^{th}-order Markov processes to binary labelled trees. But first we will derive a few facts about general tree processes.

A tree process can be used to randomly construct a tree from $\mathcal{T}$ as follows: In a top-down way we determine for every tree node its label (from Σ) and its number of children, where this decision depends on the history of the tree node. We start at the root node, whose history is the empty word ε. If we have reached a tree node v with history $z \in \mathcal{L}$ then we use the probability distribution P_z to randomly choose a pair $(a,i) \in \Sigma \times \{0,2\}$. We assign the label $a \in \Sigma$ to v. If $i = 0$ then v becomes a leaf, otherwise the process continues at the two children $v0$ and $v1$ (whose history is well-defined). Note that in this way we may produce infinite trees with non-zero

probability (e.g. if $P_z(a,2) = 1$ for some $a \in \Sigma$). For a finite tree $s \in \mathcal{T}$ we obtain the probability

$$\mathsf{Prob}_{\mathcal{P}}(s) = \prod_{v \in V(s)} P_{h(v)}(\lambda_s(v)). \tag{10}$$

For technical reason we will use this definition also for the case that s is a context. In other words: we associate with $\mathcal{P}$ the function $\mathsf{Prob}_{\mathcal{P}} : \mathcal{T} \cup \mathcal{C} \to [0,1]$ using (10). Note that if s is a context, then the parameter node of s is not in $V(s)$ and therefore does not contribute to $\mathsf{Prob}_{\mathcal{P}}(s)$.

We first prove upper bounds for the sums $\sum_{t \in \mathcal{T}} \mathsf{Prob}_{\mathcal{P}}(t)$ and $\sum_{c \in \mathcal{C}} \mathsf{Prob}_{\mathcal{P}}(c)$. These bounds will be needed in the proof of our main technical result (Lemma 7 in Section 4).

The reason that we obtain only an inequality in the following lemma is that the above tree generating process may also produce infinite trees with non-zero probability.

Lemma 2. *Let $\mathcal{P}$ be a tree process. Then $\sum_{t \in \mathcal{T}} \mathsf{Prob}_{\mathcal{P}}(t) \leq 1$.*

Proof. First, note that as $\mathsf{Prob}_{\mathcal{P}}(t)$ is non-negative for every tree $t \in \mathcal{T}$, the order of summation in the sum $\sum_{t \in \mathcal{T}} \mathsf{Prob}_{\mathcal{P}}(t)$ does not matter: If $\sum_{t \in \mathcal{T}} \mathsf{Prob}_{\mathcal{P}}(t) < \infty$, then this sum converges absolutely, and thus any rearrangement of the order of summation does not change its value.

Define the set of trees $\mathcal{T}'_n$ inductively by $\mathcal{T}'_1 = \mathcal{T}_1$ and

$$\mathcal{T}'_{n+1} = \mathcal{T}_1 \cup \{a(t_0, t_1) \mid a \in \Sigma, t_0, t_1 \in \mathcal{T}'_n\}.$$

Thus, $\mathcal{T}'_{n+1}$ is the set of all trees of height at most n. We have $\mathcal{T}'_n \subsetneq \mathcal{T}'_{n+1}$ and $\mathcal{T} = \bigcup_{n \geq 1} \mathcal{T}'_n$. It then suffices to show $\sum_{t \in \mathcal{T}'_n} \mathsf{Prob}_{\mathcal{P}}(t) \leq 1$ for every $n \geq 1$. We prove this by induction on n. For this, it turns out to be useful to define for every $z \in \mathcal{L}$ the shifted tree process $\mathcal{P}_z = (P_{zz'})_{z' \in \mathcal{L}}$. We then prove by induction on n that $\sum_{t \in \mathcal{T}'_n} \mathsf{Prob}_{\mathcal{P}_z}(t) \leq 1$ for every $n \geq 1$ and all $z \in \mathcal{L}$ (in particular, for $z = \varepsilon$ as well, which then yields the original statement). For $n = 1$ we have

$$\sum_{t \in \mathcal{T}'_1} \mathsf{Prob}_{\mathcal{P}_z}(t) = \sum_{t \in \mathcal{T}_1} \mathsf{Prob}_{\mathcal{P}_z}(t) = \sum_{a \in \Sigma} P_z(a,0) \leq 1.$$

Now assume that $n \geq 1$. We get

$$\begin{aligned}
&\sum_{t \in \mathcal{T}'_{n+1}} \mathsf{Prob}_{\mathcal{P}_z}(t) \\
= \;& \sum_{t \in \mathcal{T}_1} \mathsf{Prob}_{\mathcal{P}_z}(t) + \sum_{a \in \Sigma} \sum_{t_0 \in \mathcal{T}'_n} \sum_{t_1 \in \mathcal{T}'_n} \mathsf{Prob}_{\mathcal{P}_z}(a(t_0,t_1)) \\
= \;& \sum_{a \in \Sigma} P_z(a,0) + \\
& \sum_{a \in \Sigma} \sum_{t_0 \in \mathcal{T}'_n} \sum_{t_1 \in \mathcal{T}'_n} P_z(a,2) \cdot \prod_{v \in V(t_0)} P_{za0h(v)}(\lambda_{t_0}(v)) \cdot \prod_{v \in V(t_1)} P_{za1h(v)}(\lambda_{t_1}(v)) \\
= \;& \sum_{a \in \Sigma} P_z(a,0) + \\
& \sum_{a \in \Sigma} \left(P_z(a,2) \cdot \sum_{t_0 \in \mathcal{T}'_n} \prod_{v \in V(t_0)} P_{za0h(v)}(\lambda_{t_0}(v)) \cdot \sum_{t_1 \in \mathcal{T}'_n} \prod_{v \in V(t_1)} P_{za1h(v)}(\lambda_{t_1}(v)) \right)
\end{aligned}$$

$$\begin{aligned}
&= \sum_{a\in\Sigma} P_z(a,0) + \\
&\quad \sum_{a\in\Sigma}\left(P_z(a,2)\cdot \sum_{t_0\in\mathcal{T}'_n} \mathsf{Prob}_{\mathcal{P}_{za0}}(t_0)\cdot \sum_{t_1\in\mathcal{T}'_n} \mathsf{Prob}_{\mathcal{P}_{za1}}(t_1)\right)\\
&\leq \sum_{a\in\Sigma} P_z(a,0) + \sum_{a\in\Sigma} P_z(a,2)\\
&= 1.
\end{aligned}$$

This proves the lemma. □

Lemma 2 cannot be extended to contexts, but the following bound will suffice for our purpose.

Lemma 3. *Let $\mathcal{P}$ be a tree process. We have $\sum_{c\in\mathcal{C}_n} \mathsf{Prob}_{\mathcal{P}}(c) \leq n+1$ for every $n \geq 1$.*

Proof. In order to bound $\sum_{c\in\mathcal{C}_n} \mathsf{Prob}_{\mathcal{P}}(c)$, we first represent the probability of each context $c \in \mathcal{C}_n$ as a sum of probabilities of trees. So fix a context $c \in \mathcal{C}_n$ for the first part of the proof. Note first that in general no tree t exists such that $\mathsf{Prob}_{\mathcal{P}}(c) \leq \mathsf{Prob}_{\mathcal{P}}(t)$ (or even $\mathsf{Prob}_{\mathcal{P}}(c) = \mathsf{Prob}_{\mathcal{P}}(t)$) since $\omega(c)$ (the parameter node of c) does not contribute to the probability of the context c. For example, the tree $c[a]$ ($a \in \Sigma$) which results from c by replacing the parameter node by an a-labelled leaf node has probability $\mathsf{Prob}_{\mathcal{P}}(c)\cdot P_{h(\omega(c))}(a,0) \leq \mathsf{Prob}_{\mathcal{P}}(c)$. In order to bound $\mathsf{Prob}_{\mathcal{P}}(c)$, the idea is to replace the parameter node by all possible trees and not only by a single node. So consider the set $c[\mathcal{T}] = \{c[t] \mid t \in \mathcal{T}\}$ of all trees that arise from c by replacing the parameter by an arbitrary tree. Unfortunately, the total probability $\sum_{t\in c[\mathcal{T}]} \mathsf{Prob}_{\mathcal{P}}(t)$ can still be strictly smaller than $\mathsf{Prob}_{\mathcal{P}}(c)$ since there might be infinite trees with positive probability with respect to $\mathcal{P}$. To get rid of this problem, we fix an element $a \in \Sigma$ and modify $\mathcal{P}$ to a tree process $\mathcal{P}' = (P'_z)_{z\in\mathcal{L}}$ such that (i) $P'_z = P_z$ for $|z| \leq 2n$ and (ii) $P'_z(a,0) = 1$ and $P'_z(a',i) = 0$ for every $(a',i) \in \Sigma\times\{0,2\}\setminus\{(a,0)\}$ and $|z| > 2n$. The tree process $\mathcal{P}'$ is created such that all nodes v of depth $|v| \leq n$ contribute the probability $P_{h(v)}(\lambda(v))$ as before and all nodes of depth $n+1$ in a tree are a-labelled leaves with probability 1. Note first that for each context $c \in \mathcal{C}_n$ and each node $v \in V(c)$ we have $|v| \leq n$ and thus $P'_{h(v)}(\lambda(v)) = P_{h(v)}(\lambda(v))$. Secondly, all trees of depth larger than $n+1$ have probability 0 with respect to $\mathcal{P}'$ (including infinite trees). Hence, we get $\sum_{t\in\mathcal{T}} \mathsf{Prob}_{\mathcal{P}'}(t) = 1$. We obtain

$$\begin{aligned}
\sum_{t\in c[\mathcal{T}]} \mathsf{Prob}_{\mathcal{P}'}(t) &= \sum_{t\in c[\mathcal{T}]}\prod_{v\in V(t)} P'_{h(v)}(\lambda(v))\\
&= \sum_{t\in c[\mathcal{T}]}\left(\prod_{v\in V(c)} P'_{h(v)}(\lambda(v))\prod_{v\in V(t)\setminus V(c)} P'_{h(v)}(\lambda(v))\right)\\
&= \mathsf{Prob}_{\mathcal{P}}(c)\cdot \underbrace{\sum_{t\in c[\mathcal{T}]}\prod_{v\in V(t)\setminus V(c)} P'_{h(v)}(\lambda(v))}_{(a)}.
\end{aligned}$$

We claim that (a) equals 1. To see this, consider the tree process $\mathcal{P}'' = (P''_z)_{z\in\mathcal{L}}$ with $P''_z = P'_{h(\omega(c))z}$. Also for $\mathcal{P}''$ only finite trees have non-zero probability and

thus $\sum_{t\in\mathcal{T}} \mathsf{Prob}_{\mathcal{P}''}(t) = 1$. We have

$$\begin{aligned}(a) &= \sum_{t\in\mathcal{T}} \prod_{v\in V(t)} P'_{h(\omega(c))h(v)}(\lambda(v)) \\ &= \sum_{t\in\mathcal{T}} \prod_{v\in V(t)} P''_{h(v)}(\lambda(v)) \\ &= \sum_{t\in\mathcal{T}} \mathsf{Prob}_{\mathcal{P}''}(t) = 1.\end{aligned}$$

It follows that $\mathsf{Prob}_{\mathcal{P}}(c) = \sum_{t\in c[\mathcal{T}]} \mathsf{Prob}_{\mathcal{P}'}(t)$. In the second part of the proof it remains to bound $\sum_{c\in\mathcal{C}_n} \mathsf{Prob}_{\mathcal{P}}(c) = \sum_{c\in\mathcal{C}_n} \sum_{t\in c[\mathcal{T}]} \mathsf{Prob}_{\mathcal{P}'}(t)$. The key point here is that for each tree $t \in \mathcal{T}$ there are at most $n+1$ different contexts $c \in \mathcal{C}_n$ such that $t \in c[\mathcal{T}]$. Note that for a tree t, the number of different contexts $c \in \mathcal{C}_n$ such that $t \in c[\mathcal{T}]$ is exactly the number of nodes $v \in V(t)$ such that replacing the subtree rooted at v by the parameter x yields a context c with $|c| = n$. This is the same as the number of subtrees of t with $|t| - n$ leaves. Since different subtrees in t of equal size do not share nodes, we can bound the number of subtrees with $|t| - n$ leaves by $|t|/(|t| - n)$. We can assume that $|t| > n$ since otherwise there is no context $c \in \mathcal{C}_n$ such that $t \in c[\mathcal{T}]$. So we have $|t| = n + k$ for some $k > 0$ and the number of subtrees of t with $|t| - n$ leaves is at most $(n + k)/k = n/k + 1 \leq n + 1$. We get

$$\sum_{c\in\mathcal{C}_n} \sum_{t\in c[\mathcal{T}]} \mathsf{Prob}_{\mathcal{P}'}(t) \leq (n+1) \sum_{t\in\mathcal{T}} \mathsf{Prob}_{\mathcal{P}'}(t) = n + 1.$$

This concludes the proof of the lemma. □

Lemma 3 will be needed in order to prove our main technical Lemma (Lemma 7 in Section 4).

A *k^{th}-order tree process* is a tree process $\mathcal{P} = (P_z)_{z\in\mathcal{L}}$ such that $P_z = P_{z'}$ if $\ell_k((\Box 0)^k z) = \ell_k((\Box 0)^k z')$. Thus, the probability distribution that is chosen for a certain tree node v depends only on the k-history of v. As mentioned before, k^{th}-order tree processes can be seen as a tree extension of k^{th}-order Markov processes for strings. We will identify the k^{th}-order tree process $\mathcal{P} = (P_z)_{z\in\mathcal{L}}$ with the finite tuple $(P_z)_{z\in\mathcal{L}_k}$ (recall that $\mathcal{L}_k$ is the set of k-histories); it contains all information about $\mathcal{P}$. Note that for a k^{th}-order tree process $\mathcal{P}$ we can compute $\mathsf{Prob}_{\mathcal{P}}(s)$ for a tree or context s as

$$\mathsf{Prob}_{\mathcal{P}}(s) = \prod_{z\in\mathcal{L}_k} \prod_{v\in V_z(s)} P_z(\lambda(v)), \tag{11}$$

where $V_z(s)$ is defined in (9) and the empty product (which arises in case $V_z(s) = \emptyset$) is 1.

2.2.4. *Higher-order entropy of a tree.* We now come to the definition of our k^{th}-order empirical entropy of trees. As for stings, the term "empirical" refers to the fact that we assign an information content to a single tree instead of a probability distribution on trees. This has the advantage that empirical entropy is also useful in situations where we do not know the underlying probability distribution.

Let us fix $k \geq 0$. We define the *k^{th}-order (unnormalized) empirical entropy* $H_k(t)$ of a tree $t \in \mathcal{T}_n$ as follows: For $z \in \mathcal{L}_k$ let

$$m_z^t = |V_z(t)|$$

be the number of nodes of t with k-history z and for $\tilde{a} \in \Sigma \times \{0,2\}$ let

$$m^t_{z,\tilde{a}} = |\{v \in V_z(t) \mid \lambda(v) = \tilde{a}\}|. \tag{12}$$

We then define the *empirical k^{th}-order tree process* $\mathcal{P}^t = (P^t_z)_{z \in \mathcal{L}_k}$ by

$$P^t_z(\tilde{a}) = \frac{m^t_{z,\tilde{a}}}{m^t_z} \tag{13}$$

for all $\tilde{a} \in \Sigma \times \{0,2\}$ and all $z \in \mathcal{L}_k$ with $m^t_z > 0$. If $\tilde{a} = (a,i) \in \Sigma \times \{0,2\}$, then this is the probability that a node of t that is randomly chosen among the nodes with history z is labelled with a and has i children. If $m^t_z = 0$, then we can define P^t_z as an arbitrary distribution. Finally, we define

$$H_k(t) = \sum_{z \in \mathcal{L}_k} m^t_z \cdot H(P^t_z). \tag{14}$$

This definition extends the notion of k^{th}-order empirical entropy for strings to trees. The k-history of a tree node takes the role of the k preceding symbols of a string position. Note that

$$0 \le H_k(t) \le (2n-1)\log_2(2\sigma) = (2n-1)(1+\log_2\sigma)$$

since $0 \le H(P^t_z) \le \log_2(2\sigma)$ and $\sum_{z \in \mathcal{L}_k} m^t_z = 2n-1$. This upper bound on the entropy matches the information theoretic bound for the worst-case output length of any tree encoder on $\mathcal{T}_n$. Using the asymptotic bound (8) for the Catalan numbers, one sees that for any tree encoder there must exist a tree $t \in \mathcal{T}_n$ which is encoded with $2\log_2(2\sigma)n - o(n) = 2(\log_2\sigma + 1)n - o(n)$ bits. The k^{th}-order empirical entropy $H_k(t)$ is a lower bound on the coding length of a tree encoder that encodes for each node the relevant information (the label of the node and the binary information whether the node is a leaf or internal) depending on the k-history of the node: Thus, k^{th}-order empirical entropy is the expected uncertainty about the label and degree (0 or 2) of a node, given the last k directions and labels on the path from the root node to the node.

Example 3. *Let t denote the binary tree $t = a(b(b(a,b),a),a(b,a))$ as depicted on the left of Figure1. In order to compute the first order empirical entropy $H_1(t)$ of t, we have to consider k-histories of t with $k = 1$: Let $\square = a$. It follows that $V_{a0}(t) = \{\varepsilon, 0, 10\}$, $V_{b0}(t) = \{00, 000\}$, $V_{a1}(t) = \{1, 11\}$ and $V_{b1}(t) = \{01, 001\}$. Thus, we have $m^t_{a0} = 3$ and $m^t_{a1} = m^t_{b0} = m^t_{b1} = 2$. Next, for each k-history z, we consider $\lambda(v)$ for $v \in V_z(t)$: For $z = a0$, we have $\lambda(\varepsilon) = (a,2)$, $\lambda(0) = (b,2)$ and $\lambda(10) = (b,0)$. Hence, $m^t_{a0,(a,2)} = m^t_{a0,(b,0)} = m^t_{a0,(b,2)} = 1$ and $H(P^t_{a0}) = \log_2(3)$. Analogously, we find $H(P^t_{b0}) = H(P^t_{a1}) = H(P^t_{b1}) = 1/2\log_2(2) + 1/2\log_2(2) = 1$. Altogether, this yields $H_1(t) = 3 \cdot \log_2(3) + 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 1$ which is roughly 9.3.*

One can define $H_k(t)$ alternatively in the following way: Take a k-history $z \in \mathcal{L}_k$ and enumerate the set $V_z(t)$ in an arbitrary way as $v_1, v_2, \ldots, v_j$. Define the string $w(t,z) = \lambda(v_1)\lambda(v_2)\cdots\lambda(v_j) \in (\Sigma \times \{0,2\})^*$. We have

$$H_k(t) = \sum_{z \in \mathcal{L}_k} H(w(t,z)),$$

where the empirical entropy $H(w(t,z))$ is defined according to (6).

The following theorem and its proof are very similar to a corresponding statement for strings shown by Gagie [11]. One obtains Gagie's result by replacing in the

following theorem (i) k^{th}-order tree processes by k^{th}-order Markov processes and (ii) k^{th}-order empirical entropy of trees by k^{th}-order empirical entropy of strings.

Theorem 1. *Let $t \in \mathcal{T}$. For every k^{th}-order tree process $\mathcal{P} = (P_z)_{z\in\mathcal{L}_k}$ with $\mathsf{Prob}_{\mathcal{P}}(t) > 0$ we have*

$$H_k(t) \leq -\log_2 \mathsf{Prob}_{\mathcal{P}}(t)$$

with equality if and only if $P_z^t = P_z$ for all $z \in \mathcal{L}_k$ with $m_z^t > 0$.

Proof. We have

$$\begin{aligned}
-\log_2 \mathsf{Prob}_{\mathcal{P}}(t) &\overset{(11)}{=} \sum_{z\in\mathcal{L}_k} \sum_{v\in V_z(t)} \log_2(1/P_z(\lambda(v)) \\
&\overset{(12)}{=} \sum_{z\in\mathcal{L}_k} \sum_{\tilde{a}\in\Sigma\times\{0,2\}} m_{z,\tilde{a}}^t \log_2(1/P_z(\tilde{a})) \\
&\overset{(13)}{=} \sum_{z\in\mathcal{L}_k} m_z^t \sum_{\tilde{a}\in\Sigma\times\{0,2\}} P_z^t(\tilde{a}) \cdot (\log_2(P_z^t(\tilde{a})/P_z(\tilde{a})) + \log_2(1/P_z^t(\tilde{a}))) \\
&\overset{(5)}{=} \sum_{z\in\mathcal{L}_k} m_z^t \cdot (D(P_z^t \,\|\, P_z) + H(P_z^t)) \\
&\overset{(14)}{\geq} H_k(t)
\end{aligned}$$

with equality in the last line if and only if $P_z^t = P_z$ for all $z \in \mathcal{L}_k$ with $m_z^t > 0$. □

Theorem 1 will be a main ingredient for the proof of our main result (Theorem 4 in Section 4) in the same way as the above mentioned result from [11] is used by Ochoa and Navarro [33] in order to bound the compression ratio of grammar-based string compressors by the k^{th}-order empirical entropy of strings: In our main technical result (Theorem 3 in Section 4), we show that for every k^{th}-order tree process $\mathcal{P}$ with $\mathsf{Prob}_{\mathcal{P}}(t) > 0$, we have that $|B(\mathcal{G}_t)|$ is upper-bounded by $-\log_2 \mathsf{Prob}_{\mathcal{P}}(t)$ plus certain lower-order terms, where B is a binary TSLP-encoding (to be defined in the following section) and $t \to \mathcal{G}_t$ is a grammar-based tree compressor (also to be defined in the following section). Our main result (Theorem 4 in Section 4), i.e., that $|B(\mathcal{G}_t)|$ is upper-bounded in terms of the k^{th} order empirical entropy $H_k(t)$ plus lower-order terms, will then easily follow as a corollary from Theorem 3 by applying Theorem 1.

3. Tree straight-line programs and compression of binary trees

We now introduce tree straight-line programs (TSLPs) and use them for the compression of binary trees. The main idea behind TSLPs can be best explained by viewing them as a generalization of DAGs (directed acyclic graphs). A binary tree t can be compressed into a DAG by keeping for several isomorphic subtrees only one copy. Formally, such a DAG D can be represented by a set of rules of the form $A \to a(B, C)$ (resp, $A \to a$). Here $a \in \Sigma$ is a node label and A, B, C are nodes of D. The rule $A \to a(B, C)$ tells us that node A is labelled with a, and its left (right) outgoing edge goes to B (C). The rule $A \to a$ says that node A is labelled with a and has no outgoing edges. These rules formally define a so-called regular tree grammar from which t (and only t) can be derived. The DAG nodes A, B, C, etc. then become nonterminals of the regular tree grammar. There is a unique start nonterminal S from which the whole tree t can be derived.

The main limitation of DAGs for tree compression is that they only allow to exploit repetitions of subtrees. We can obtain a better compression by also exploiting repetitions of subcontexts. Consider a tree $t = a(c[a], c[b])$, where c is a context. The two occurrences of c in t cannot be shared in a DAG. But they can be shared if we extend regular tree grammars by a second type of nonterminal that derives contexts instead of trees. Then we can derive t using the rule $S \to a(C(a), C(b))$, where $C = C(x)$ is a context nonterminal (called a nonterminal of rank one in the formal definition below). Additional rules are needed to derive this context nonterminal C to the context c. If, for instance, $c = f(a, f(a, x)$ then these rules might be $C(x) \to B(B(x))$ and $B(x) \to f(a, x)$. These rules define a so-called context-free tree grammar. This grammar must be acyclic, i.e., from a nonterminal we cannot reach the same nonterminal in an arbitrary number of derivation steps. Moreover, every nonterminal A has exactly one rule with A on the left-hand side; the corresponding right-hand side is denoted with $r(A)$ in the formal definition below.

3.1. **General tree straight-line programs.** Let V be a finite alphabet of symbols, where each symbol $A \in V$ has an associated rank 0 or 1 (we also speak of a ranked alphabet). The elements of V are called *nonterminals*. We assume that V contains at least one nonterminal of rank 0 and that V is disjoint from the set $\Sigma \cup \{x\}$, which are the labels used for binary trees and contexts. We use V_0 (resp., V_1) for the set of nonterminals of rank 0 (resp., of rank 1). The idea is that nonterminals from V_0 (resp., V_1) derive to trees from $\mathcal{T}$ (resp., contexts from $\mathcal{C}$). We denote by $\mathcal{T}_V(\Sigma)$ the set of trees over $\Sigma \cup V$, i.e., each node in a tree $t \in \mathcal{T}_V(\Sigma)$ is labelled with a symbol from $\Sigma \cup V$ such that nodes labelled by symbols from Σ have zero or two children and if a node is labelled by a symbol from V, then the number of children of this node corresponds to the rank of its label (a formal definition follows). With $\mathcal{C}_V(\Sigma)$ we denote the corresponding set of all contexts, i.e., the set of trees over $\Sigma \cup \{x\} \cup V$, where the parameter symbol x occurs exactly once and at a leaf position. Formally, we define $\mathcal{T}_V(\Sigma)$ and $\mathcal{C}_V(\Sigma)$ as the smallest sets of formal expressions with the following conditions, where here and in the rest of the paper we use the abbreviations $\mathcal{T}_V$ for $\mathcal{T}_V(\Sigma)$ and $\mathcal{C}_V$ for $\mathcal{C}_V(\Sigma)$:

- $\Sigma \cup V_0 \subseteq \mathcal{T}_V$ and $x \in \mathcal{C}_V$,
- if $a \in \Sigma$, $A \in V_1$ and $t_1, t_2 \in \mathcal{T}_V$ then $A(t_1), a(t_1, t_2) \in \mathcal{T}_V$, and
- if $a \in \Sigma$, $A \in V_1$, $s \in \mathcal{C}_V$ and $t \in \mathcal{T}_V$ then $A(s), a(s, t), a(t, s) \in \mathcal{C}_V$.

If e.g. $\Sigma = \{a, b\}$, $V_0 = \{A\}$ and $V_1 = \{B\}$, then $B(a(b(A, b), B(a))) \in \mathcal{T}_V$ and $B(a(b(A, b), B(x))) \in \mathcal{C}_V$ as depicted in Figure 2. Note that $\mathcal{T}(\Sigma) \subseteq \mathcal{T}_V(\Sigma)$ and $\mathcal{C}(\Sigma) \subseteq \mathcal{C}_V(\Sigma)$.

A *tree straight-line program* $\mathcal{G}$, or *TSLP* for short, is a tuple (V, A_0, r), where $A_0 \in V_0$ is the start nonterminal and $r : V \to (\mathcal{T}_V \cup \mathcal{C}_V)$ is a function which assigns to each nonterminal its unique right-hand side. It is required that if $A \in V_0$ (resp., $A \in V_1$), then $r(A) \in \mathcal{T}_V$ (resp., $r(A) \in \mathcal{C}_V$). Furthermore, the binary relation $\{(A, B) \in V \times V \mid B \text{ occurs in } r(A)\}$ has to be acyclic. These conditions ensure that exactly one tree is derived from the start nonterminal A_0 by using the rewrite rules $A \to r(A)$ for $A \in V$. To define this formally, we define $\mathsf{val}_{\mathcal{G}}(t) \in \mathcal{T}$ for $t \in \mathcal{T}_V$ and $\mathsf{val}_{\mathcal{G}}(c) \in \mathcal{C}$ for $c \in \mathcal{C}_V$ inductively by the following rules:

- $\mathsf{val}_{\mathcal{G}}(a) = a$ for $a \in \Sigma$ and $\mathsf{val}_{\mathcal{G}}(x) = x$,

FIGURE 2. Elements of $\mathcal{T}_V$ (left) and $\mathcal{C}_V$ (right), where $a, b \in \Sigma$, $A \in V_0$ and $B \in V_1$.

- $\mathsf{val}_{\mathcal{G}}(a(s_1, s_2)) = a(\mathsf{val}_{\mathcal{G}}(s_1), \mathsf{val}_{\mathcal{G}}(s_2))$ for $a \in \Sigma$ and $s_1, s_2 \in \mathcal{T}_V \cup \mathcal{C}_V$ (and $s_1 \in \mathcal{T}_V$ or $s_2 \in \mathcal{T}_V$ since there is at most one parameter in $a(s_1, s_2)$),
- $\mathsf{val}_{\mathcal{G}}(A) = \mathsf{val}_{\mathcal{G}}(r(A))$ for $A \in V_0$,
- $\mathsf{val}_{\mathcal{G}}(A(s)) = \mathsf{val}_{\mathcal{G}}(r(A))[\mathsf{val}_{\mathcal{G}}(s)]$ for $A \in V_1$ and $s \in \mathcal{T}_V \cup \mathcal{C}_V$ (note that $\mathsf{val}_{\mathcal{G}}(r(A))$ is a context c, so we can build $c[\mathsf{val}_{\mathcal{G}}(s)]$).

The tree defined by $\mathcal{G}$ is $\mathsf{val}(\mathcal{G}) = \mathsf{val}_{\mathcal{G}}(A_0) \in \mathcal{T}$.

Example 4. *Let* $\Sigma = \{a, b\}$ *and* $\mathcal{G} = (\{A_0, A_1, A_2\}, A_0, r)$ *be a TSLP such that* $A_0, A_1 \in V_0, A_2 \in V_1$ *and*

$$r(A_0) = a(A_1, A_2(b)),\ r(A_1) = A_2(A_2(b)),\ r(A_2) = b(x, a).$$

We get $\mathsf{val}_{\mathcal{G}}(A_2) = b(x, a)$, $\mathsf{val}_{\mathcal{G}}(A_1) = b(b(b, a), a)$ *and* $\mathsf{val}(\mathcal{G}) = \mathsf{val}_{\mathcal{G}}(A_0) = a(b(b(b, a), a), b(b, a))$.

3.2. **Tree straight-line programs in normal form.** In this section, we introduce TSLPs in a certain normal form. The properties of this normal form make the binary encoding of TSLPs in Section 3.3 easier.

A TSLP $\mathcal{G} = (V, A_0, r)$ is in *normal form* if the following conditions hold:

- $V = \{A_0, A_1, \ldots, A_{m-1}\}$ for some $m \in \mathbb{N}$, $m \geq 1$.
- For every $A_i \in V_0$, the right-hand side $r(A_i)$ is an expression of the form $A_j(\alpha)$, where $A_j \in V_1$ and $\alpha \in V_0 \cup \Sigma$.
- For every $A_i \in V_1$ the right-hand side $r(A_i)$ is an expression of the form $A_j(A_k(x))$, $a(\alpha, x)$, or $a(x, \alpha)$, where $A_j, A_k \in V_1$, $a \in \Sigma$ and $\alpha \in V_0 \cup \Sigma$.
- For every $A_i \in V$ define the word $\rho(A_i) \in (V \cup \Sigma)^*$ as follows:

$$\rho(A_i) = \begin{cases} A_j\alpha & \text{if } r(A_i) = A_j(\alpha) \\ A_jA_k & \text{if } r(A_i) = A_j(A_k(x)) \\ a\alpha & \text{if } r(A_i) = a(\alpha, x) \text{ or } a(x, \alpha) \end{cases}$$

  Let $\rho_{\mathcal{G}} = \rho(A_0)\rho(A_1)\cdots\rho(A_{m-1}) \in (\Sigma \cup \{A_1, A_2, \ldots, A_{m-1}\})^*$. Then we require that $\rho_{\mathcal{G}}$ is of the form $\rho_{\mathcal{G}} = A_1u_1A_2u_2\cdots A_{m-1}u_{m-1}$ with $u_i \in (\Sigma \cup \{A_1, A_2, \ldots, A_i\})^*$.
- $\mathsf{val}_{\mathcal{G}}(A_i) \neq \mathsf{val}_{\mathcal{G}}(A_j)$ for $i \neq j$

We also allow the TSLP $\mathcal{G}_a = (\{A_0\}, A_0, A_0 \mapsto a)$ for every $a \in \Sigma$ in order to get the singleton tree a. In this case, we set $\rho_{\mathcal{G}_a} = \rho(A_0) = a$.

Let $\mathcal{G} = (V, A_0, r)$ be a TSLP in normal form with $V = \{A_0, A_1, \ldots, A_{m-1}\}$ for the further definitions. We define the size of $\mathcal{G}$ as $|\mathcal{G}| = |V| = m$. Thus $2|\mathcal{G}|$ is the length of $\rho_{\mathcal{G}}$. Let $\omega_{\mathcal{G}}$ be the word obtained from $\rho_{\mathcal{G}}$ by removing the first (i.e., left-most) occurrence of A_i from $\rho_{\mathcal{G}}$ for every $1 \leq i \leq m-1$. Thus,

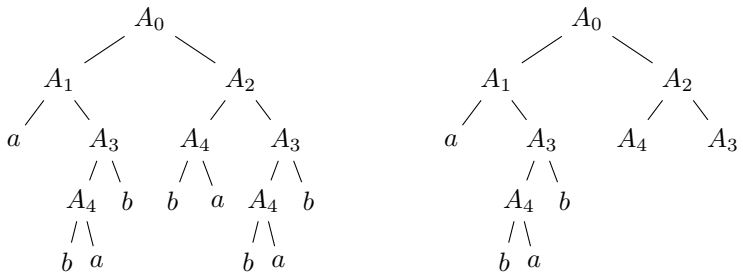

FIGURE 3. The derivation tree $T_{\mathcal{G}}$ of the TSLP from Example 6 (left) and an initial subtree T' of $T_{\mathcal{G}}$ (right).

if $\rho_{\mathcal{G}} = A_1 u_1 A_2 u_2 \cdots A_{m-1} u_{m-1}$ with $u_i \in (\Sigma \cup \{A_1, A_2, \ldots, A_i\})^*$, then $\omega_{\mathcal{G}} = u_1 u_2 \cdots u_{m-1}$. Note that $|\omega_{\mathcal{G}}| = |\rho_{\mathcal{G}}| - m + 1 = m + 1$. The *entropy* $H(\mathcal{G})$ of the normal form TSLP $\mathcal{G}$ is defined as the empirical unnormalized entropy of the word $\omega_{\mathcal{G}}$ (see (6)):

$$H(\mathcal{G}) = H(\omega_{\mathcal{G}}).$$

Example 5. *Let $\Sigma = \{a, b\}$ and $\mathcal{G} = (\{A_0, A_1, A_2, A_3, A_4\}, A_0, r)$ be the normal form TSLP with $A_0, A_2, A_3 \in V_0, A_1, A_4 \in V_1$ and*

$$r(A_0) = A_1(A_2(x)),\ r(A_1) = a(x, A_3),\ r(A_2) = A_4(A_3(x)),$$
$$r(A_3) = A_4(b),\ r(A_4) = b(x, a).$$

We have $\mathsf{val}(\mathcal{G}) = a(b(b(b, a), a), b(b, a))$, $\rho_{\mathcal{G}} = A_1 A_2 a A_3 A_4 A_3 A_4 bba$ ($u_1 = u_3 = \varepsilon$, $u_2 = a$, $u_4 = A_3 A_4 bba$), $|\mathcal{G}| = 5$ and $\omega_{\mathcal{G}} = a A_3 A_4 bba$.

The *derivation tree* $T_{\mathcal{G}}$ of the normal form TSLP $\mathcal{G}$ is a binary tree with node labels from $V \cup \Sigma$. The root is labelled with A_0. Nodes labelled with a symbol from Σ are the leaves of $T_{\mathcal{G}}$. A node v that is labelled with a nonterminal A_i has $|\rho(A_i)| = 2$ many children. If $\rho(A_i) = \alpha\beta$ with $\alpha, \beta \in V \cup \Sigma$, then the left child of v is labelled with α and the right child is labelled with β. For every node u of $T_{\mathcal{G}}$ we define the tree or context $s_u = \mathsf{val}_{\mathcal{G}}(\alpha)$ where $\alpha \in V \cup \Sigma$ is the label of u. If $\alpha \in V_0 \cup \Sigma$ then $s_u \in \mathcal{T}$ and if $\alpha \in V_1$ then $s_u \in \mathcal{C}$. An *initial subtree* of the derivation tree $T_{\mathcal{G}}$ is a tree that can be obtained from $T_{\mathcal{G}}$ as follows: Take a subset U of the nodes of $T_{\mathcal{G}}$ and remove from $T_{\mathcal{G}}$ all proper descendants of nodes from U, i.e., all nodes that are located strictly below a node from U.

Example 6. *Let $\mathcal{G}$ be the normal form TSLP from Example 5. The derivation tree $T_{\mathcal{G}}$ is shown in Figure 3 on the left; an initial subtree T' of it is shown on the right.*

We obtain the following lemma with respect to initial subtrees of a derivation tree $T_{\mathcal{G}}$, which will be needed in order to prove our technical main lemma (Lemma 7 in Section 4):

Lemma 4. *Let $\mathcal{G}$ be a TSLP in normal form with $t = \mathsf{val}(\mathcal{G})$. Let T' be an initial subtree of $T_{\mathcal{G}}$ and let $v_1, \ldots, v_l$ be the sequence of all leaves of T' (in left-to-right order). Then $2|t| \geq \sum_{i=1}^{l} |s_{v_i}|$.*

Proof. Let u be a node of $T_{\mathcal{G}}$ and let T_u be the subtree of $T_{\mathcal{G}}$ rooted in u. Then, the nodes of s_u are in a one-to-one correspondence with the leaves of T_u, that is, if $s_u \in \mathcal{T}$, we have $2|s_u| - 1 = |T_u|$ and if $s_u \in \mathcal{C}$, we have $2|s_u| = |T_u|$ (recall that

$|T_u|$ is the number of leaves of T_u). Thus, $2|s_u| - 1 \leq |T_u|$. Since T' is an initial subtree of $T_{\mathcal{G}}$ we get $2|t| - 1 = 2|\mathsf{val}(\mathcal{G})| - 1 = |T_{\mathcal{G}}| = \sum_{i=1}^{l} |T_{v_i}| \geq \sum_{i=1}^{l} (2|s_{v_i}| - 1)$. Since $|s_{v_i}| \geq 1$ we get $2|t| \geq \sum_{i=1}^{l} 2|s_{v_i}| - l + 1 \geq \sum_{i=1}^{l} |s_{v_i}| + 1$ and the statement follows. □

A *grammar-based tree compressor* is an algorithm ψ that produces for a given tree $t \in \mathcal{T}$ a TSLP $\mathcal{G}_t$ in normal form such that $t = \mathsf{val}(\mathcal{G}_t)$. It is not hard to show that every TSLP can be transformed with a linear size increase into a normal form TSLP that derives the same tree. For example, the TSLP from Example 4 is transformed into the normal form TSLP described in Example 5. We will not use this fact, since all we need is the following theorem from [12] (recall that $\hat{\sigma} = \max\{2, \sigma\}$):

Theorem 2. *There exists a grammar-based compressor ψ (working in linear time) with $\max_{t \in \mathcal{T}_n} |\mathcal{G}_t| \leq \mathcal{O}(n / \log_{\hat{\sigma}} n)$.*

3.3. **Binary coding of TSLPs in normal form.** In this section we fix a binary encoding for normal form TSLPs. This binary encoding is a straightforward extension of the one for TSLPs producing unlabelled binary trees [14] (which in turn is based on the encoding for SLPs from [22] and the encoding of DAGs from [38]). We only have to incorporate node labels into the encoding from [14].

Let $\mathcal{G} = (V, A_0, r)$ be a TSLP in normal form with $m = |V| = |\mathcal{G}|$ nonterminals. We define the type $\mathsf{type}(A_i) \in \{0, 1, 2, 3\}$ of a nonterminal $A_i \in V$ as follows:

$$\mathsf{type}(A_i) = \begin{cases} 0 & \text{if } r(A_i) = A_j(\alpha) \text{ for some } A_j \in V_1, \alpha \in V_0 \cup \Sigma \\ 1 & \text{if } r(A_i) = A_j(A_k(x)) \text{ for some } A_j, A_k \in V_1 \\ 2 & \text{if } r(A_i) = a(\alpha, x) \text{ for some } \alpha \in V_0 \cup \Sigma, a \in \Sigma \\ 3 & \text{if } r(A_i) = a(x, \alpha) \text{ for some } \alpha \in V_0 \cup \Sigma, a \in \Sigma \end{cases}$$

We define the binary word $B(\mathcal{G}) = w_0 w_1 w_2 w_3 w_4$, where the words $w_i \in \{0,1\}^+$, $0 \leq i \leq 4$, are defined as follows:

- $w_0 = 0^{m-1}1$
- $w_1 = a_0 b_0 a_1 b_1 \cdots a_{m-1} b_{m-1}$, where $a_j b_j$ is the 2-bit binary encoding of $\mathsf{type}(A_j)$. Note that $|w_1| = 2m$.
- Let $\rho_{\mathcal{G}} = A_1 u_1 A_2 u_2 \cdots A_{m-1} u_{m-1}$ with $u_i \in (\Sigma \cup \{A_1, A_2, \ldots, A_i\})^*$. Then $w_2 = 10^{|u_1|}10^{|u_2|} \cdots 10^{|u_{m-1}|}$. Note that $|w_2| = 2m$.
- For $1 \leq i \leq m-1$ let $k_i = |\rho_{\mathcal{G}}|_{A_i} \geq 1$ be the number of occurrences of the nonterminal A_i in the word $\rho_{\mathcal{G}}$. Moreover, fix a total ordering on Σ. For $1 \leq i \leq \sigma$, let a_i denote the i^{th} symbol in Σ according to this ordering and let $l_i = |\rho_{\mathcal{G}}|_{a_i} \geq 0$ be the number of occurences of the symbol a_i in the word $\rho_{\mathcal{G}}$. Then $w_3 = 0^{k_1 - 1}10^{k_2 - 1}1 \cdots 0^{k_{m-1} - 1}10^{l_1}10^{l_2}1 \cdots 0^{l_\sigma}1$. Note that $|w_3| = 2m + \sigma$.
- The word w_4 encodes the word $\omega_{\mathcal{G}}$ using the well-known enumerative encoding [5]. Every nonterminal A_i, $1 \leq i \leq m-1$, has $\eta(A_i) := k_i - 1$ occurrences in $\omega_{\mathcal{G}}$. Every symbol $a_i \in \Sigma$, $1 \leq i \leq \sigma$, has $\eta(a_i) = l_i$ occurences in $\omega_{\mathcal{G}}$. Let S be the set of words over the alphabet $\Sigma \cup \{A_1, \ldots, A_{m-1}\}$ with $\eta(a_i)$ occurrences of $a_i \in \Sigma$ ($1 \leq i \leq \sigma$) and $\eta(A_i)$ occurrences of A_i ($1 \leq i \leq m-1$). Hence,

$$|S| = \frac{(m+1)!}{\prod_{i=1}^{\sigma} \eta(a_i)! \prod_{i=1}^{m-1} \eta(A_i)!}. \tag{15}$$

Let $v_0, v_1, \ldots, v_{|S|-1}$ be the lexicographic enumeration of the words from S with respect to the alphabet order $a_1, \ldots, a_\sigma, A_1, \ldots, A_{m-1}$. Then w_4 is the binary encoding of the unique index i such that $\omega_\mathcal{G} = v_i$, where $|w_4| = \lceil \log_2 |S| \rceil$ (leading zeros are added to the binary encoding of i to obtain the length $\lceil \log_2 |S| \rceil$).

Example 7. *Consider the normal form TSLP $\mathcal{G}$ from Example 5. We have $w_0 =$ 00001, $w_1 =$ 0011000011, $w_2 =$ 1101100000 and $w_3 =$ 110101001001. To compute w_4, note first that there are $|S| = 180$ words with two occurrences of a and b and one occurrence of A_3 and A_4. It follows that $|w_4| = \lceil \log_2(180) \rceil = 8$. Furthermore, with the canonical ordering on $\Sigma = \{a, b\}$, the order of the alphabet is a, b, A_3, A_4. The word $\omega_\mathcal{G} = aA_3A_4bba$ is the lexicographically largest word in S starting with aA_3. There are 132 words in S that are lexicographically larger than aA_3A_4bba, namely all words in S that start with b (60 words), A_3 (30 words), A_4 (30 words), or aA_4 (12 words). Hence $\omega_\mathcal{G} = aA_3A_4bba$ is the 48^{th} word in S in lexicographic order, i.e., $\omega_G = v_{47}$ and thus $w_4 =$ 00101111.*

The following lemma generalizes a result from [14]:

Lemma 5. *The set of code words $B(\mathcal{G})$, where $\mathcal{G}$ ranges over all TSLPs in normal form, is a prefix code.*

Proof. Let $B(\mathcal{G}) = w_0w_1w_2w_3w_4$ with w_i defined as above. We show how to recover the TSLP $\mathcal{G}$, given the alphabet Σ and the ordering on Σ. From w_0 we can determine $m = |V|$ and the factors w_1, w_2, and w_3 of $B(\mathcal{G})$. Hence, we can determine the type of every nonterminal from w_1. The types allow to compute $\mathcal{G}$ from the word ρ_G. Hence, it remains to determine $\rho_\mathcal{G}$. To compute $\rho_\mathcal{G}$ from w_2, one only needs $\omega_\mathcal{G}$. For this, one determines the frequencies $\eta(A_1), \ldots, \eta(A_{m-1}), \eta(a_1), \ldots, \eta(a_\sigma)$ of the symbols in $\omega_\mathcal{G}$ from w_3. Using these frequencies one computes the size $|S|$ from (15) and the length $\lceil \log_2 |S| \rceil$ of w_4. From w_4, one can finally compute $\omega_\mathcal{G}$. □

Note that $|B(\mathcal{G})| \leq 7|\mathcal{G}| + \sigma + |w_4|$. By using the well-known bound on the code length of enumerative encoding [6, Theorem 11.1.3], we get the following bound, which extends [14, Lemma 11] to node-labelled binary trees:

Lemma 6. *For the length of the binary coding $B(\mathcal{G})$ we have*

$$|B(\mathcal{G})| \leq \mathcal{O}(|\mathcal{G}|) + \sigma + H(\mathcal{G}).$$

Intuitively, by Lemma 5, we can uniquely recover a binary tree t from $B(\mathcal{G}_t)$, where $\mathcal{G}_t$ is a TSLP for t. By Lemma 6, we find that in order to upper-bound $|B(\mathcal{G})|$ in terms of the k^{th}-order empirical entropy (plus lower-order terms), we can focus on the entropy $H(\mathcal{G})$ of the grammar $\mathcal{G}$: Our technical main result (Lemma 7 in the following section) will provide a suitable upper-bound on $H(\mathcal{G})$. We furthermore remark that a corresponding result (corresponding to Lemma 6) exists for strings ([22, Lemma 8] and [33, Lemma 2]).

4. Entropy bounds for binary encoded TSLPs

For this section we fix a grammar-based tree compressor $\psi : t \mapsto \mathcal{G}_t$ such that $\max_{t \in \mathcal{T}_n} |\mathcal{G}_t| \in \mathcal{O}(n/\log_{\hat{\sigma}} n)$; see Theorem 2. Let $\gamma > 0$ be a concrete constant such that

$$|\mathcal{G}_t| \leq \frac{\gamma n}{\log_{\hat{\sigma}} n} \tag{16}$$

for every tree $t \in \mathcal{T}_n$ and n large enough. We allow that the alphabet size σ grows with n, i.e., $\sigma = \sigma(n)$ is a function in the tree size n such that $1 \leq \sigma(n) \leq 2n-1$ (a binary tree $t \in \mathcal{T}_n$ has $2n-1$ nodes).

Our technical main result is the following bound on the entropy $H(\mathcal{G}_t)$. A similar bound (but for a different class of probability distributions on trees) is stated in [14, Lemma 13].

Lemma 7. *Let $k \geq 0$, $t \in \mathcal{T}_n$ with $n \geq 2$ and let $\mathcal{P} = (P_w)_{w \in \mathcal{L}_k}$ be a k^{th}-order tree process with $\mathsf{Prob}_{\mathcal{P}}(t) > 0$. We have*

$$H(\mathcal{G}_t) \leq -\log_2 \mathsf{Prob}_{\mathcal{P}}(t) + \mathcal{O}\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + \mathcal{O}\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right).$$

Proof. Let $m = |\mathcal{G}_t| = |V|$ be the size of $\mathcal{G}_t$. Let $T = T_{\mathcal{G}_t}$ be the derivation tree of $\mathcal{G}_t$. We define an initial subtree T' as follows: If v_1 and v_2 are non-leaf nodes of T that are labelled with the same nonterminal and v_1 comes before v_2 in preorder (depth-first left-to-right), then we remove from T all proper descendants of v_2. Thus, for every $A_i \in V$ there is exactly one non-leaf node in T' that is labelled with A_i. For the TSLP from Example 5, the tree T' is shown in Figure 3 on the right. We now use the tree T' in order to define a factorization of the tree t into several subtrees, subcontexts and inner nodes. A similar factorization is also used in [14, proof of Lemma 7].

Recall the definition of the words $\rho_{\mathcal{G}_t}$ and $\omega_{\mathcal{G}_t}$ from Section 3.2. The word $\rho_{\mathcal{G}_t}$ can be obtained by writing down for every node v of T' the labels of v's children and then concatenating these labels. Moreover, the word $\omega_{\mathcal{G}_t}$ is obtained by writing down (in the right order) the labels of the leaves of T'. Note that T' has m non-leaf nodes and $m+1$ leaves. Let $v_1, v_2, \ldots, v_{m+1}$ be the sequence of all leaves of T' (w.l.o.g. in preorder) and let $\alpha_i \in \Sigma \cup \{A_1, \ldots, A_{m-1}\}$ be the label of v_i. Let $\overline{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_{m+1})$. Then $\overline{\alpha}$ is a permutation of $\omega_{\mathcal{G}_t}$. We therefore have $|\omega_{\mathcal{G}_t}|_\alpha = |\overline{\alpha}|_\alpha$ for every $\alpha \in \Sigma \cup \{A_1, \ldots, A_{m-1}\}$. Hence, $p_{\overline{\alpha}}$ and $p_{\omega_{\mathcal{G}_t}}$ are the same empirical distributions. For the TSLP from Example 5 we get $\overline{\alpha} = (a, b, a, b, A_4, A_3)$. Let $s_i = \mathsf{val}_{\mathcal{G}_t}(\alpha_i) \in \mathcal{T} \cup (\mathcal{C} \setminus \{x\})$. Since $\mathsf{val}_{\mathcal{G}_t}(A_i) \neq \mathsf{val}_{\mathcal{G}_t}(A_j)$ for all $i \neq j$ ($\mathcal{G}_t$ is in normal form) and $\mathsf{val}_{\mathcal{G}_t}(A_i) \notin \Sigma$ for all i (this holds for every normal form TSLP that produces a tree of size at least two), the tuple $\overline{s} = (s_1, s_2, \ldots, s_{m+1})$ satisfies for all $1 \leq i \leq m+1$:

$$p_{\omega_{\mathcal{G}_t}}(\alpha_i) = p_{\overline{s}}(s_i). \tag{17}$$

We define from $\mathcal{P}$ for every $z \in \mathcal{L}_k$ a modified tree process $\mathcal{P}_z = (P_{z,w})_{w \in \mathcal{L}}$ by setting

$$P_{z,w}(\tilde{a}) = P_{\ell_k(zw)}(\tilde{a}) \tag{18}$$

for all $\tilde{a} \in \Sigma \times \{0, 2\}$. Note that the k^{th}-order tree process $\mathcal{P}$ is obtained for $z = (\Box 0)^k$ for the fixed padding symbol $\Box \in \Sigma$. We define a mapping $\tau : \mathcal{T} \cup \mathcal{C} \to [0, 1]$ by

$$\tau(s) = \begin{cases} 1 & \text{if } s \in \mathcal{T}_1 = \Sigma \\ \max_{z \in \mathcal{L}_k} \mathsf{Prob}_{\mathcal{P}_z}(s) & \text{if } s \in (\mathcal{T} \cup \mathcal{C}) \setminus \mathcal{T}_1 \end{cases} \tag{19}$$

for every $s \in \mathcal{T} \cup \mathcal{C}$. Thus, for every $s \in (\mathcal{T} \cup \mathcal{C}) \setminus \mathcal{T}_1$, the function τ maximizes the values of the function $\mathsf{Prob}_{\mathcal{P}}$ associated with the k^{th}-order tree process $\mathcal{P} =$

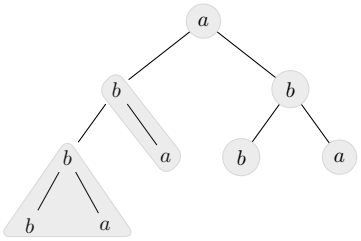

FIGURE 4. The tree $\mathsf{val}(\mathcal{G})$ of the TSLP from Example 5. The canonical occurrences of the trees/contexts or inner nodes in $(a, b, a, b, \mathsf{val}_{\mathcal{G}}(A_4), \mathsf{val}_{\mathcal{G}}(A_3)) = (a, b, a, b, b(x, a), b(b, a))$ used in the proof of Lemma 7 are highlighted.

$(P_w)_{w\in\mathcal{L}_k}$ by choosing an optimal k-history for the nodes of s whose history is of length smaller than $2k$. We show that τ satisfies

$$\tau(t) \leq \prod_{i=1}^{m+1} \tau(s_i). \tag{20}$$

In order to prove (20), first note that by definition of the tree/context s_u, for each node u of the derivation tree T, the tree/context s_u corresponds to a subtree/subcontext or a single inner node of the binary tree t. We define a function χ which maps a node u of the derivation tree T to a node $\chi(u) \in V(t) \subseteq \{0, 1\}^*$: Intuitively, $\chi(u)$ is the root of the subtree/subcontext, respectively, the inner node of t which corresponds to s_u. Formally, χ is defined inductively as follows: For the root node u of T, we set $\chi(u) = \varepsilon$. Furthermore, let u be a non-leaf node of T which is labelled with the non-terminal A_i and for which $\chi(u)$ has been defined. Let u_1 be the left child and u_2 be the right child of u in T. We define $\chi(u_1) = \chi(u)$. The node $\chi(u_2)$ is defined as follows:

(i) If $r(A_i) = A_j(\alpha)$ with $A_j \in V_1$ and $\alpha \in V \cup \Sigma$, then we set $\chi(u_2) = \chi(u)\omega(s_{u_1})$ (recall that $\omega(s_{u_1}) \neq \varepsilon$ is the position of the parameter x in the context $s_{u_1} = \mathsf{val}_{\mathcal{G}}(A_j)$).
(ii) If $r(A_i) = a(\alpha, x)$ (respectively, $r(A_i) = a(x, \alpha)$) for $a \in \Sigma$ and $\alpha \in \Sigma \cup V_0$, then we define $\chi(u_2) = \chi(u)0$ (respectively, $\chi(u_2) = \chi(u)1$).

This yields a well-defined function χ mapping a node u of T to a node $\chi(u) \in V(t)$. Let us define

$$V_u = \{\chi(u)v \mid v \in V(s_u)\} \subseteq V(t).$$

Then, the mapping

$$V(s_u) \ni v \mapsto \chi(u)v \in V_u \tag{21}$$

is bijective. The definition of the sets V_u implies that if two nodes u and v of T are not in an ancestor-descendant relationship, then $V_u \cap V_v = \emptyset$. Since the nodes $v_1, \ldots, v_{m+1}$ are the leaves of the initial subtree T' and hence not in an ancestor-descendant relationship, the sets $V_i := V_{v_i}$ are disjoint subsets of $V(t)$. For the TSLP from Example 5, the node sets V_1, V_2, V_3, V_4, V_5 and V_6 corresponding to the six leaves of the initial subtree depicted in Figure 3 (right) are shown in Figure 4. Note that if $s_i \notin \mathcal{T}_1$ then the bijection from (21) also preserves the λ-mapping in

the following sense:

$$\lambda_t(\chi(v_i)w) = \lambda_{s_i}(w) \tag{22}$$

for every $w \in V(s_i)$. However, if $s_i \in \mathcal{T}_1$ then this statement can be wrong since the number of children is not preserved in general: If $s_i \in \mathcal{T}_1$, then s_i might correspond to a single inner node of t. In this case, we have $V_i = \{\chi(v_i)\}$, $V(s_i) = \{\varepsilon\}$ and $\lambda_t(\chi(v_i)) = (a, 2)$ for some $a \in \Sigma$, but $\lambda_{s_i}(\varepsilon) = (a, 0)$. For example, in the TSLP from Example 5, the left-most leaf node of its initial subtree depicted in Figure 3 corresponds to the root node of the tree $\mathsf{val}(\mathcal{G})$ (see Figure 4). We define

$$\mathcal{I} := \{i \in \{1, \ldots, m+1\} \mid s_i \notin \mathcal{T}_1\}.$$

In our running example, we have $(s_1, s_2, s_3, s_4, s_5, s_6) = (a, b, a, b, b(x, a), b(b, a))$ and hence $\mathcal{I} := \{5, 6\}$.

The history $h(\chi(v_i)w)$ of a node $\chi(v_i)w \in V_i$ with $w \in V(s_i)$ in the tree t is the concatenation of the history $h(\chi(v_i))$ of $\chi(v_i)$ in t and the history $h(w)$ of w in the tree/context s_i. Thus, if $i \in \mathcal{I}$, we have

$$\begin{aligned}
\max_{z \in \mathcal{L}_k} \prod_{v \in V_i} P_{\ell_k(zh(v))}(\lambda_t(v)) &= \max_{z \in \mathcal{L}_k} \prod_{w \in V(s_i)} P_{\ell_k(zh(\chi(v_i))h(w))}(\lambda_t(\chi(v_i)w)) \\
&\overset{(22)}{=} \max_{z \in \mathcal{L}_k} \prod_{w \in V(s_i)} P_{\ell_k(zh(\chi(v_i))h(w))}(\lambda_{s_i}(w)) \\
&\leq \max_{z \in \mathcal{L}_k} \prod_{w \in V(s_i)} P_{\ell_k(zh(w))}(\lambda_{s_i}(w)).
\end{aligned} \tag{23}$$

For the inequality in the last line, note that every k-history $\ell_k(zh(\chi(v_i))h(w))$ for $z \in \mathcal{L}_k$ is also of the form $\ell_k(z'h(w))$ for some $z' \in \mathcal{L}_k$.

We can now show (20). Since $t \in \mathcal{T}_n$ with $n \geq 2$ we have

$$\begin{aligned}
\tau(t) &\overset{(19)}{=} \max_{z \in \mathcal{L}_k} \mathsf{Prob}_{\mathcal{P}_z}(t) \\
&\overset{(18)}{=} \max_{z \in \mathcal{L}_k} \prod_{v \in V(t)} P_{\ell_k(zh(v))}(\lambda_t(v)) \\
&\leq \max_{z \in \mathcal{L}_k} \prod_{i \in \mathcal{I}} \prod_{v \in V_i} P_{\ell_k(zh(v))}(\lambda_t(v)) \quad (\text{since } P_{\ell_k(zh(v))}(\lambda(v)) \leq 1 \text{ for } v \in V(t)) \\
&\leq \prod_{i \in \mathcal{I}} \max_{z \in \mathcal{L}_k} \prod_{v \in V_i} P_{\ell_k(zh(v))}(\lambda_t(v)) \\
&\overset{(23)}{\leq} \prod_{i \in \mathcal{I}} \max_{z \in \mathcal{L}_k} \prod_{w \in V(s_i)} P_{\ell_k(zh(w))}(\lambda_{s_i}(w)) \\
&= \prod_{i=1}^{m+1} \tau(s_i) \quad (\text{since } \tau(s_i) = 1 \text{ for } i \notin \mathcal{I}).
\end{aligned}$$

Next, we define the function $\xi : \mathcal{T} \cup \mathcal{C} \setminus \{x\} \to [0, 1]$ as follows:

$$\xi(s) = \begin{cases} 2^{-(k+2)} \sigma^{-(k+1)} \tau(s) & \text{if } s \in \mathcal{T} \\ \frac{6}{\pi^2} 2^{-(k+1)} \sigma^{-k} \frac{\tau(s)}{|s|^2(|s|+1)} & \text{if } s \in \mathcal{C} \setminus \{x\}. \end{cases}$$

We get

$$\begin{aligned}
\sum_{s\in\mathcal{T}\cup\mathcal{C}\setminus\{x\}} \xi(s) &= 2^{-(k+2)}\sigma^{-(k+1)}\sum_{s\in\mathcal{T}}\tau(s) + \frac{6}{\pi^2}2^{-(k+1)}\sigma^{-k}\sum_{s\in\mathcal{C}\setminus\{x\}}\frac{\tau(s)}{|s|^2(|s|+1)} \\
&= 2^{-(k+2)}\sigma^{-(k+1)}\Bigg(\sum_{s\in\mathcal{T}\setminus\mathcal{T}_1}\max_{z\in\mathcal{L}_k}\mathsf{Prob}_{\mathcal{P}_z}(s) + \sum_{s\in\mathcal{T}_1}1\Bigg) + \\
&\quad \frac{6}{\pi^2}2^{-(k+1)}\sigma^{-k}\sum_{r\geq 1}\frac{1}{r^2(r+1)}\sum_{s\in\mathcal{C}_r}\max_{z\in\mathcal{L}_k}\mathsf{Prob}_{\mathcal{P}_z}(s) \\
&\leq 2^{-(k+2)}\sigma^{-(k+1)}\Bigg(\sum_{z\in\mathcal{L}_k}\sum_{s\in\mathcal{T}\setminus\mathcal{T}_1}\mathsf{Prob}_{\mathcal{P}_z}(s) + \sigma\Bigg) + \\
&\quad \frac{6}{\pi^2}2^{-(k+1)}\sigma^{-k}\sum_{z\in\mathcal{L}_k}\sum_{r\geq 1}\frac{1}{r^2(r+1)}\sum_{s\in\mathcal{C}_r}\mathsf{Prob}_{\mathcal{P}_z}(s) \\
&\overset{(*)}{\leq} 2^{-(k+2)}\sigma^{-(k+1)}\left(2^k\sigma^k+\sigma\right) + \frac{6}{\pi^2}2^{-(k+1)}\sigma^{-k}2^k\sigma^k\sum_{r\geq 1}\frac{1}{r^2} \\
&\overset{(**)}{=} 2^{-2}\sigma^{-1} + 2^{-(k+2)}\sigma^{-k} + 1/2 \\
&\leq 1,
\end{aligned}$$

where $(*)$ follows from Lemmas 2 and 3 and $|\mathcal{L}_k| = 2^k\sigma^k$ and $(**)$ follows from the well-known fact that $\sum_{r\geq 1} r^{-2} = \pi^2/6$. In particular, we have $\sum_{s\in\{s_1,\ldots,s_{m+1}\}}\xi(s) \leq 1$. Thus, with Shannon's inequality (7), we obtain:

$$H(\mathcal{G}_t) = H(\omega_{\mathcal{G}_t}) = \sum_{i=1}^{m+1} -\log_2 p_{\omega_{\mathcal{G}_t}}(\alpha_i) \overset{(17)}{=} \sum_{i=1}^{m+1} -\log_2 p_{\overline{s}}(s_i) \leq \sum_{i=1}^{m+1} -\log_2 \xi(s_i).$$

With $\mathcal{I}_0 = \{i \mid 1 \leq i \leq m+1, s_i \in \mathcal{T}\}$ and $\mathcal{I}_1 = \{i \mid 1 \leq i \leq m+1, s_i \in \mathcal{C}\}$ we obtain

$$\begin{aligned}
H(\mathcal{G}_t) &\leq \sum_{i\in\mathcal{I}_0} -\log_2\xi(s_i) + \sum_{i\in\mathcal{I}_1} -\log_2\xi(s_i) \\
&= \sum_{i\in\mathcal{I}_0} -\log_2\left(2^{-(k+2)}\sigma^{-(k+1)}\tau(s_i)\right) + \\
&\quad \sum_{i\in\mathcal{I}_1} -\log_2\left(\frac{6}{\pi^2}2^{-(k+1)}\sigma^{-k}\frac{\tau(s_i)}{|s_i|^2(|s_i|+1)}\right)
\end{aligned}$$

by definition of ξ. Using logarithmic identities, we get

$$\begin{aligned}
H(\mathcal{G}_t) \leq\ & |\mathcal{I}_0|(k+2) + |\mathcal{I}_0|(k+1)\log_2\sigma - \log_2\left(\prod_{i\in\mathcal{I}_0}\tau(s_i)\right) + \\
& \log_2\left(\frac{\pi^2}{6}\right)|\mathcal{I}_1| + |\mathcal{I}_1|(k+1) + |\mathcal{I}_1|k\log_2\sigma - \log_2\left(\prod_{i\in\mathcal{I}_1}\tau(s_i)\right) + \\
& \sum_{i\in\mathcal{I}_1}\log_2|s_i|^2(|s_i|+1).
\end{aligned}$$

Using $|\mathcal{I}_0| + |\mathcal{I}_1| = m + 1 \leq 2m = 2|\mathcal{G}_t|$, $\log_2(\pi^2/6)|\mathcal{I}_1| \leq |\mathcal{I}_1|$ and $|s_i| + 1 \leq 2|s_i|$, we obtain

$$H(\mathcal{G}_t) \leq 2(k+2)|\mathcal{G}_t| + 2(k+1)|\mathcal{G}_t| \log_2 \sigma - \log_2 \left(\prod_{i=1}^{m+1} \tau(s_i) \right) + \sum_{i=1}^{m+1} \log_2 2|s_i|^3.$$

Equation (20) and $\tau(t) \geq \mathsf{Prob}_{\mathcal{P}}(t)$ yield

$$\begin{aligned} H(\mathcal{G}_t) &\leq 2(k+3)|\mathcal{G}_t| + 2(k+1)|\mathcal{G}_t| \log_2 \sigma - \log_2 \tau(t) + 3 \sum_{i=1}^{m+1} \log_2 |s_i| \\ &\leq -\log_2 \mathsf{Prob}_{\mathcal{P}}(t) + \mathcal{O}(k|\mathcal{G}_t| \log \hat{\sigma} + \sum_{i=1}^{m+1} \log_2 |s_i|). \end{aligned}$$

Let us bound the sum $\sum_{i=1}^{m+1} \log_2 |s_i|$: Using Jensen's inequality and Lemma 4 (which yields $\sum_{i=1}^{m+1} |s_i| \leq 2n$), we get

$$\begin{aligned} \sum_{i=1}^{m+1} \log_2 |s_i| &\leq (m+1) \log_2 \left(\sum_{i=1}^{m+1} \frac{|s_i|}{m+1} \right) \\ &\leq (m+1) \log_2 \left(\frac{2n}{m+1} \right) \\ &\leq 2|\mathcal{G}_t| \log_2 \left(\frac{2n}{|\mathcal{G}_t|} \right) \end{aligned}$$

and thus

$$H(\mathcal{G}_t) \leq -\log_2 \mathsf{Prob}_{\mathcal{P}}(t) + \mathcal{O}\Big(k|\mathcal{G}_t| \log \hat{\sigma} + |\mathcal{G}_t| \log_2 \left(\frac{n}{|\mathcal{G}_t|} \right) \Big). \tag{24}$$

To bound the term $|\mathcal{G}_t| \log_2(n/|\mathcal{G}_t|)$ recall that for n large enough we have $|\mathcal{G}_t| \leq \gamma \cdot n / \log_{\hat{\sigma}} n = \gamma \cdot n \cdot \log \hat{\sigma} / \log n$ by (16). Here, γ is a constant. Since $\sigma \leq 2n - 1$ there is a constant $\gamma' \geq 1$ with $\gamma \cdot n / \log_{\hat{\sigma}} n \leq \gamma' n$. Since for every fixed $z \geq 1$, the function $\phi(x) = x \log_2 \left(\frac{z}{x} \right)$ is monotonically increasing for $0 < x \leq \frac{z}{e}$ (where e is Euler's number), we get

$$|\mathcal{G}_t| \log_2 \left(\frac{n}{|\mathcal{G}_t|} \right) \leq |\mathcal{G}_t| \log_2 \left(\frac{e\gamma' n}{|\mathcal{G}_t|} \right) \leq \frac{\gamma n \log_2 \left(\frac{e\gamma'}{\gamma} \log_{\hat{\sigma}} n \right)}{\log_{\hat{\sigma}} n} \leq \mathcal{O}\Big(\frac{n \log \log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n} \Big).$$

With (24) we get

$$H(\mathcal{G}_t) \leq -\log_2 \mathsf{Prob}_{\mathcal{P}}(t) + \mathcal{O}\Big(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n} \Big) + \mathcal{O}\Big(\frac{n \log \log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n} \Big),$$

which proves the lemma. □

We now consider the tree encoder $E_\psi : \mathcal{T} \to \{0,1\}^*$ defined by $E_\psi(t) = B(\mathcal{G}_t)$, where as before the grammar-based tree compressor $t \mapsto \mathcal{G}_t$ has to satisfy (16) for every tree $t \in \mathcal{T}_n$ and n large enough. The following theorem says that the encoder E_ψ is universal with respect to the class of all k^{th}-order tree processes. Universality means that the maximal pointwise redundancy [34] converges to zero for $n \to \infty$.

Theorem 3. *For every $t \in \mathcal{T}_n$, every $k \geq 0$ and every k^{th}-order tree process $\mathcal{P} = (P_z)_{z\in\{0,1\}^k}$ with $\mathsf{Prob}_{\mathcal{P}}(t) > 0$ we have*

$$|E_\psi(t)| \leq -\log_2 \mathsf{Prob}_{\mathcal{P}}(t) + \mathcal{O}\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + \mathcal{O}\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right) + \sigma.$$

Proof. Let $\mathcal{P} = (P_z)_{z\in\{0,1\}^k}$ be a k^{th}-order tree process with $\mathsf{Prob}_{\mathcal{P}}(t) > 0$. Lemmas 6 and 7 yield

$$\begin{aligned} |E_\psi(t)| &\leq \mathcal{O}(|\mathcal{G}_t|) + H(\mathcal{G}_t) + \sigma \\ &\leq \mathcal{O}(|\mathcal{G}_t|) - \log_2 \mathsf{Prob}_{\mathcal{P}}(t) + \mathcal{O}\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + \mathcal{O}\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right) + \sigma \\ &= -\log_2 \mathsf{Prob}_{\mathcal{P}}(t) + \mathcal{O}\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + \mathcal{O}\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right) + \sigma, \end{aligned}$$

where the last equality uses the bound $|\mathcal{G}_t| \in \mathcal{O}(n/\log_{\hat{\sigma}} n)$. □

By taking in Theorem 3 for $\mathcal{P}$ the empirical k^{th}-order tree process $\mathcal{P}^t$ and using Theorem 1, we obtain:

Theorem 4. *For every $t \in \mathcal{T}_n$ and every $k \geq 0$ we have*

$$|E_\psi(t)| \leq H_k(t) + \mathcal{O}\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + \mathcal{O}\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right) + \sigma.$$

5. Extension to unranked trees

So far, we have only considered node-labelled binary trees. In this section, we consider unranked, ordered node-labelled trees, where the number of children of a node (also-called its degree) can be any natural number and the children of every node are totally ordered. As before, each node is labelled by an element of some finite alphabet Σ. Let us denote by $\mathcal{U}(\Sigma)$ (or simply $\mathcal{U}$) the set of all such trees. For technical reasons we also define *forests* which are ordered sequences of trees from $\mathcal{U}$. The set of forests is denoted with $\mathcal{F}$. The sets $\mathcal{U}$ and $\mathcal{F}$ can be inductively defined as the smallest sets of strings over the alphabet $\Sigma \cup \{(,)\}$ such that the following conditions hold:

- $\varepsilon \in \mathcal{F}$ (this is the empty forest),
- if $a \in \Sigma$ and $f \in \mathcal{F}$ then $a(f) \in \mathcal{U}$ (the tree whose node is labelled with a and whose children form the forest f)
- if $t \in \mathcal{U}$ and $f \in \mathcal{F}$ then $tf \in \mathcal{F}$ (the forest consisting of the tree t followed by the forest f).

The singleton tree $a()$ (which is obtained by taking $f = \varepsilon$ in the second point) is usually written as a. Note that $\mathcal{U} \subseteq \mathcal{F}$ and that $\mathcal{F} = \mathcal{U}^*$. The size $|f|$ of $f \in \mathcal{F}$ is the number of occurrences of Σ-labels in f; formally: $|\varepsilon| = 0$, $|a(f)| = 1 + |f|$ and $|tf| = |t| + |f|$ for $a \in \Sigma$, $t \in \mathcal{U}$, and $f \in \mathcal{F}$.

The *first-child/next-sibling encoding* transforms a forest $f \in \mathcal{F}$ into a binary tree $\mathrm{fcns}(f) \in \mathcal{T}$. It is defined inductively as follows (recall that $\square \in \Sigma$ is a fixed distinguished symbol in Σ):

- $\mathrm{fcns}(\varepsilon) = \square$ and
- $\mathrm{fcns}(a(f)g) = a(\mathrm{fcns}(f), \mathrm{fcns}(g))$ for $f, g \in \mathcal{F}$ and $a \in \Sigma$.

Thus, the left (resp., right) child of a node in $\mathrm{fcns}(f)$ is the first child (resp., right sibling) of the node in f or a $\square$-labelled leaf if it does not exist.

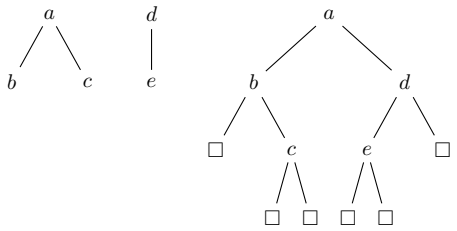

FIGURE 5. Forest f on the left and $\mathrm{fcns}(f)$ on the right from Example 8.

Example 8. *If $f = a(bc)d(e)$ then*

$$\begin{aligned} \mathit{fcns}(f) &= \mathit{fcns}(a(bc)d(e)) = a(\mathit{fcns}(bc), \mathit{fcns}(d(e)) \\ &= a(b(\Box, \mathit{fcns}(c)), d(\mathit{fcns}(e), \Box)) = a(b(\Box, c(\Box, \Box)), d(e(\Box, \Box), \Box)), \end{aligned}$$

see also Figure 5.

Note that if $t \in \mathcal{U}$, $|t| = n$ then $\mathrm{fcns}(t)$ is a binary tree with n internal nodes. Hence we have $|\mathrm{fcns}(t)| = n+1$ (which is the number of leaves of $\mathrm{fcns}(t)$). We define the k^{th}-order empirical entropy of an unranked tree $t \in \mathcal{U}$ as $H_k(t) = H_k(\mathrm{fcns}(t))$. Note that this definition is independent of the choice of the symbol $\Box \in \Sigma$. From Theorem 4, we immediately obtain:

Theorem 5. *For every $t \in \mathcal{U}$ with $|t| = n$ and every $k \geq 0$ we have*

$$|E_\psi(\mathit{fcns}(t))| \leq H_k(t) + \mathcal{O}\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + \mathcal{O}\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right) + \sigma.$$

The above definition of the k^{th}-order empirical entropy of an unranked tree can be also applied to binary trees t (a binary tree can be viewed as a particular unranked tree). This yields $H_k(\mathrm{fcns}(t))$ and leads to the question how this value relates to $H_k(t)$ (the k^{th}-order empirical entropy of t as defined before in (14)). In one direction, we have the following bound:

Lemma 8. *Let $t \in \mathcal{T}(\Sigma)$ denote a binary tree with first-child next-sibling encoding $\mathit{fcns}(t) \in \mathcal{T}(\Sigma)$. Then $H_{2k}(\mathit{fcns}(t)) \leq H_{k-1}(t)$ for $1 \leq k \leq |t|$.*

The somewhat technical proof of Lemma 8 can be found in Appendix B. In contrast to Lemma 8, there are families of binary trees t_n where $H_k(\mathrm{fcns}(t_n))$ is exponentially smaller than $H_{k'}(t_n)$ for every $n \geq 1$ and $k, k' \geq 2$ with $k, k' \in o(n)$. Define t_n inductively by $t_1 = a$ and $t_n = a(c, t_{n-1})$ if n is even and $t_n = a(b, t_{n-1})$ if n is odd. Thus, t_n denotes a right-degenerate binary tree of size n, whose inner nodes and right-most leaf are labelled with a and whose leaves except for the right-most leaf are alternately labelled b and c. We get $H_k(t_n) \in \Theta(n-k)$: there are $n-k$ many nodes v with k-history $(a1)^{k-1}a0$, and about half of them are b-labelled leaves, while the other half are c-labelled leaves. Moreover, we have $H_k(\mathrm{fcns}(t_n)) \in \Theta(\log(n-k))$: the fcns-encodings of the binary trees t_n can be inductively defined by $\mathrm{fcns}(t_1) = a(\Box, \Box)$ and $\mathrm{fcns}(t_n) = a(c(\Box, \mathrm{fcns}(t_{n-1})), \Box)$ if n is even and $\mathrm{fcns}(t_n) = a(b(\Box, \mathrm{fcns}(t_{n-1})), \Box)$ if n is odd. Intuitively, as the labels b and c are thus incorporated in k-histories of nodes of $\mathrm{fcns}(t_n)$, we can thus determine the label of a node from its k-history for $k \geq 2$ for most nodes of $\mathrm{fcns}(t_n)$.

Our definition of the k^{th}-order empirical entropy of an unranked tree via the fcns-encoding has a practical motivation. Unranked trees occur for instance in the context of XML, where the hierarchical structure of a document is represented as an unranked node labelled tree. In this setting, the label of a node quite often depends on (i) the labels of the ancestor nodes and (ii) the labels of the (left) siblings. This dependence is captured by our definition of the k^{th}-order empirical entropy.

We also confirmed this intuition by experimental data (shown in Table 1) with real XML document trees (ignoring textual data at the leaves) showing that in these cases the k^{th}-order empirical entropy is indeed very small compared to the worst-case bit size. More precisely, we computed for 21 real XML document trees[5] the k^{th}-order empirical entropy (for $k = 1, 2, 4, 8$) and divided the value by the worst-case bit length $2n + \log_2(\sigma)n$, where n is the number of nodes and σ is the number of node labels [18].

Our experimental results combined with our entropy bound (2) for grammar-based compression are in accordance with the fact that grammar-based tree compressors yield impressive compression ratios for XML document trees, see e.g. [29]. Some of the XML documents from our experiments were also used in [29], where the performance of the grammar-based tree compressor TreeRePair was tested. An interesting observation is that those XML trees, for which our k-th order empirical entropy is large are indeed those XML trees with the worst compression ratio for TreeRePair in [29]. This is in particular true for the Treebank document, see Table 1. TreeRePair obtained for Treebank a compression ratio of around 20%, whereas for all other documents tested in [29] TreeRePair achieved a compression ratio below 8%.

XML document	n	σ	$w := (2 + \log_2 \sigma)n$	H_1/w	H_2/w	H_4/w	H_8/w
Baseball	28 306	46	212 961.9447	2.9818 %	1.2547 %	0.6739 %	0.6662 %
DBLP	3 332 130	35	23 755 697.8193	10.9775 %	8.7407 %	8.2134 %	6.7270 %
DCSD-Normal	2 242 699	50	17 142 868.6330	4.2437 %	2.2481 %	1.7517 %	1.3038 %
EnWikiNew	404 652	20	2 558 180.8475	9.5317 %	3.0760 %	3.0759 %	2.9378 %
EnWikiQuote	262 955	20	1 662 382.6021	9.4270 %	3.1014 %	3.1014 %	3.1006 %
EnWikiVersity	495 839	20	3 134 658.5046	8.8952 %	2.3753 %	2.3753 %	2.3750 %
EXI-Array	226 523	47	1 711 288.1304	0.2506 %	0.2495 %	0.2492 %	0.2483 %
EXI-factbook	55 453	199	534 379.7451	2.2034 %	0.9450 %	0.8132 %	0.8092 %
EXI-Invoice	15 075	52	116 084.1288	0.0484 %	0.0268 %	0.0139 %	0.0098 %
EXI-Telecomp	177 634	39	1 294 135.1377	1.5405 %	0.0044 %	0.0034 %	0.0021 %
EXI-weblog	93 435	12	521 830.9713	0.0032 %	0.0028 %	0.0028 %	0.0028 %
Lineitem	1 022 976	18	6 311 685.1983	0.0003 %	0.0003 %	0.0003 %	0.0003 %
Mondial	22 423	23	146 277.8297	11.1285 %	9.2940 %	8.4702 %	7.7679 %
NASA	476 646	61	3 780 154.2290	7.7424 %	4.4588 %	3.8898 %	3.8054 %
Shakespeare	179 690	22	1 160 695.2676	11.9140 %	10.8416 %	10.6368 %	10.4765 %
SwissProt	2 977 031	85	25 035 017.5080	12.1892 %	10.5249 %	9.2455 %	8.1204 %
TCSD-Normal	2 749 751	24	18 107 007.2213	8.5450 %	8.4004 %	8.2862 %	8.2472 %
Treebank	2 437 666	250	24 293 253.5140	30.8912 %	23.0825 %	19.2444 %	13.4058 %
USHouse	6 712	43	49 845.0890	21.0500 %	18.2164 %	12.6572 %	9.3754 %
XMark1	167 865	74	1 378 079.8892	12.1610 %	9.5101 %	9.2271 %	8.4281 %
XMark2	1 666 315	74	13 679 535.2849	12.2125 %	9.5634 %	9.3259 %	8.9400 %

TABLE 1. Experimental results for XML tree structures, where n denotes the number of nodes and σ denotes the number of node labels.

[5]All data are available from `http://xmlcompbench.sourceforge.net/Dataset.html`.

6. String straight-line programs versus higher-order empirical entropy of strings

Our definition of k^{th}-order empirical entropy does not capture all regularities that can be exploited in grammar-based compression. Take for instance a complete unlabelled binary tree t_n of height n (all paths from the root to a leaf have length n and the alphabet Σ of node labels contains a single symbol). This tree has 2^n leaves and is very well compressible: its minimal DAG has only $n+1$ nodes, hence there also exists a TSLP of size $n+1$ for t_n. But for every fixed k the k^{th}-order empirical entropy of t_n divided by n converges to 2 (the trivial upper bound) for $n \to \infty$. If $n \gg k$ then for every k-history z the number of leaves with k-history z is roughly the same as the number of internal nodes with k-history z. Hence, although t_n is highly compressible with TSLPs (and even DAGs), its k^{th}-order empirical entropy is close to the maximal value. We show in the following that the same phenomenon occurs for grammar-based string compression and the well-established empirical entropy of strings.

The k^{th}-order empirical entropy of a string is defined as follows (see e.g. [11]). Let Σ denote a finite alphabet and let $w \in \Sigma^*$. For a non-empty string $\alpha \in \Sigma^+$, define $w(\alpha) \in \Sigma^*$ as the string whose i^{th} symbol is the symbol in w immediately following the i^{th} occurrence of the string α in w. Thus, if α is not a suffix of w, the length of $w(\alpha)$ is equal to the number of occurrences of the string α in w. In case α is a suffix of w, $|w(\alpha)|$ is the number of occurrences of α in w minus one. Recall the definition of the unnormalized empirical entropy $H(w)$ of a string $w \in \Sigma^+$ (or tuple) from Section 2.1. For an integer $k \geq 1$, the *k^{th}-order (unnormalized) empirical entropy* of a string $w \in \Sigma^+$ is defined as

$$H_k(w) = \sum_{\alpha \in \Sigma^k} H(w(\alpha)),$$

where we set $H(\varepsilon) = 0$. For $k = 0$, $H_0(w) = H(w)$ is the (unnormalized) empirical entropy of w.

A *straight-line program* (SLP) for a string w is a context-free grammar that produces only the string w. The size of an SLP is the sum of the lengths of the right-hand sides of the production rules of the context-free grammar, see e.g. [27] for details. We prove that for each $n \geq 1$ there exists a string of length $2^{n+1} - 1$, which is highly compressible with SLPs, but whose k^{th}-order empirical entropy is close to the maximum.

Theorem 6. *There exists a family of strings $(S_n)_n$ ($n \geq 1$) over a binary alphabet with the following properties:*

- $|S_n| = 2^{n+1} - 1$,
- *there exists an SLP of size $3n$ for S_n, and*
- $H_k(S_n) \geq 2^{n+1-k}(1 - o(1))$ *for* $k \in o(n)$.

Proof. We inductively define a string $S_n \in \{a, b\}^*$ for $n \geq 1$ as follows: We set

- $S_1 = baa$ and
- $S_n = bS_{n-1}S_{n-1}$.

We have $|S_n| = 2^{n+1} - 1$. The string S_n corresponds to the preorder traversal of the perfect binary tree of size 2^n, whose internal nodes are labelled with the symbol b and whose leaves are labelled with the symbol a. The recursive definition of S_n directly translates to an SLP for S_n of size $3n$ (there is a nonterminal for each S_i

with $1 \leq i \leq n$ and each rule has three symbols on the right-hand side according to the recursive definition).

It remains to show that $H_k(S_n) \geq 2^{n-k}$ for $0 \leq k < n$. We start with the case $k = 0$. Recall that $|w|_x$ denotes the number of occurrences of a symbol x in a string w, as defined in Section 2. We have $|S_n|_a = 2^n$ and $|S_n|_b = 2^n - 1$, which yields

$$\begin{aligned} H(S_n) &= \sum_{x \in \{a,b\}} |S_n|_x \log_2 \left(\frac{|S_n|}{|S_n|_x} \right) \\ &= 2^n \log_2 \left(\frac{2^{n+1}-1}{2^n} \right) + (2^n - 1) \log_2 \left(\frac{2^{n+1}-1}{2^n - 1} \right). \end{aligned}$$

Define the function $g : [2, \infty) \to \mathbb{R}$ by

$$g(x) = \frac{x}{2x-1} \log_2 \left(\frac{2x-1}{x} \right) + \frac{x-1}{2x-1} \log_2 \left(\frac{2x-1}{x-1} \right).$$

It converges to 1 from below for $x \to \infty$. Since $|S_n| = 2^{n+1} - 1$ we have $H(S_n) = g(2^n)|S_n| \geq 2^{n+1}(1 - o(1))$.

Let us now consider the case $k \geq 1$ and let $1 \leq m \leq n$. By construction of S_n, the last symbol of S_n is a. Therefore, the length of the string $S_n(b^m)$ equals the number of occurrences of the string b^m in S_n. In order to lower-bound the k^{th}-order empirical entropy of S_n, we first show inductively in n, that

$$|S_n(b^m)| = 2^{n-m+1} - 1 \tag{25}$$

for $1 \leq m \leq n$: For the base case, let $n = 1$. We have $S_1 = baa$ and thus, $|S_1(b)| = 1$. For the induction step, let $n > 1$. By definition of S_n, we have $S_n = bS_{n-1}S_{n-1}$. By the induction hypothesis, we have $|S_{n-1}(b^m)| = 2^{n-m} - 1$ for $1 \leq m \leq n-1$. Moreover, b^n does not occur in S_{n-1} (which follows by induction), i.e., $|S_{n-1}(b^n)| = 0 = 2^{n-n} - 1$. By construction, the last symbol of the string S_{n-1} is a. Thus, for all $1 \leq m \leq n$ we have $|S_{n-1}S_{n-1}(b^m)| = 2|S_{n-1}(b^m)| = 2^{n-m+1} - 2$. Hence, as the string b^m with $1 \leq m \leq n$ occurs additionally as a prefix of the string $S_n = bS_{n-1}S_{n-1}$, the number of occurrences of b^m in S_n in total is $|S_n(b^m)| = 2^{n-m+1} - 1$ for every $1 \leq m \leq n$. This proves (25).

Next, we count the number of occurrences of b^m in S_n, which are followed by the symbol a, that is, we count $|S_n(b^m)|_a$. We show inductively in n, that

$$|S_n(b^m)|_a = 2^{n-m}$$

for $1 \leq m \leq n$: For the base case, let $n = 1$. As $S_1 = baa$, we have $|S_1(b)|_a = 1$. For the induction step, let $n > 1$. By the induction hypothesis, we have $|S_{n-1}(b^m)|_a = 2^{n-1-m}$ for $1 \leq m \leq n-1$. As S_{n-1} ends with a, we obtain $|S_{n-1}S_{n-1}(b^m)|_a = 2^{n-m}$ for $1 \leq m \leq n-1$. Moreover, the construction of S_n implies that the prefix b^n of S_n, which is the only occurrence of b^n in S_n, is followed by the symbol a. Thus, $|S_n(b^m)|_a = 2^{n-m}$ for $1 \leq m \leq n$, which proves the claim.

As $|S_n(b^m)|_a = 2^{n-m}$, we have $|S_n(b^m)|_b = 2^{n-m} - 1$. Thus, we obtain the following lower bound for the k^{th}-order empirical entropy of S_n for $k \in o(n)$.

$$\begin{aligned} H_k(S_n) &= \sum_{\alpha \in \{a,b\}^k} H(S_n(\alpha)) \\ &\geq \sum_{x \in \{a,b\}} |S_n(b^k)|_x \log_2 \left(\frac{|S_n(b^k)|}{|S_n(b^k)|_x} \right) \end{aligned}$$

$$\begin{aligned} &= 2^{n-k}\log_2\left(\frac{2^{n-k+1}-1}{2^{n-k}}\right) + (2^{n-k}-1)\log_2\left(\frac{2^{n-k+1}-1}{2^{n-k}-1}\right) \\ &= (2^{n-k+1}-1)g(2^{n-k}) \\ &\geq 2^{n-k+1}(1-o(1)) \end{aligned}$$

This proves the theorem. □

References

[1] Janos Aczél. On Shannon's inequality, optimal coding, and characterizations of Shannon's and Renyi's entropies. Technical Report Research Report AA-73-05, University of Waterloo, 1973. `https://cs.uwaterloo.ca/research/tr/1973/CS-73-05.pdf`.

[2] Dror Baron and Yoram Bresler. An O(N) semipredictive universal encoder via the BWT. *IEEE Transactions on Information Theory*, 50(5): 928–937, 2004.

[3] Philip Bille, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Tree compression with top trees. *Information and Computation*, 243:166–177, 2015.

[4] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.

[5] Thomas M. Cover. Enumerative source encoding. *IEEE Transactions on Information Theory*, 19(1):73–77, 1973.

[6] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (2. ed.)*. Wiley, 2006.

[7] Michelle Effros, Karthik Visweswariah, Sanjeev R. Kulkarni, and Sergio Verdú. Universal lossless source coding with the Burrows Wheeler Transform. *IEEE Transactions on Information Theory*, 48(5):1061–1081, 2002.

[8] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S.Muthikrishnan. Structuring labelled trees for optimal succinctness, and beyond. *Proceedings of the* 46*th Annual Symposium on Foundations of Computer Science, FOCS 2005*, pages 184-196. IEEE Computer Society Press, 2005.

[9] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labelled trees, with applications. *Journal of the ACM*, 57(1):4:1–4:33, 2009.

[10] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[11] Travis Gagie. Large alphabets and incompressibility. *Information Processing Letters*, 99(6):246–251, 2006.

[12] Moses Ganardi, Danny Hucke, Artur Jeż, Markus Lohrey, and Eric Noeth. Constructing small tree grammars and small circuits for formulas. *Journal of Computer and System Sciences*, 86:136–158, 2017.

[13] Moses Ganardi and Markus Lohrey. A universal tree balancing theorem. *ACM Transaction on Computation Theory*, 11(1):1:1–1:25, 2018.

[14] Moses Ganardi, Danny Hucke, Markus Lohrey, and Louisa Seelbach Benkner. Universal tree source coding using grammar-based compression. *IEEE Transactions on Information Theory*, 65(10):6399–6413, 2019.

[15] Michal Ganczorz. Entropy bounds for grammar compression. *CoRR*, abs/1804.08547, 2018.

[16] Michal Ganczorz. Using statistical encoding to achieve tree succinctness never seen before. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020*, volume 154 of *LIPIcs*, pages 22:1–22:29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[17] 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.

[18] Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2(4):510–534, 2006.

[19] Danny Hucke, Markus Lohrey, and Louisa Seelbach Benkner. Entropy bounds for grammar-based tree compressors. In *Proceedings of the IEEE International Symposium on Information Theory, ISIT 2019*, pages 1687–1691. IEEE Computer Society Press, 2019.

[20] Danny Hucke, Markus Lohrey, and Louisa Seelbach Benkner. A comparison of empirical tree entropies. In *Proceedings of the 27th International Symposium on String Processing and*

Information Retrieval, SPIRE 2020, volume 12303 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2020.

[21] Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees with applications. *Journal of Computer and System Sciences*, 78(2):619–631, 2012.

[22] John C. Kieffer and En hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000.

[23] John C. Kieffer, En-Hui Yang, Gregory J. Nelson, and Pamela C. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Transactions on Information Theory*, 46(4):1227–1245, 2000.

[24] Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theoretical Computer Science*, 483:115–133, 2013.

[25] Raphail E. Krichevsky and Victor K. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory*, 27(2):199-206, 1981.

[26] N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Proceedings of the 1999 Data Compression Conference, DCC 1999*, pages 296–305. IEEE Computer Society Press, 1999.

[27] Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

[28] Markus Lohrey. Grammar-based tree compression. In *Proceedings of the 19th International Conference on Developments in Language Theory, DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 46–57. Springer, 2015.

[29] Markus Lohrey, Sebastian Maneth, and Roy Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013.

[30] Markus Lohrey, Sebastian Maneth, and Carl Philipp Reh. Compression of Unordered XML Trees. In *Proceedings of the 20th International Conference on Database Theory, ICDT 2017*, volume 68 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[31] Gonzalo Navarro and Luís M. S. Russo. Re-pair achieves high-order entropy. In *Proceedings of the 2008 Data Compression Conference, DCC 2008*, page 537. IEEE Computer Society, 2008.

[32] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

[33] Carlos Ochoa and Gonzalo Navarro. RePair and all irreducible grammars are upper bounded by high-order empirical entropy. *IEEE Transactions on Information Theory*, 65(5):3160–3164, 2019.

[34] Eli Plotnik, Marcelo J. Weinberger, and Jacob Ziv. Upper bounds on the probability of sequences emitted by finite-state sources and on the redundancy of the Lempel-Ziv algorithm. *IEEE Transactions on Information Theory*, 38(1):66–72, 1992.

[35] Yuri M. Shtarkov. Universal sequential coding of single messages. *Problems of Information Transmission,* 23(3):175-186, 1987.

[36] Slavko Simic. On a global upper bound for Jensen's inequality. *Journal of Mathematical Analysis and Applications* 343: 414–419, 2008.

[37] Frans M. J. Willems and Yuri M. Shtarkov and Tjalling J. Tjalkens. The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41(3): 653–664, 1995.

[38] Jie Zhang, En-Hui Yang, and John C. Kieffer. A universal grammar-based code for lossless compression of binary trees. *IEEE Transactions on Information Theory*, 60(3):1373–1386, 2014.

[39] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

Appendix A. Histories of length smaller than k

In order to define k^{th}-order empirical entropy for binary trees, there are basically three possibilities how to deal with nodes whose history is shorter than $2k$:

(i) pad the histories with a fixed dummy symbol $\square \in \Sigma$ and direction $i \in \{0, 1\}$,

(ii) allow histories of length smaller than $2k$, or, equivalently, pad the histories with a fixed dummy symbol $\diamond \notin \Sigma$ and direction $i \in \{0,1\}$, or
(iii) ignore nodes whose history is of length smaller than $2k$.

Recall that in the main text we used the variant (i) with $i = 0$. In this subsection, we show that the above three variants are basically equivalent if k is small compared to the size of the binary tree.

Fix an integer $k \geq 1$. Recall that in Section 2.2.4 we defined for a tree t, a k-history $z \in \mathcal{L}_k$, and $\tilde{a} \in \Sigma \times \{0,2\}$ the numbers $m_z^t = |V_z(t)|$ and $m_{z,\tilde{a}}^t = |\{v \in V_z(t) \mid \lambda(v) = \tilde{a}\}|$. The tree t will be fixed in this section; hence we will write m_z and $m_{z,\tilde{a}}$ in the following. We define several variants of these numbers.

For a k-history $z \in \mathcal{L}_k$ and $\tilde{a} \in \Sigma \times \{0,2\}$ we define:

$$\begin{aligned} m^{<} &= |\{v \in V(t) \mid |v| < k\}|, \\ m_z^{<} &= |\{v \in V_z(t) \mid |v| < k\}|, \\ m_{z,\tilde{a}}^{<} &= |\{v \in V_z(t) \mid |v| < k, \lambda(v) = \tilde{a}\}|, \\ m_z^{\geq} &= |\{v \in V_z(t) \mid |v| \geq k\}|, \\ m_{z,\tilde{a}}^{\geq} &= |\{v \in V_z(t) \mid |v| \geq k, \lambda(v) = \tilde{a}\}|. \end{aligned}$$

We have $m^{<} \leq 2^k - 1$ and $m^{<} \geq 2k - 1$ if $|t| \geq k$. Also note that $m_z = m_z^{<} + m_z^{\geq}$ and $\sum_{z \in \mathcal{L}_k} m_z^{<} = m^{<}$ and $\sum_{z \in \mathcal{L}_k} m_z^{\geq} = 2|t| - 1 - m^{<}$.

Fix a fresh symbol $\diamond \notin \Sigma$ and let $\mathcal{L}^{\diamond} = ((\Sigma \cup \{\diamond\})\{0,1\})^*$ and $\mathcal{L}_k^{\diamond} = \{w \in \mathcal{L}^{\diamond} \mid |w| = 2k\}$. Clearly, $\mathcal{L} \subseteq \mathcal{L}^{\diamond}$ and $\mathcal{L}_k \subseteq \mathcal{L}_k^{\diamond}$. Let $\ell_k : \mathcal{L}^{\diamond} \to \mathcal{L}_k^{\diamond}$ denote the partial function mapping a string $z \in \mathcal{L}^{\diamond}$ with $|z| \geq 2k$ to the suffix of z of length $2k$. For a binary tree t and a node $v \in V(t)$, define $h_k^{\diamond}(v) = \ell_k((\diamond 0)^k h(v))$. Note that $h_k^{\diamond}(v) = h_k(v)$ for nodes $v \in V(t)$ with $|v| \geq k$. Finally, for $z \in \mathcal{L}_k^{\diamond}$ and $\tilde{a} \in \Sigma \times \{0,2\}$ we define

$$\begin{aligned} m_z^{\diamond} &= |\{v \in V(t) \mid h_k^{\diamond}(v) = z, |v| < k\}|, \\ m_{z,\tilde{a}}^{\diamond} &= |\{v \in V(t) \mid h_k^{\diamond}(v) = z, |v| < k, \lambda(v) = \tilde{a}\}|. \end{aligned}$$

Using the above numbers, we can define three natural variations of the k^{th}-order empirical entropy of a binary node-labelled tree t:

(i) Padding histories of length shorter than $2k$ with $\square \in \Sigma$ and $i \in \{0,1\}$ yields the definition of k^{th}-order empirical entropy from Section 2 (for $i = 0$):

$$H_k(t) = \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}} \log_2 \left(\frac{m_z}{m_{z,\tilde{a}}} \right).$$

(ii) Padding histories of length shorter than $2k$ with $\diamond \notin \Sigma$ and $i = 0$ yields

$$H_k^{\diamond}(t) = \sum_{z \in \mathcal{L}_k^{\diamond}} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{\geq} \log_2 \left(\frac{m_z^{\geq}}{m_{z,\tilde{a}}^{\geq}} \right) + m_{z,\tilde{a}}^{\diamond} \log_2 \left(\frac{m_z^{\diamond}}{m_{z,\tilde{a}}^{\diamond}} \right).$$

This is equivalent to allowing histories of length shorter than $2k$: By padding with a symbol $\diamond \notin \Sigma$, we have $h_k^{\diamond}(v_1) = h_k^{\diamond}(v_2)$ if and only if $h(v_1) = h(v_2)$ for nodes $v_1, v_2 \in V(t)$ with $|v_1|, |v_2| < k$.

(iii) Ignoring nodes whose history is of length smaller than $2k$ yields

$$H_k^{\geq}(t) = \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{\geq} \log_2 \left(\frac{m_z^{\geq}}{m_{z,\tilde{a}}^{\geq}} \right).$$

We can now show that these three approaches are basically equivalent:

Theorem 7. *For every $k \geq 1$ and every binary tree t, we have the following:*

$$\begin{aligned}
|H_k(t) - H_k^{\diamond}(t)| &\leq m^{<}\left(1 + \frac{1}{\ln(2)} + \log_2 \sigma + \log_2\left(\frac{2|t|-1}{m^{<}}\right)\right),\\
|H_k(t) - H_k^{\geq}(t)| &\leq m^{<}\left(1 + \frac{1}{\ln(2)} + \log_2 \sigma + \log_2\left(\frac{2|t|-1}{m^{<}}\right)\right),\\
|H_k^{\geq}(t) - H_k^{\diamond}(t)| &\leq m^{<}(1 + \log_2 \sigma).
\end{aligned}$$

Proof. First, note that

$$0 \leq \sum_{z \in \mathcal{L}_k} m_z^{<} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} \frac{m_{z,\tilde{a}}^{<}}{m_z^{<}} \log_2\left(\frac{m_z^{<}}{m_{z,\tilde{a}}^{<}}\right) \leq m^{<}(1 + \log_2 \sigma), \tag{26}$$

as the inner sum is the Shannon entropy $H(P)$ of the probability distribution $P : \Sigma \times \{0,2\} \to [0,1]$ given by $P(\tilde{a}) = m_{z,\tilde{a}}^{<}/m_z^{<}$ (and hence $H(P) \leq \log_2(2\sigma) = 1 + \log_2 \sigma$) and as $\sum_{z \in \mathcal{L}_k} m_z^{<} = m^{<}$. Analogously, we get

$$0 \leq \sum_{z \in \mathcal{L}_k^{\diamond}} m_z^{\diamond} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} \frac{m_{z,\tilde{a}}^{\diamond}}{m_z^{\diamond}} \log_2\left(\frac{m_z^{\diamond}}{m_{z,\tilde{a}}^{\diamond}}\right) \leq m^{<}(1 + \log_2 \sigma). \tag{27}$$

We start with upper-bounding $|H_k(t) - H_k^{\geq}(t)|$: By the log-sum inequality (Lemma 1) and (26), we get

$$\begin{aligned}
H_k(t) &= \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}} \log_2\left(\frac{m_z}{m_{z,\tilde{a}}}\right)\\
&= \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} (m_{z,\tilde{a}}^{\geq} + m_{z,\tilde{a}}^{<}) \log_2\left(\frac{m_z^{\geq} + m_z^{<}}{m_{z,\tilde{a}}^{\geq} + m_{z,\tilde{a}}^{<}}\right)\\
&\geq \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{\geq} \log_2\left(\frac{m_z^{\geq}}{m_{z,\tilde{a}}^{\geq}}\right) + \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{<} \log_2\left(\frac{m_z^{<}}{m_{z,\tilde{a}}^{<}}\right)\\
&\geq \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{\geq} \log_2\left(\frac{m_z^{\geq}}{m_{z,\tilde{a}}^{\geq}}\right)\\
&= H_k^{\geq}(t).
\end{aligned}$$

Moreover, we find

$$\begin{aligned}
H_k(t) &= \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} (m_{z,\tilde{a}}^{\geq} + m_{z,\tilde{a}}^{<}) \log_2\left(\frac{m_z^{\geq} + m_z^{<}}{m_{z,\tilde{a}}^{\geq} + m_{z,\tilde{a}}^{<}}\right)\\
&= \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{\geq} \log_2\left(\frac{m_z^{\geq} + m_z^{<}}{m_z^{\geq}} \cdot \frac{m_z^{\geq}}{m_{z,\tilde{a}}^{\geq} + m_{z,\tilde{a}}^{<}}\right) +\\
&\quad \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{<} \log_2\left(\frac{m_z^{\geq} + m_z^{<}}{m_z^{<}} \cdot \frac{m_z^{<}}{m_{z,\tilde{a}}^{\geq} + m_{z,\tilde{a}}^{<}}\right)\\
&\leq \sum_{z \in \mathcal{L}_k} m_z^{\geq} \log_2\left(\frac{m_z^{\geq} + m_z^{<}}{m_z^{\geq}}\right) + \sum_{z \in \mathcal{L}_k} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{\geq} \log_2\left(\frac{m_z^{\geq}}{m_{z,\tilde{a}}^{\geq}}\right) +
\end{aligned}$$

$$\sum_{z\in\mathcal{L}_k} m_z^< \log_2\left(\frac{m_z^\geq + m_z^<}{m_z^<}\right) + \sum_{z\in\mathcal{L}_k}\sum_{\tilde{a}\in\Sigma\times\{0,2\}} m_{z,\tilde{a}}^< \log_2\left(\frac{m_z^<}{m_{z,\tilde{a}}^<}\right)$$
$$\leq H_k^\geq(t) + m^< (1+\log_2\sigma) +$$
$$(2|t|-1-m^<)\log_2\left(\frac{2|t|-1}{2|t|-1-m^<}\right) + m^<\log_2\left(\frac{2|t|-1}{m^<}\right)$$

by the log-sum inequality (Lemma 1) and our estimate from (26). We have

$$(2|t|-1-m^<)\log_2\left(\frac{2|t|-1}{2|t|-1-m^<}\right) \leq \frac{m^<}{\ln(2)}, \tag{28}$$

which follows immediately from the mean-value theorem: as a consequence of the mean-value theorem, for every mapping $f : [a,b] \to \mathbb{R}$, which is differentiable on $[a,b]$, we have

$$|f(b)-f(a)| \leq \max_{x\in[a,b]} |f'(x)| \cdot |b-a|.$$

With $f(x) = \log_2(x)$, $a = 2|t|-1-m^<$ and $b = 2|t|-1$ and by logarithmic identities, we obtain the estimate (28). Thus, we have:

$$|H_k(t) - H_k^\geq(t)| \leq m^< \left(1+\log_2\sigma + \frac{1}{\ln(2)} + \log_2\left(\frac{2|t|-1}{m^<}\right)\right).$$

Next, we upper-bound $|H_k^\geq(t) - H_k^\diamond(t)|$: From the definitions of $H_k^\geq(t)$ and $H_k^\diamond(t)$, we get

$$H_k^\diamond(t) = H_k^\geq(t) + \sum_{z\in\mathcal{L}_k^\diamond}\sum_{\tilde{a}\in\Sigma\times\{0,2\}} m_{z,\tilde{a}}^\diamond \log_2\left(\frac{m_z^\diamond}{m_{z,\tilde{a}}^\diamond}\right).$$

As the second sum on the right-hand side is between 0 and $m^<(1+\log_2\sigma)$ (see (27)), we get $|H_k^\geq(t) - H_k^\diamond(t)| \leq m^<(1+\log_2\sigma)$.

Finally, as $H_k^\diamond(t) \geq H_k^\geq(t)$ and $H_k(t) \geq H_k^\geq(t)$, we have

$$|H_k(t) - H_k^\diamond(t)| \leq m^< \left(1+\log_2\sigma + \frac{1}{\ln(2)} + \log_2\left(\frac{2|t|-1}{m^<}\right)\right).$$

This proves the theorem. □

Theorem 7 moreover shows that the choice of the symbol $\square \in \Sigma$ used for padding the histories only affects the value of the k^{th}-order empirical entropy by an additive term of at most $m^<(1+\log_2\sigma + 1/\ln(2)) + m^<\log_2((2|t|-1)/m^<)$.

Appendix B. Proof of Lemma 8

Fix a binary tree $t \in \mathcal{T}(\Sigma)$. By definition of the first-child next-sibling encoding, every inner node of $\mathrm{fcns}(t)$ corresponds in a bijective manner to a node of t: For an inner node v of $\mathrm{fcns}(t)$, let $\mathrm{fcns}^{-1}(v)$ denote the corresponding node of t and let $\mathrm{fcns}(v)$ denote the corresponding inner node of $\mathrm{fcns}(t)$ of a node v of t. If v is a node of t, then we obtain $h(\mathrm{fcns}(v))$ as follows: If $v = \varepsilon$, then $h(\mathrm{fcns}(v)) = \varepsilon$. Moreover, if v is a left child of a node $\mathrm{parent}(v)$ with label $a \in \Sigma$, then $h(\mathrm{fcns}(v)) = h(\mathrm{fcns}(\mathrm{parent}(v)))a0$ (and $h(v) = h(\mathrm{parent}(v))a0$). Finally, if v is a right child of a node $\mathrm{parent}(v)$ with label $a \in \Sigma$ and v's left sibling has label $a' \in \Sigma$, then $h(\mathrm{fcns}(v)) = h(\mathrm{fcns}(\mathrm{parent}(v)))a0a'1$ (and $h(v) = h(\mathrm{parent}(v))a1$). Thus, we are also able to determine $h(\mathrm{fcns}^{-1}(v))$ from $h(v)$ for every inner node v of $\mathrm{fcns}(t)$:

locating every occurrence of a pattern of the form $0a1$ with $a \in \Sigma$ in the string $h(v)$ and replacing it by 1 yields $h(\mathrm{fcns}^{-1}(v))$.

In particular, we have $|h(\mathrm{fcns}(v))| \leq 2|h(v)|$ for every node v of t, respectively, $|h(\mathrm{fcns}^{-1}(v))| \geq 1/2|h(v)|$ for every inner node v of $\mathrm{fcns}(t)$. Moreover, for every inner node v of $\mathrm{fcns}(t)$, we can uniquely determine $h_k(\mathrm{fcns}^{-1}(v))$ from $h_{2k}(v)$. Thus, we are also able to determine $h_{k-1}(\mathrm{fcns}^{-1}(v))$ from $h_{2k-1}(v)$ for every inner node v of $\mathrm{fcns}(t)$. Let

$$\mathcal{L}_m(\mathrm{fcns}(t)) = \{h_m(v) \mid v \text{ is an inner node of } \mathrm{fcns}(t)\}$$

denote the set of m-histories that appear as m-history of an inner node of $\mathrm{fcns}(t)$. We define a mapping $\varphi : \mathcal{L}_{2k}(\mathrm{fcns}(t)) \to \mathcal{L}_k$ by $\varphi(h_{2k}(v)) = h_k(\mathrm{fcns}^{-1}(v))$, which maps the $2k$-history of an inner node of $\mathrm{fcns}(t)$ to the k-history of the corresponding node in t: By the above considerations, this mapping is well-defined. Furthermore, we define a mapping $\pi : \mathcal{L}_{2k-1}(\mathrm{fcns}(t)) \to \mathcal{L}_{k-1}$ by $\pi(h_{2k-1}(v)) = h_{k-1}(\mathrm{fcns}^{-1}(v))$. Again, by the above considerations, this mapping is well-defined, as we are able to determine $h_{k-1}(\mathrm{fcns}^{-1}(v))$ from $h_{2k-1}(v)$.

For $m \geq 2$ we partition $\mathcal{L}_m$ into the following disjoint subsets:

$$\begin{aligned}
\mathcal{L}_m^0 &= \{a_1 i_1 \cdots a_m i_m \in \mathcal{L}_m \mid i_m = 0\},\\
\mathcal{L}_m^{01} &= \{a_1 i_1 \cdots a_m i_m \in \mathcal{L}_m \mid i_{m-1} = 0 \text{ and } i_m = 1\},\\
\mathcal{L}_m^{11} &= \{a_1 i_1 \cdots a_m i_m \in \mathcal{L}_m \mid i_{m-1} = 1 \text{ and } i_m = 1\}.
\end{aligned}$$

Moreover, we define $\mathcal{L}_{2k}^s(\mathrm{fcns}(t)) = \mathcal{L}_{2k}^s \cap \mathcal{L}_{2k}(\mathrm{fcns}(t))$ for $s \in \{0, 01, 11\}$. We observe the following:

(i) If $h_{2k}(v) \in \mathcal{L}_{2k}^{11}$ for a node v of $\mathrm{fcns}(t)$, then v is a $\square$-labelled leaf of $\mathrm{fcns}(t)$: As t is a binary tree, the right sibling of a node has no right sibling. Thus, there are no inner nodes v in $\mathrm{fcns}(t)$ with $h_{2k}(v) \in \mathcal{L}_{2k}^{11}$.

(ii) If $h_{2k}(v) \in \mathcal{L}_{2k}^{01}$ for a node v of $\mathrm{fcns}(t)$, then v is an inner node of $\mathrm{fcns}(t)$: This follows again from the fact that t is a binary tree (and hence does not have unary nodes).

(iii) If $h_{2k}(v) \in \mathcal{L}_{2k}^0$ for a node v of $\mathrm{fcns}(t)$, then v can be an inner node or a leaf of $\mathrm{fcns}(t)$. If v is a leaf, then its label is the fixed dummy symbol $\square \in \Sigma$.

(iv) For every $i \in \{0, 1\}$ and node v of t, we have $h_k(v) \in \mathcal{L}_k^i$ if and only if $h_{2k}(\mathrm{fcns}(v)) \in \mathcal{L}_{2k}^i(\mathrm{fcns}(t))$. In particular $\varphi(z) \in \mathcal{L}_k^0$ for every $z \in \mathcal{L}_{2k}^0(\mathrm{fcns}(t))$ and $\varphi(z) \in \mathcal{L}_k^1$ for every $z \in \mathcal{L}_{2k}^{01}(\mathrm{fcns}(t))$. Hence $\varphi(z) \neq \varphi(z')$ if $z \in \mathcal{L}_{2k}^{01}(\mathrm{fcns}(t))$ and $z' \in \mathcal{L}_{2k}^0(\mathrm{fcns}(t))$.

From (i), we obtain

$$\sum_{z \in \mathcal{L}_{2k}^{11}} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{\mathrm{fcns}(t)} \log_2 \left(\frac{m_z^{\mathrm{fcns}(t)}}{m_{z,\tilde{a}}^{\mathrm{fcns}(t)}} \right) = 0. \tag{29}$$

From (ii) and (iv), we obtain the following:

$$\begin{aligned}
&\sum_{z \in \mathcal{L}_{2k}^{01}} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m_{z,\tilde{a}}^{\mathrm{fcns}(t)} \log_2 \left(\frac{m_z^{\mathrm{fcns}(t)}}{m_{z,\tilde{a}}^{\mathrm{fcns}(t)}} \right)\\
= &\sum_{z \in \mathcal{L}_{2k}^{01}(\mathrm{fcns}(t))} \sum_{a \in \Sigma} m_{z,(a,2)}^{\mathrm{fcns}(t)} \log_2 \left(\frac{m_z^{\mathrm{fcns}(t)}}{m_{z,(a,2)}^{\mathrm{fcns}(t)}} \right)
\end{aligned}$$

$$\leq \sum_{y\in\mathcal{L}_k^1}\sum_{a\in\Sigma}\left(\sum_{z\in\varphi^{-1}(y)} m_{z,(a,2)}^{\mathrm{fcns}(t)}\right)\log_2\left(\frac{\sum_{z\in\varphi^{-1}(y)} m_z^{\mathrm{fcns}(t)}}{\sum_{z\in\varphi^{-1}(y)} m_{z,(a,2)}^{\mathrm{fcns}(t)}}\right),$$

where the last estimate follows from the log-sum inequality (Lemma 1). For every $y \in \mathcal{L}_k^1$ we have

$$\sum_{z\in\varphi^{-1}(y)} m_{z,(a,2)}^{\mathrm{fcns}(t)} = m_{y,(a,0)}^t + m_{y,(a,2)}^t,$$
$$\sum_{z\in\varphi^{-1}(y)} m_z^{\mathrm{fcns}(t)} = m_y^t.$$

Thus, we obtain

$$\begin{aligned}&\sum_{z\in\mathcal{L}_{2k}^{01}}\sum_{\tilde{a}\in\Sigma\times\{0,2\}} m_{z,\tilde{a}}^{\mathrm{fcns}(t)}\log_2\left(\frac{m_z^{\mathrm{fcns}(t)}}{m_{z,\tilde{a}}^{\mathrm{fcns}(t)}}\right)\\ &\leq \sum_{y\in\mathcal{L}_k^1}\sum_{a\in\Sigma}\left(m_{y,(a,0)}^t + m_{y,(a,2)}^t\right)\log_2\left(\frac{m_y^t}{m_{y,(a,0)}^t + m_{y,(a,2)}^t}\right). \qquad (30)\end{aligned}$$

From (iii) and (iv), we obtain

$$\begin{aligned}&\sum_{z\in\mathcal{L}_{2k}^0}\sum_{\tilde{a}\in\Sigma\times\{0,2\}} m_{z,\tilde{a}}^{\mathrm{fcns}(t)}\log_2\left(\frac{m_z^{\mathrm{fcns}(t)}}{m_{z,\tilde{a}}^{\mathrm{fcns}(t)}}\right)\\ &= \sum_{z\in\mathcal{L}_{2k}^0(\mathrm{fcns}(t))}\sum_{a\in\Sigma} m_{z,(a,2)}^{\mathrm{fcns}(t)}\log_2\left(\frac{m_z^{\mathrm{fcns}(t)}}{m_{z,(a,2)}^{\mathrm{fcns}(t)}}\right) + \sum_{z\in\mathcal{L}_{2k}^0} m_{z,(\square,0)}^{\mathrm{fcns}(t)}\log_2\left(\frac{m_z^{\mathrm{fcns}(t)}}{m_{z,(\square,0)}^{\mathrm{fcns}(t)}}\right).\end{aligned}$$

For the first summand, we find analogously as in the previous estimate (30):

$$\begin{aligned}&\sum_{z\in\mathcal{L}_{2k}^0(\mathrm{fcns}(t))}\sum_{a\in\Sigma} m_{z,(a,2)}^{\mathrm{fcns}(t)}\log_2\left(\frac{m_z^{\mathrm{fcns}(t)}}{m_{z,(a,2)}^{\mathrm{fcns}(t)}}\right)\\ &\leq \sum_{y\in\mathcal{L}_k^0}\sum_{a\in\Sigma}\left(m_{y,(a,0)}^t + m_{y,(a,2)}^t\right)\log_2\left(\frac{m_y^t}{m_{y,(a,0)}^t + m_{y,(a,2)}^t}\right). \qquad (31)\end{aligned}$$

For the second summand, we obtain as $k \geq 1$:

$$\begin{aligned}&\sum_{z\in\mathcal{L}_{2k}^0} m_{z,(\square,0)}^{\mathrm{fcns}(t)}\log_2\left(\frac{m_z^{\mathrm{fcns}(t)}}{m_{z,(\square,0)}^{\mathrm{fcns}(t)}}\right)\\ &= \sum_{z\in\mathcal{L}_{2k-1}}\sum_{a\in\Sigma} m_{za0,(\square,0)}^{\mathrm{fcns}(t)}\log_2\left(\frac{m_{za0}^{\mathrm{fcns}(t)}}{m_{za0,(\square,0)}^{\mathrm{fcns}(t)}}\right)\\ &\leq \sum_{y\in\mathcal{L}_{k-1}}\sum_{a\in\Sigma}\left(\sum_{z\in\pi^{-1}(y)} m_{za0,(\square,0)}^{\mathrm{fcns}(t)}\right)\log_2\left(\frac{\sum_{z\in\pi^{-1}(y)} m_{za0}^{\mathrm{fcns}(t)}}{\sum_{z\in\pi^{-1}(y)} m_{za0,(\square,0)}^{\mathrm{fcns}(t)}}\right),\end{aligned}$$

where the last inequality follows from the log-sum inequality. Moreover, for all $y \in \mathcal{L}_{k-1}$ we have

$$\sum_{z \in \pi^{-1}(y)} m^{\mathrm{fcns}(t)}_{za0,(\square,0)} = m^t_{y,(a,0)},$$
$$\sum_{z \in \pi^{-1}(y)} m^{\mathrm{fcns}(t)}_{za0} = m^t_{y,(a,0)} + m^t_{y,(a,2)}.$$

Thus, we find

$$\begin{aligned} &\sum_{z \in \mathcal{L}^0_{2k}} m^{\mathrm{fcns}(t)}_{z,(\square,0)} \log_2 \left(\frac{m^{\mathrm{fcns}(t)}_z}{m^{\mathrm{fcns}(t)}_{z,(\square,0)}} \right) \\ \leq \quad &\sum_{y \in \mathcal{L}_{k-1}} \sum_{a \in \Sigma} m^t_{y,(a,0)} \log_2 \left(\frac{m^t_{y,(a,0)} + m^t_{y,(a,2)}}{m^t_{y,(a,0)}} \right). \end{aligned} \tag{32}$$

Altogether, if we combine the estimates from (29), (30), (31) and (32), we obtain:

$$\begin{aligned} H_{2k}(\mathrm{fcns}(t)) &= \sum_{z \in \mathcal{L}_{2k}} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m^{\mathrm{fcns}(t)}_{z,\tilde{a}} \log_2 \left(\frac{m^{\mathrm{fcns}(t)}_z}{m^{\mathrm{fcns}(t)}_{z,\tilde{a}}} \right) \\ &= \sum_{z \in \mathcal{L}^0_{2k}} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m^{\mathrm{fcns}(t)}_{z,\tilde{a}} \log_2 \left(\frac{m^{\mathrm{fcns}(t)}_z}{m^{\mathrm{fcns}(t)}_{z,\tilde{a}}} \right) + \sum_{z \in \mathcal{L}^{01}_{2k}} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m^{\mathrm{fcns}(t)}_{z,\tilde{a}} \log_2 \left(\frac{m^{\mathrm{fcns}(t)}_z}{m^{\mathrm{fcns}(t)}_{z,\tilde{a}}} \right) \\ &+ \sum_{z \in \mathcal{L}^{11}_{2k}} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m^{\mathrm{fcns}(t)}_{z,\tilde{a}} \log_2 \left(\frac{m^{\mathrm{fcns}(t)}_z}{m^{\mathrm{fcns}(t)}_{z,\tilde{a}}} \right) \\ &\leq \sum_{z \in \mathcal{L}^1_k} \sum_{a \in \Sigma} \left(m^t_{z,(a,0)} + m^t_{z,(a,2)} \right) \log_2 \left(\frac{m^t_z}{m^t_{z,(a,0)} + m^t_{z,(a,2)}} \right) \\ &+ \sum_{z \in \mathcal{L}^0_k} \sum_{a \in \Sigma} \left(m^t_{z,(a,0)} + m^t_{z,(a,2)} \right) \log_2 \left(\frac{m^t_z}{m^t_{z,(a,0)} + m^t_{z,(a,2)}} \right) \\ &+ \sum_{z \in \mathcal{L}_{k-1}} \sum_{a \in \Sigma} m^t_{z,(a,0)} \log_2 \left(\frac{m^t_{z,(a,0)} + m^t_{z,(a,2)}}{m^t_{z,(a,0)}} \right) \\ &\leq \sum_{z \in \mathcal{L}_{k-1}} \sum_{a \in \Sigma} \left(m^t_{z,(a,0)} + m^t_{z,(a,2)} \right) \log_2 \left(\frac{m^t_z}{m^t_{z,(a,0)} + m^t_{z,(a,2)}} \right) \\ &+ \sum_{z \in \mathcal{L}_{k-1}} \sum_{a \in \Sigma} m^t_{z,(a,0)} \log_2 \left(\frac{m^t_{z,(a,0)} + m^t_{z,(a,2)}}{m^t_{z,(a,0)}} \right) \\ &\leq \sum_{z \in \mathcal{L}_{k-1}} \sum_{\tilde{a} \in \Sigma \times \{0,2\}} m^t_{z,\tilde{a}} \log_2 \left(\frac{m^t_z}{m^t_{z,\tilde{a}}} \right) = H_{k-1}(t), \end{aligned}$$

where the last-but-one estimate follows again from the log-sum inequality. This proves Lemma 8. □

Universität Siegen, Germany
E-mail address: {hucke,lohrey,seelbach}@eti.uni-siegen.de