Hypersuccinct Trees – New universal tree source codes for optimal compressed tree data structures and range minima

J. Ian Munro[*] Patrick K. Nicholson[†] Louisa Seelbach Benkner[‡] Sebastian Wild[§]

September 3, 2021

Abstract

We present a new universal source code for distributions of unlabeled binary and ordinal trees that achieves optimal compression to within lower order terms for all tree sources covered by existing universal codes. At the same time, it supports answering many navigational queries on the compressed representation in constant time on the word-RAM; this is not known to be possible for any existing tree compression method. The resulting data structures, "hypersuccinct trees", hence combine the compression achieved by the best known universal codes with the operation support of the best succinct tree data structures.

We apply hypersuccinct trees to obtain a universal compressed data structure for range-minimum queries. It has constant query time and the optimal worst-case space usage of $2n+o(n)$ bits, but the space drops to $1.736n+o(n)$ bits on average for random permutations of n elements, and $2\lg\binom{n}{r}+o(n)$ for arrays with r increasing runs, respectively. Both results are optimal; the former answers an open problem of Davoodi et al. (2014) and Golin et al. (2016).

Compared to prior work on succinct data structures, we do not have to tailor our data structure to specific applications; hypersuccinct trees automatically adapt to the trees at hand. We show that they simultaneously achieve the optimal space usage to within lower order terms for a wide range of distributions over tree shapes, including: binary search trees (BSTs) generated by insertions in random order / Cartesian trees of random arrays, random fringe-balanced BSTs, binary trees with a given number of binary/unary/leaf nodes, random binary tries generated from memoryless sources, full binary trees, unary paths, as well as uniformly chosen weight-balanced BSTs, AVL trees, and left-leaning red-black trees.

[*]University of Waterloo, Canada, imunro@uwaterloo.ca

[†]pat.nicholson@gmail.com

[‡]Universität Siegen, Germany, seelbach@eti.uni-siegen.de, supported by the DFG project LO 748/10-2

[§]University of Liverpool, UK, wild@liverpool.ac.uk

Contents

1. Introduction

As space usage and memory access become the bottlenecks in computation, working directly on a compressed representation ("computing over compressed data") has become a popular field. For text data, substantial progress over the last two decades culminated in compressed text indexing methods that had wide-reaching impact on applications and satisfy strong analytical guarantees. For structured data, the picture is much less developed and clear. In this paper, we develop the analog of entropy-compressed string indices for trees: a data structure that allows one to query a tree stored in compressed form, with optimal query times and space matching the best universal tree codes.

Computing over compressed data became possible by combining techniques from information theory, string compression, and data structures. The central object of study in (classical) information theory is that of a *source* of random strings, whose entropy rate is the fundamental limit for source coding. The ultimate goal in compressing such strings is a *universal code*, which achieves optimal compression (to within lower order terms) for *distributions* of strings from a large class of possible sources *without knowing the used source*.

A classic result in this area is that Lempel-Ziv methods are universal codes for finite-state sources, i.e., sources in which the next symbol's distribution depends on the previous k emitted symbols (see, e.g., [10, § 13]). The same is true for methods based on the Burrows-Wheeler-transform [15] and for grammar-based compression [54]. The latter two results were only shown around 2000, marking a renewed interest in compression methods.

The year 2000 also saw breakthroughs in compressed text indexing, with the first compressed self-indices that can represent a string and support pattern matching queries using $O(nH_0)$ bits of space [41, 42] and $O(nH_k) + o(n \log |\Sigma|)$ bits of space [19] for H_k the kth order empirical entropy of the string (for $k \geq 0$); many improvements have since been obtained on space and query time; (see [67, 6] for surveys and [28] for lower bounds on redundancy; [65, 66] summarizes more recent trends). For strings, computing over compressed data has mainly been achieved.

In this article, we consider structure instead of strings; focusing on one of the simplest forms of structured data: unlabeled binary and ordinal trees. Unlike for strings, the information theory of structured data is still in its infancy. Random sources of binary trees have (to our knowledge) first been suggested and analyzed in 2009 [55]; a more complete formalization then appeared in [83], together with a first universal tree source code.

For trees, computational results predate information-theoretic developments. *Succinct data structures* date back to 1989 [50] and have their roots in storing trees space-efficiently while supporting fast queries. A succinct data structure is allowed to use $\lg U_n(1+o(1))$ bits of space to represent one out of U_n possible objects of size n – corresponding to a uniform distribution over these objects. This has become a flourishing field, and several succinct data structures for ordinal or cardinal (including binary) trees supporting many operations are known [64]. Apart from the exceptions discussed below (in particular [52, 11]), these methods do not achieve any compression beyond $\lg U_n$ no matter what the input is.

At the other end of the spectrum, more recent representations for highly repetitive trees [7, 8, 24, 26, 30, 31] can realize exponential space savings over $\lg U_n$ in extreme cases, but recent lower bounds [71] imply that these methods cannot simultaneously achieve constant time[1] for queries; they are also not known to be succinct when the tree is not highly compressible.

Table 1: Navigational operations on succinct binary trees. (v denotes a node and i an integer).

parent(v)	the parent of v, same as anc($v, 1$)
degree(v)	the number of children of v
left_child(v)	the left child of node v
right_child(v)	the right child of node v
depth(v)	the depth of v, i.e., the number of edges between the root and v
anc(v, i)	the ancestor of node v at depth depth(v) $- i$
nbdesc(v)	the number of descendants of v
height(v)	the height of the subtree rooted at node v
LCA(v, u)	the lowest common ancestor of nodes u and v
leftmost_leaf(v)	the leftmost leaf descendant of v
rightmost_leaf(v)	the rightmost leaf descendant of v
level_leftmost(ℓ)	the leftmost node on level ℓ
level_rightmost(ℓ)	the rightmost node on level ℓ
level_pred(v)	the node immediately to the left of v on the same level
level_succ(v)	the node immediately to the right of v on the same level
node_rank$_X$(v)	the position of v in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$, i.e., in a preorder, postorder, or inorder traversal of the tree
node_select$_X$(i)	the ith node in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$
leaf_rank(v)	the number of leaves before and including v in preorder
leaf_select(i)	the ith leaf in preorder

In this paper, we fill this gap between succinct trees and dictionary-compressed trees by presenting the first data structure for unlabeled binary trees that answers all queries supported in previous succinct data structures (cf. Table 1) in $O(1)$ time and simultaneously achieves optimal compression over the same tree sources as the best previously known universal tree codes. We also extend the tree-source concepts and our data structure to unlabeled ordinal trees. In contrast to previous succinct trees, we give a single, *universal* data structure, the *hypersuccinct trees*[2], that does not need to be adapted to specific classes or distributions of trees.

[1] All running times assume the word-RAM model with word size $w = \Theta(\log n)$.
[2] The name "hypersuccinct trees" is the escalation of the "ultrasuccinct trees" of [52].

Our hypersuccinct trees require only a minor modification of existing succinct tree data structures based on tree covering [34, 44, 16], (namely Huffman coding micro-tree types); the contribution of our work is the careful analysis of the information-theoretic properties of the tree-compression method, the "hypersuccinct code", that underlies these data structures.

As a consequence of our results, we solve an open problem for succinct range-minimum queries (RMQ): Here the task is to construct a data structure from an array $A[1..n]$ of comparable items at preprocessing time that can answer subsequent queries without inspecting A again. The answer to the query $\mathtt{RMQ}(i,j)$, for $1 \le i \le j \le n$, is the index (in A) of the (leftmost) minimum in $A[i..j]$, i.e., $\mathtt{RMQ}(i,j) = \arg\min_{i \le k \le j} A[k]$. We give a data structure that answers RMQ in constant time using the optimal expected space of $1.736n + o(n)$ bits when the array is a random permutation, (and $2n + o(n)$ in the worst case); previous work either had suboptimal space [11] or $\Omega(n)$ query time [37]. We obtain the same (optimal) space usage for storing a binary search tree (BST) built from insertions in random order ("random BSTs" hereafter). Finally, we show that the space usage of our RMQ data structure is also bounded by $2 \lg \binom{n}{r} + o(n)$ whenever A has r increasing runs, and that this is again best possible.

Outline The rest of our article is structured as follows: A comprehensive list of the contributions appears below in Section 2. Section 3 describes our compressed tree encoding. In Section 4, we illustrate the techniques for proving universality of our hypersuccinct code on two well-known types of binary-trees shape distributions – random BSTs and weight-balanced trees – and sketch the extensions necessary for the general results. In Section 5, we present our RMQ data structures. Finally, Section 6 concludes the paper with future directions.

The appendix contains a comprehensive comparison to previous work (Section A), full formal proofs of all results (Part I) and the extension to ordinal trees (Part II). (The proofs in Sections D, E, F, G, I, J, K can all be read in isolation.)

2. Results

In a binary tree, each node has a left and a right child, either of which can be empty ("null"). For a binary tree t we denote by $|t|$ the number of nodes in t. Unless stated otherwise, $n = |t|$. A binary tree source $\mathcal{S}$ emits a tree t with a certain probability $\mathbb{P}_{\mathcal{S}}[t]$ (potentially $\mathbb{P}_{\mathcal{S}}[t] = 0$); we write $\mathbb{P}[t]$ if $\mathcal{S}$ is clear from the context. $\lg(1/0)$ is taken to mean $+\infty$.

Theorem 2.1 (Hypersuccinct binary trees): *Let t be a binary tree over n nodes. The hypersuccinct representation of t supports all queries from Table 1 in $O(1)$ time and uses $|\mathsf{H}(t)| + o(n)$ bits of space, where*

$$|\mathsf{H}(t)| \;\le\; \min\left\{2n+1,\ \min_{\mathcal{S}} \lg\left(\frac{1}{\mathbb{P}_{\mathcal{S}}[t]}\right) + o(n)\right\},$$

and $\mathbb{P}_{\mathcal{S}}[t]$ is the probability that t is emitted by source $\mathcal{S}$. The minimum is taken over all binary-tree sources $\mathcal{S}$ in the following families (which are explained in Table 4):

(i) *memoryless node-type processes,*

(ii) *kth-order node-type processes (for $k = o(\log n)$),*

(iii) *monotonic fixed-size sources,*

Table 2: Overview of random tree sources for binary and ordinal trees.

Name	**Notation**	**Intuition**	**Reference**	**Formal Definition** of $\mathbb{P}[t]$
Memoryless Processes	τ	A binary tree is constructed top-down, drawing each node's *type* (0 = leaf, 1 = left-unary, 2 = binary, 3 = right-unary) i.i.d. according to the distribution $(\tau_0, \tau_1, \tau_2, \tau_3)$.	Sec. D Eq. (3) [11, 37]	$\mathbb{P}[t] = \prod_{v \in t} \tau(\mathrm{type}(v))$
Higher-order Processes	$(\tau_z)_z$	A binary tree is constructed top-down, drawing node v's *type* according to $\tau_{h_k(v)} : \{0,1,2,3\} \to [0,1]$, which depends on the types of the k closest *ancestors* of v.	Sec. D Eq. (3)	$\mathbb{P}[t] = \prod_{v \in t} \tau_{h_k(v)}(\mathrm{type}(v))$
Fixed-size Binary Tree Sources	$\mathcal{S}_{fs}(p)$	A binary tree of size n is constructed top-down, asking source p at each node for its left- and right *subtree size*.	Sec. E Eq. (6) [83, 25, 76]	$\mathbb{P}[t] = \prod_{v \in t} p(\lvert t_\ell(v)\rvert, \lvert t_r(v)\rvert)$ $t_{\ell/r}(v)$= left/right subtree of v
Fixed-height Binary Tree Sources	$\mathcal{S}_{fh}(p)$	A binary tree of height h is constructed top-down, asking source p at each node for a left and right subtree *height*.	Sec. E Eq. (7) [83, 25]	$\mathbb{P}[t] = \prod_{v \in t} p(h(t_\ell(v)), h(t_r(v)))$ $h(t)$ = height of t
Uniform Subclass Sources	$\mathcal{U}_{\mathcal{P}}$	A binary tree is drawn uniformly at random from the set $\mathcal{T}_n(\mathcal{P})$ of all binary trees of size n that satisfy property $\mathcal{P}$.	Sec. F Eq. (9)	$\mathbb{P}[t] = \dfrac{1}{\lvert \mathcal{T}_n(\mathcal{P})\rvert}$
Memoryless Ordinal Tree Sources	d	An ordinal tree is constructed top-down, drawing each node v's degree $\deg(v)$ according to distribution $d = (d_0, d_1, \ldots)$.	Sec. I Eq. (14)	$\mathbb{P}[t] = \prod_{v \in t} d_{\deg(v)}$
Fixed-size Ordinal Tree Sources	$\mathfrak{S}_{fs}(p)$	An ordinal tree of size n is constructed top-down, asking source p at each node for the number and sizes of the subtrees.	Sec. J	$\mathbb{P}[t] = \prod_{v \in t} p(\lvert t_1[v]\rvert, \ldots, \lvert t_{\deg(v)}[v]\rvert)$

(iv) *worst-case fringe-dominated fixed-size sources,*

(v) *monotonic fixed-height sources,*

(vi) *worst-case fringe-dominated fixed-height sources,*

(vii) *tame uniform subclass sources.*

Corollary 2.2 (Hypersuccinct binary trees: Examples & Empirical entropies): *Hypersuccinct trees achieve optimal compression to within lower order terms for all example distributions listed in Table 3. Moreover, for every binary tree t, we have:*

(i) $|\mathsf{H}(t)| \le H_k^{\mathrm{type}}(t) + o(n)$ *with* $H_k^{\mathrm{type}}(t)$ *the (unnormalized) kth-order empirical entropy of node types (leaf, left-unary, binary, or right-unary) for* $k = o(\log n)$.

(ii) $|\mathsf{H}(t)| \le H_{st}(t) + o(n)$ *with* $H_{st}(t)$ *the "subtree-size entropy", i.e., the sum of the logarithm of the subtree size of v for all nodes v in t, (a.k.a. the splay-tree potential).*

The hypersuccinct code is a *universal code* for the families of binary-tree sources listed in Theorem 2.1 with bounded maximal pointwise redundancy. We also present a more general class of sources, for which our code achieves $o(n)$ *expected* redundancy in the appendix; see also Table 4.

Table 3: An overview over the concrete examples of tree-shape distributions that our hypersuccinct code compresses optimally (up to lower-order terms).

Tree-Shape Distribution	Entropy	Corresponding Source	Def.	Result
(Uniformly random) binary trees of size n	$2n$	Memoryless binary, monotonic fixed-size binary	Ex. D.2 Ex. E.2	Cor. D.10 Cor. E.22
(Uniformly random) full binary trees of size n	n	Memoryless binary	Ex. D.3	Cor. D.10
(Uniformly random) unary paths of length n	n	Memoryless binary	Ex. D.4	Cor. D.10
(Uniformly random) Motzkin trees of size n	$1.585n$	Memoryless binary	Ex. D.5	Cor. D.10
Binary search trees generated by insertions in random order ("random BSTs")	$1.736n$	Monotonic fixed-size binary	Ex. E.1	Cor. E.22 Cor. E.31
Binomial random trees	$P(\lg n)n$[a]	Average-case fringe-dominated fixed-size binary	Ex. E.3	Cor. E.31
Almost paths	—[b]	Monotonic fixed-size binary	Ex. E.4	Cor. E.22
Random fringe-balanced binary search trees	—[b]	Average-case fringe-dominated fixed-size binary	Ex. E.5	Cor. E.31
(Uniformly random) AVL trees of height h	—[b]	Worst-case fringe-dominated fixed-height binary	Ex. E.6	Cor. E.31
(Uniformly random) weight-balanced binary trees of size n	—[b]	Worst-case fringe-dominated fixed-size binary	Ex. F.4	Cor. E.31
(Uniformly random) AVL trees of size n	$0.938n$	Uniform-subclass	Ex. F.2	Cor. F.7
(Uniformly random) left-leaning red-black trees of size n	$0.879n$	Uniform-subclass	Ex. F.3	Cor. F.7
(Uniformly random) full m-ary trees of size n	$\lg(\frac{m}{m-1})n$	Memoryless ordinal	Ex. I.2	Cor. I.5
Uniform composition trees	—[b]	Monotonic fixed-size ordinal	Ex. J.2	Cor. J.9
Random LRM-trees	$1.736n$	Monotonic fixed-size ordinal	Ex. J.3	Cor. J.9

[a] Here P is a nonconstant, continuous, periodic function with period 1.
[b] No (concise) asymptotic approximation known.

Table 4: Sufficient conditions under which we show universality of our hypersuccinct code H for binary trees.

Family of sources	Restriction	Redundancy	Def.	Reference
Memoryless node-type	—	$O(n \log\log n / \log n)$	Sec. D	Thm. D.9
kth-order node-type	—	$O((nk + n \log\log n)/\log n)$	Sec. D	Thm. D.9
Monotonic fixed-size	$p(\ell, r) \ge p(\ell+1, r)$ and $p(\ell, r) \ge p(\ell, r+1)$ for all $\ell, r \in \mathbb{N}_0$	$O(n \log\log n / \log n)$	Def. E.7	Thm. E.21
Worst-case fringe-dominated fixed-size	$n_{\ge B}(t) = o(n/\log\log n)$ for all t with $\mathbb{P}[t] > 0$; $n_{\ge B}(t) = \#$nodes with subtree size in $\Omega(\log n)$	$O(n_{\ge B}(t) \log\log n + n \log\log n/\log n)$	Def. E.10	Thm. E.26
Weight-balanced fixed-size	$\sum_{\frac{n}{c} \le \ell \le n - \frac{n}{c}} p(\ell-1, n-\ell-1) = 1$ for constant $c \ge 3$	$O(n \log\log n / \log n)$	Def. E.15	Cor. E.29
Average-case fringe-dominated fixed-size	$\mathbb{E}[n_{\ge B}(T)] = o(n/\log\log n)$ for random T generated by source $\mathcal{S}$	$O(n_{\ge B}(t) \log\log n + n \log\log n/\log n)$[a]	Def. E.9	Thm. E.25
Monotonic fixed-height	$p(\ell, r) \ge p(\ell+1, r)$ and $p(\ell, r) \ge p(\ell, r+1)$ for all $\ell, r \in \mathbb{N}_0$	$O(n \log\log n / \log n)$	Def. E.7	Thm. E.21
Worst-case fringe-dominated fixed-height	$n_{\ge B}(t) = o(n/\log\log n)$ for all t with $\mathbb{P}[t] > 0$	$O(n_{\ge B}(t) \log\log n + n \log\log n/\log n)$	Def. E.10	Thm. E.26
Tame uniform-subclass	class of trees $\mathcal{T}_n(\mathcal{P})$ is hereditary (i.e., closed under taking subtrees), $n_{\ge B}(t) = o(n/\log\log n)$ for $t \in \mathcal{T}_n(\mathcal{P})$, $\lg \lvert\mathcal{T}_n(\mathcal{P})\rvert = cn + o(n)$ for constant $c > 0$, heavy-twigged: if v has subtree size $\Omega(\log n)$, v's subtrees have size $\omega(1)$	$o(n)$	Def. F.1	Thm. F.6

[a] Stated redundancy is achieved in expectation for a random tree t generated by the source.

To our knowledge, the list in Theorem 2.1 is a comprehensive account of *all* concrete binary-tree sources for which any universal code is known. Remarkably, in all cases the bounds on redundancies proven for the hypersuccinct code are identical (up to constant factors) to those known for existing universal binary-tree codes. *Our hypersuccinct code thus achieves the same compression as all previous universal codes, but simultaneously supports constant-time queries on the compressed representation with $o(n)$ overhead.*

In terms of queries, previous solutions either have suboptimal query times [7, 8, 26], higher space usage [71], or rely on tailoring the representation to a specific subclass of trees [52, 16] to achieve good space and time for precisely these instances, but they fail to generalize to other use cases. Some also do not support all queries. We give a detailed comparison with the state of the art in Section A.

We focus here on our results for binary trees. In the appendix, Part II, we extend the above notions of tree sources (except fixed-height sources) to ordinal trees, which has not been done to our knowledge. Moreover, we extend both our code and data structure to ordinal trees, and show their universality for these sources.

3. From Tree Covering to Hypersuccinct Trees

Our universally compressed tree data structures are based on *tree covering* [34, 44, 16]: A (binary or ordinal) tree t is decomposed into *mini trees*, each of which is further decomposed into *micro trees*; the size of the latter, $B = B(n) = \lg n/8$, is chosen so that we can tabulate all possible shapes of micro trees and the answers to various micro-tree-local queries in one global lookup table (the "Four-Russian Table" technique). For each micro tree, its local shape is stored, e.g., using the balanced-parenthesis (BP) encoding, using a total of exactly $2n$ bits (independent of the tree shape). Using additional data structures occupying only $o(n)$ bits of space, a long list of operations can be supported in constant time (Table 1). The space usage of this representation is optimal to within lower order terms for the worst case, since $\lg C_n \sim 2n$ bits are necessary to distinguish all $C_n = \binom{2n}{n}/(n+1)$ trees of n nodes. (This worst-case bound applies both to ordinal trees and binary trees).

A core observation is that the dominant space in tree-covering data structures comes from storing the micro-tree types, and these can be further compressed using a different code. This has been used in an ad-hoc manner for specific tree classes [16, 11, 29], but has not been investigated systematically. A natural idea is to use a Huffman code for the micro tree types to simultaneously beat the compression of all these special cases; we dub this as the "Four Russians and One American"[3] trick. Applying it to the data structures based on the Farzan-Munro tree-covering algorithm [16] yields our hypersuccinct trees.

The main contribution of our present work is the careful analysis of the potential of the Four Russians and One American trick for (binary and ordinal) tree source coding. As an immediate corollary, we obtain a single data structure that achieves optimal compression for all special cases covered in previous work, plus a much wider class of distributions over trees for which no efficient data structure was previously known.

Our analysis builds on previous work on tree compression, specifically DAG compression and

[3] It deems us only fair to do D. A. Huffman the same questionable honor of reducing the person to a country of residence that V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev have experienced ever since their table-lookup technique has become known as the "Four-Russians trick".

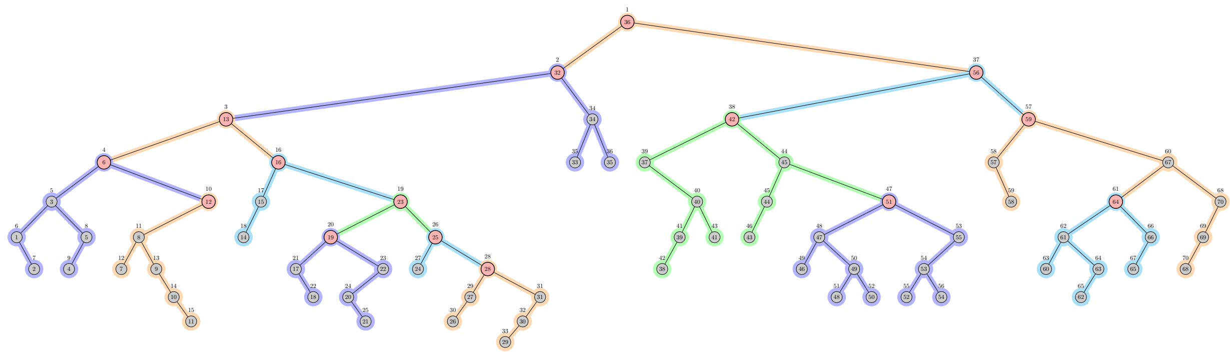


Figure 1: Example binary tree with $n = 70$ nodes and micro trees computed by the Farzan-Munro tree-covering algorithm [16] with parameter $B = 6$. For the reader's convenience, the algorithm is summarized in the appendix (Section B.3). The micro trees are indicated by colors. The algorithm guarantees that each node is part of exactly one micro tree and that each micro tree has at most three edges shared with other micro trees, namely to a parent, a left- and a right-child micro tree.

tree straight-line programs (TSLPs) [56]. Our core idea is to interpret (parts of the) tree-covering data structures as a *code* for trees, the "hypersuccinct code": it stores the type, i.e., the local shape, of all micro trees separately from how they interface to form the entire tree (details are given in the appendix, Section C for binary trees and Section H for ordinal trees). Intuitively, our hypersuccinct code is a restricted version of a grammar-based tree code, where we enforce having nonterminals for certain subtrees;[4] we strengthen and extend existing universality proofs from general grammar-based tree codes to the restricted hypersuccinct code.

4. Universality for Fixed-Size Sources

In this section, we sketch the proof that our hypersuccinct trees achieve optimal compression for two exemplary tree-shape distributions: random binary search trees and uniform weight-balanced trees (defined below). These examples serve to illustrate the proof techniques and to showcase the versatility of the approach. The extension to the general sufficient conditions from Table 4 and full details of computations are spelled out in the appendix.

By *random BSTs*, we mean the distribution of tree shapes obtained by successively inserting n keys in random order into an (initially empty) unbalanced binary search tree (BST). We obtain random BSTs from a fixed-size tree source $\mathcal{S}_{fs}(p_{bst})$ with $p_{bst}(\ell, n-1-\ell) = \frac{1}{n}$ for all $\ell \in \{0, \dots, n-1\}$ and $n \in \mathbb{N}_{\geq 1}$, i.e., making every possible split equally likely. (Any left subtree size ℓ is equally likely in a random BST of a given size n.) Hence, $\mathbb{P}[t] = \prod_{v \in t} 1/|t[v]|$ where $t[v]$ is the subtree rooted at v and $|t[v]|$ its size (in number of nodes).

The second example are the shapes of uniformly random *weight-balanced BSTs* (BB[α]-trees, [69]): A binary tree t is α-weight-balanced if we have for every node v in t that $\min\{|t_\ell[v]|, |t_r[v]|\} +$

[4] Differences in technical details make the direct comparison difficult, though: in TSLPs, holes in contexts must be stored (and encoded) alongside the local shapes as they are both part of the right-hand side of productions; in our hypersuccinct code, we separately encode the shapes of micro trees and the positions of portals, potentially gaining a small advantage. Our comment thus remains a motivational hint as to why similar analysis techniques are useful in both cases, but falls short of providing a formal reduction.

$1 \geq \alpha(|t[v]|+1)$. Here $t_\ell[v]$ resp. $t_r[v]$ are the left resp. right subtrees of $t[v]$. We denote the set of α-weight-balanced trees of size n by $\mathcal{T}_n(\mathcal{W}_\alpha)$. We obtain random α-weight-balanced trees from another fixed-size source $\mathcal{S}_{fs}(p_{wb})$ with

$$p_{wb}(\ell, n-1-\ell) \;=\; \begin{cases} \dfrac{|\mathcal{T}_\ell(\mathcal{W}_\alpha)||\mathcal{T}_{n-1-\ell}(\mathcal{W}_\alpha)|}{|\mathcal{T}_n(\mathcal{W}_\alpha)|} & \text{if } \min\{\ell+1, n-\ell\} \geq \alpha(n+1), \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to check that this yields the uniform probability distribution on $\mathcal{T}_n(\mathcal{W}_\alpha)$, i.e., with $\mathbb{P}[t] = 1/|\mathcal{T}_n(\mathcal{W}_\alpha)|$ for $t \in \mathcal{T}_n(\mathcal{W}_\alpha)$ and $\mathbb{P}[t] = 0$ otherwise. We note that computing $|\mathcal{T}_n(\mathcal{W}_\alpha)|$ is a formidable challenge in combinatorics, but we never have to do so; we only require the *existence* of the fixed-size source for weight-balanced BSTs.

The *hypersuccinct code* $\mathsf{H}(t)$ is formed by partitioning the nodes of a given binary tree t into $m = \Theta(n/\log n)$ micro trees $\mu_1, \ldots, \mu_m$, each of which is a connected subtree of at most $\mu = O(\log n)$ nodes; an example is shown in Figure 1. Previous work on tree covering shows how to compute these and how to encode everything but the local shape of the micro trees in $o(n)$ bits of space [16]. (For a mere encoding, $O(n \log\log n/\log n)$ bits suffice; see Section C).

The dominant part of the hypersuccinct code is the list of *types* of all micro trees, i.e., the (local) shapes of the induced subtrees formed by the set of nodes in the micro trees. Let C be a Huffman code for the string $\mu_1, \ldots, \mu_m$, where we identify micro trees with their types. For a variety of different tree sources $\mathcal{S}$, we can prove that $\sum_{i=1}^m |C(\mu_i)|$, the total length of codewords for the micro trees, is upper bounded by $\lg(1/\mathbb{P}[t])$ + lower-order terms, where $\mathbb{P}[t]$ is the probability that t is emitted by $\mathcal{S}$; this is the best possible code length to within lower order terms achievable for that source. We will now show this for our two example distributions.

4.1. Random BSTs

The proof consists of four steps that can be summarized as follows:

Step 1	**Step 2**	**Step 3**	**Step 4**
Construct a source-specific micro-tree encoding $D_\mathcal{S}\colon \{\mu_1, \ldots, \mu_m\} \to \{0,1\}^\star$ **Goal:** $\lvert D_\mathcal{S}(\mu_i)\rvert \approx \lg(1/\mathbb{P}[\mu_i])$	By optimality of Huffman codes: $\sum_{i=1}^m \lvert C(\mu_i)\rvert \leq \sum_{i=1}^m \lvert D_\mathcal{S}(\mu_i)\rvert$	Use properties of $\mathcal{S}$ to show that $\prod_{i=1}^m \mathbb{P}[\mu_i] \gtrsim \mathbb{P}[t]$	Conclude $\sum_{i=1}^m \lvert C(\mu_i)\rvert \approx \lg(1/\mathbb{P}[t])$

Steps 2 and 4 do not depend on the source and indeed follow immediately; Steps 1 and 3 are the creative parts. Ignoring proper tracing of error terms, the result then follows as

$$|\mathsf{H}(t)| \;\sim\; \sum_{i=1}^n |C(\mu_i)| \;\leq\; \sum_{i=1}^n |D_\mathcal{S}(\mu_i)| \;\leq\; \sum_{i=1}^n \lg(1/\mathbb{P}[\mu_i]) \;\lesssim\; \lg(1/\mathbb{P}[t]).$$

Let us consider $\mathcal{S}_{fs}(p_{bst})$, the fixed-size source producing (shapes of) random BSTs, and address these steps independently.

Our task in Step 1 is to find a code $D_\mathcal{S}$ for the micro-tree types that can occur in t, so that $|D_\mathcal{S}(\mu_i)| = \lg(1/|\mathbb{P}_\mathcal{S}[\mu_i]) + O(\log\log n)$. This code may rely on the decoder to have knowledge of $\mathcal{S}$.

For random BSTs, $D_{\mathcal{S}}(t)$ can be constructed as follows: We initially store n using Elias gamma code[5] and then, following a depth-first (preorder) traversal of the tree, we encode the size of the left subtree using *arithmetic coding*. Inductively, the size of the currently encoded node is always known, and the source-specific code is allowed to use the probability distributions hardwired into $\mathcal{S}$ without storing them; for random BSTs, we simply encode a number uniformly distributed in $[0..s-1]$ at a node with subtree size s, using exactly $\lg s$ bits. Apart from storing the initial size and the small additive overhead from arithmetic coding, the code length of this "depth-first arithmetic tree code" is best possible: $|D_{\mathcal{S}}(t)| \le \lg(1/\mathbb{P}[t]) + O(\log |t|)$. This concludes Step 1.

For Step 3, we have to show that the probability for the entire tree t is at most the product of the probabilities for all micro-trees. Recall that $\mu_1, \dots, \mu_m$ are the micro trees in t. We can write $\mathbb{P}[t]$ as a product over contributions of individual nodes, and can collect factors in $\mathbb{P}[t]$ according to micro trees; this works for any fixed-size source. For random BSTs, we can use the "monotonicity" of node contributions to show

$$\mathbb{P}[t] \;=\; \prod_{v \in t} \frac{1}{|t[v]|} \;=\; \prod_{i=1}^{m} \prod_{v \in \mu_i} \frac{1}{|t[v]|} \;\le\; \prod_{i=1}^{m} \prod_{v \in \mu_i} \frac{1}{|\mu_i[v]|} \;=\; \prod_{i=1}^{m} \mathbb{P}[\mu_i].$$

That completes Step 3, and hence the proof that $|\mathsf{H}(t)| \le \mathbb{P}_{\mathcal{S}_{fs}(p_{bst})}[t] + o(n)$.

4.2. Weight-balanced trees

Let us now consider uniformly random weight-balanced trees, i.e., the source $\mathcal{S} = \mathcal{S}_{fs}(p_{wb})$. We would like to follow the same template as above; however, this is not possible: Step 3 from above is in general not true anymore. The reason is that it is not clear whether the "non-fringe" micro trees, i.e., those that do not contain all descendants of the micro-tree root, have non-zero probability under $\mathcal{S}$. (A subtree of a tree is called *fringe*, if it consists of a node and all its descendants). Such micro trees will also make Step 1 impossible as they would require a code length of 0. While this issue is inevitable in general (Remark 4.2), we can under certain conditions circumvent Steps 1 and 3 altogether by directly bounding $\sum_{i=1}^{m} |D_{\mathcal{S}}(\mu_i)| \le \mathbb{P}[t] + o(n)$.

As a first observation, note that it suffices to have $|D_{\mathcal{S}}(\mu_i)| = \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i]) + O(\log\log n)$ for *all but a vanishing fraction* of the micro trees in any tree t; then we can still hope to show $\sum_{i=1}^{m} |D_{\mathcal{S}}(\mu_i)| \le \mathbb{P}[t] + o(n)$ overall. Second, it is known [24] that weight-balanced trees are *"fringe dominated"* in the following sense: Denoting by $n_{\ge B}(t)$ the number of "heavy" nodes, i.e., v in t with $|t[v]| \ge B = \lg n/8$, we have $n_{\ge B}(t) = O(n/B) = o(n)$ for every weight-balanced tree $t \in \mathcal{T}_n(\mathcal{W}_\alpha)$. Since only a vanishing fraction of nodes are heavy, one might hope that also only a vanishing fraction of micro trees are non-fringe, making the above route succeed. Unfortunately, that is not the case; the non-fringe micro trees can be a constant fraction of all micro trees.

Notwithstanding this issue, a more sophisticated micro-tree code $D_{\mathcal{S}}$ allows us to proceed. $D_{\mathcal{S}}$ encodes any *fringe* micro tree using a depth-first arithmetic code as for random BSTs. Any non-fringe micro tree μ_i, however, is broken up into the subtree of heavy nodes, the "boughs" of μ_i, and (fringe) subtrees $f_{i,j}$ hanging off the boughs. It is a property of the Farzan-Munro algorithm that every micro-tree root is heavy, hence all $f_{i,j}$ are indeed entirely contained within μ_i.

$D_{\mathcal{S}}(\mu_i)$ then first encodes the bough nodes using 2 bits per node (using a BP representation for the boughs subtree) and then appends the depth-first arithmetic code for the $f_{i,j}$ (in left-to-right

[5]Elias gamma code $\gamma : \mathbb{N} \to \{0,1\}^\star$ encodes an integer $n \ge 1$ using $2\lfloor \lg n \rfloor + 1$ bits by prefixing the binary representation of n with that representation's length encoded in unary.

order). While this does not actually achieve $|D_{\mathcal{S}}(\mu_i)| \approx \lg(1/\mathbb{P}[\mu_i])$ for entire micro trees μ_i, it does so for all the fringe subtrees $f_{i,j}$. Any node not contained in a fringe subtree $f_{i,j}$ must be part of a bough and hence heavy; by the fringe-dominance property, these nodes form a vanishing fraction of all nodes and hence contribute $o(n)$ bits overall. This shows that $|\mathsf{H}(t)| \le \mathbb{P}_{\mathcal{S}_{fs}(p_{wb})}[t] + o(n)$.

Remark 4.1 (A simple code whose analysis isn't): It is worth pointing out that the source specific code $D_{\mathcal{S}}$ is only a vehicle for the *analysis* of $|\mathsf{H}(t)|$; the complicated encodings $D_{\mathcal{S}}$ do not ever need to be computed when using our codes or data structures.

4.3. Other Sources

For memoryless sources, the analysis follows the four-step template, and is indeed easier than the random BSTs since Step 3 becomes trivial. For higher-order sources, in order to know the node types of the k closest ancestors (in t) of all nodes of depth $\le k$ in μ_i, we prefix the depth-first arithmetic code by the node types of the k closest ancestors of the root of μ_i. Then the k ancestor types are known inductively for all nodes in a preorder traversal of μ_i.

The tame uniform-subclass sources require the most technical proof, but it is conceptually similar to the weight-balanced trees from above. The source-specific encoding for fringe subtrees is trivial here: we can simply use the rank in an enumeration of all trees of a given size, prefixed by the size of the subtree. Using the tameness conditions, one can show that a similar decomposition into boughs and fringe subtrees yields an optimal code length for almost all nodes. Details are deferred to the appendix (Section F).

* * *

Together with the observations from Section 3 this yields Theorem 2.1. We obtained similar results for ordinal trees; details are deferred to the appendix (Part II).

Remark 4.2 (Restrictions are inevitable): We point out that some restrictions like the ones discussed above cannot possibly be overcome in general. Zhang, Yang, and Kieffer [83] prove that the unrestricted class of fixed-size sources (leaf-centric binary tree sources in their terminology) does not allow a universal code, even when only considering expected redundancy. The same is true for unrestricted fixed-height and uniform-subclass sources. While each is a natural formalism to describe possible binary-tree sources, additional conditions are strictly necessary for any interesting compression statements to be made. Our sufficient conditions are the weakest such restrictions for which any universal source code is known to exist ([83, 25, 76]), even without the requirement of efficient queries.

5. Hypersuccinct Range-Minimum Queries

We now show how hypersuccinct trees imply an optimal-space solution for the range-minimum query (RMQ) problem.[6] Let $A[1..n]$ store the numbers $x_1, \ldots, x_n$, i.e., x_j is stored at index j for $1 \le j \le n$. While duplicates naturally arise in some applications, e.g., in the longest-common extension (LCE) problem, we assume here that $x_1, \ldots, x_n$ are n distinct numbers to simplify the presentation. However, our RMQ solution works regardless of which minimum-value index is to be returned so long as the tie breaking rule is deterministic and fixed at construction time.

[6] A technical report containing preliminary results for random RMQ, but including more details on the data structure aspects of our solution, can be found on arXiv [63].

5.1. Cartesian Trees

The *Cartesian tree* T for $x_1, \ldots, x_n$ (resp. for $A[1..n]$) is a binary tree defined recursively as follows: If $n = 0$, it is the empty tree ("null"). Otherwise it consists of a root whose left child is the Cartesian tree for $x_1, \ldots, x_{j-1}$ and its right child is the Cartesian tree for $x_{j+1}, \ldots, x_n$ where j is the position of the minimum, $j = \arg\min_k A[k]$. A classic observation of Gabow et al. [23] is that range-minimum queries on A are equivalent to lowest-common-ancestor (LCA) queries on T when identifying nodes with their inorder rank:

$$\mathtt{RMQ}_A(i,j) \quad = \quad \mathtt{node\_rank_{IN}}\Big(\mathtt{LCA}\big(\mathtt{node\_select_{IN}}(i), \mathtt{node\_select_{IN}}(j)\big)\Big).$$

We can thus reduce an RMQ instance (on an arbitrary input) to an LCA instance on binary trees of the same size; (the number of nodes in T equals the length of the array).

5.2. Random RMQ

We first consider the random permutation model for RMQ: Every (relative) ordering of the elements in $A[1..n]$ is equally likely. Without loss of generality, we identify the n elements with their ranks, i.e., $A[1..n]$ contains a random permutation of $[1..n]$. We refer to this as a random RMQ instance.

We can characterize the distribution of the Cartesian tree associated with such a random RMQ instance: Since the minimum in a random permutation is located at every position $i \in [n]$ with probability $\frac{1}{n}$, the inorder index of the root is uniformly distributed in $[n]$. Apart from renaming, the subarrays $A[1..i-1]$ (resp. $A[i+1..n]$) contain a random permutation of $i-1$ (resp. $n-i$) elements, and these two permutations are independent of each other conditional on their sizes. Cartesian trees of random RMQ instances thus have the same distribution as random BSTs, and in particular shape t arises with probability $\mathbb{P}[t] = \prod_{v \in t} \frac{1}{|t[v]|}$. The former are also known as random increasing binary trees [22, Ex. II.17 & Ex. III.33]).

Since the sets of answers to range-minimum queries is in bijection with Cartesian trees, the entropy H_n of the distribution of the shape of the Cartesian tree (and hence random BSTs) gives an information-theoretic lower bound for the space required by any RMQ data structure (in the encoding model studied here). Kieffer, Yang and Szpankowski [55] show[7] that the entropy of random BSTs $H_n = \mathbb{E}_T\big[\lg(1/\mathbb{P}[T])\big] = \mathbb{E}_T\big[\sum_{v \in T} \lg(|T[v]|)\big]$ is

$$H_n \quad = \quad \lg(n) + 2(n+1)\sum_{i=2}^{n-1} \frac{\lg i}{(i+2)(i+1)} \quad \sim \quad 2n\sum_{i=2}^{\infty} \frac{\lg i}{(i+2)(i+1)} \quad \approx \quad 1.7363771n. \qquad (1)$$

With these preparations, we are ready to prove our first result on range-minimum queries.

Corollary 5.1 (Average-case optimal succinct RMQ): *There is a data structure that supports (static) range-minimum queries on an array A of n (distinct) numbers in $O(1)$ worst-case time and which occupies $H_n + o(n) \approx 1.736n + o(n)$ bits of space on average over all possible permutations of the elements in A. The worst case space usage is $2n + o(n)$ bits.*

[7] Hwang and Neininger [48] showed already in 2002 that the quicksort recurrence can be solved explicitly for arbitrary toll functions. H_n satisfies this recurrence with toll function $\lg n$, hence they implicitly proved Equation (1).

Proof: We construct a hypersuccinct tree on the Cartesian tree for A. It supports $\texttt{node\_rank}_{\texttt{IN}}$, $\texttt{node\_select}_{\texttt{IN}}$, and $\texttt{LCA}$ in $O(1)$ time and thus $\texttt{RMQ}$ in constant time without access to A. By Corollary 2.2, the space usage of hypersuccinct trees is at most $\min\{2n, \lg(1/\mathbb{P}[t])\} + o(n)$ for $\mathbb{P}[t] = \prod_{v \in t} \frac{1}{|t[v]|}$. By the above observations, this is the probability to obtain t as the Cartesian trees of a random permutation, so we store t with maximal pointwise redundancy of $o(n)$, hence also $o(n)$ expected redundancy over the entropy $H_n \sim 1.736n$. □

5.3. RMQ with Runs

A second example of compressible RMQ instances results from partially sorted arrays. Suppose that $A[1..n]$ can be split into r runs, i.e., maximal contiguous ranges $[j_i, j_{i+1} - 1]$, $(i = 1, \ldots, r$ with $j_1 = 1$ and $j_{r+1} = n + 1)$, so that $A[j_i] \le A[j_i + 1] \le \cdots \le A[j_{i+1} - 1]$.

Theorem 5.2 (Lower bound for RMQ with runs): *Any range-minimum data structure in the encoding model for an array of length n that contains r runs must occupy at least $\lg N_{n,r} \ge 2\lg\binom{n}{r} - O(\log n)$ bits of space where $N_{n,r} = \frac{1}{n}\binom{n}{r}\binom{n}{r-1}$ are the Narayana numbers.*

The proof follows from a bijection between Cartesian trees on sequences of length n with exactly r runs and mountain-valley diagrams (a.k.a. Dyck paths) of length $2n$ with exactly r "peaks"; the latter is known to be counted by the *Narayana numbers* [49]. Details are given in Section G in the appendix.

Corollary 5.3 (Optimal succinct RMQ with runs): *There is a data structure that supports (static) range-minimum queries on an array A of n numbers that consists of r runs in $O(1)$ worst-case time and which occupies $2\lg\binom{n}{r} + o(n) \le 2n + o(n)$ bits of space.*

This follows from the observation that a node's type in the Cartesian tree, i.e., whether or not its left resp. right child is empty, closely reflects the runs in A: A binary node marks the beginning of a non-singleton run, a leaf node marks the last position in a non-singleton run, a right-unary node (i.e., left child empty, right child nonempty) is a middle node of a run, and a left-unary node corresponds to a singleton run. With $s \in [0..r]$ the number of singleton runs, we can bound the space for a hypersuccinct tree in terms of its empirical node-type entropy by $H_0^{\text{type}}(T) + o(n) = nH\left(\frac{r-s}{n}, \frac{s}{n}, \frac{n-2r+s}{n}, \frac{r-s}{n}\right) + o(n)$, which can be shown to be no more than $2\lg\binom{n}{r} + o(n)$ for any value of s; again, details are deferred to Section G.

* * *

We close by pointing out that hypersuccinct trees simultaneously achieve the optimal bounds for RMQ on random permutations and arrays with r runs without taking explicit precautions for either. The same is true for any other shape distributions of Cartesian trees that can be written as one of the sources from Table 4.

6. Conclusion

We presented the first succinct tree data structures with optimally adaptive space usage for a large variety of random tree sources, both for binary trees and for ordinal trees. This is an important step towards the goal of efficient computation over compressed structures, and has immediate applications, e.g., as illustrated above for the range-minimum problem.

A goal for future work is to reduce the redundancy of $o(n)$, which becomes dominant for sources with sublinear entropy. While this has been considered for tree covering in principle [77], many details remain to be thoroughly investigated.

For very compressible trees, the space savings in hypersuccinct trees are no longer competitive. On the other hand, with current methods for random access on dictionary-compressed sequences, constant-time queries are not possible in the regime of mildly compressible strings; the same applies to known approaches to represent trees. An interesting question is whether these opposing approaches can be combined in a way to complement each other's strengths. We leave this direction for future work.

Acknowledgements. We thank Conrado Martínez and Markus Lohrey for valuable discussions and feedback on earlier drafts of this paper.

This work has been supported in part by the Canada Research Chairs Programme and an NSERC Discovery Grant, the DFG research project LO 748/10-2 (QUANT-KOMP), and the NeST (Network Sciences and Technologies) EEECS School initiative of University of Liverpool.

Appendix

In the appendix, we give full formal proof for all claims presented in the previous sections (in particular Section 2) of the paper. Furthermore, we present a comprehensive discussion of related work and applications of hypersuccinct trees.

The appendix is structured as follows: Section A puts the work in broader context and surveys relevant results from information theory, tree compression, and succinct data structures. In Section B, we introduce common notations, give basic definitions and recall important properties with respect to trees and succinct data structures. Additionally, we briefly recapitulate the Farzan-Munro algorithm from [16].

Part I gives full details for our results and proofs on binary trees: Section C formally defines our compressed tree encoding, respectively, data structure (the *hypersuccinct trees*). In Section D to Section F we show that our hypersuccinct tree encoding is universal with respect to the various types of tree sources: In Section D, we formally define memoryless and higher order tree sources and prove our results with respect to these sources. In Section E, we consider fixed-size and fixed-height binary tree sources: In particular, the results and proof sketches presented in Section 4.1 and Section 4.2 with respect to random BSTs and weight-balanced BSTs follow as special cases from more general results (Theorem E.21 and Theorem E.26) proven in Section E. In Section F, we introduce and prove our results with respect to uniform subclass sources.

Part II presents our results for ordinal trees: We describe our hypersuccinct tree encoding, respectively, data structure in Section H. Furthermore, in Section I and Section J we generalize the concepts and results with respect to memoryless/higher order and fixed-size tree sources from binary to ordinal trees. Additionally, we show that our hypersuccinct encoding achieves the so-called *Label-Shape-Entropy*, a concept introduced in [46] as a measure of empirical entropy for both labeled and unlabeled trees, in Section K. For the reader's convenience, Section L has a comprehensive list of used notation.

A. Related Work

We discuss related work here, focusing on methods that are (also) meaningful for unlabeled structures.

A.1. Information Theory of Structure

Compared with the situation for sequences (see, e.g., [10]), the information theory of structured data is much less developed. The last decade has seen increasing efforts to change that. Sources and their entropies have been studied for binary trees [55, 83, 60, 36] and families of graphs [9, 59]. We are not aware of similar works specifically focusing on ordinal trees.

Some natural notions of structure sources contain more information (more degrees of freedom) that can possibly be extracted from a given object. In particular the *leaf-* and *depth-centric binary trees sources* of [83] as general classes of sources do *not* admit a universal code [83, Ex. 6 & Ex. 8] for that reason, making suitable restrictions necessary.

Other work has focused on notions of empirical entropies. Jansson et al. [52] study the degree entropy of ordinal trees, i.e., the zeroth-order entropy of sequence of node degrees $H^{\mathrm{deg}}(t)$, and show that $H^{\mathrm{deg}}(t)n$ bits are asymptotically necessary and sufficient to represent a tree of size n with given node degree frequencies. In [46], a notion of kth order empirical entropy is introduced for full binary trees, where the type of a node v (binary / leaf) depends on the direction (left/right) of the last k edges on the path from the root to v.

A.2. Tree Compression

The most widely studied methods for compressing trees are *DAG compression*, *top-tree compression*, and *grammar-based compression*. **DAG compression** is the oldest method. It stores identical shared fringe subtrees only once and hence transforms a tree t into a DAG. The smallest such DAG is unique and can be computed in linear time [13]. While good enough to yield universal binary-tree codes for fringe-dominated trees (cf. the "Representation Ratio Negligibility Property" in [83] and similar sufficient conditions [76]), it is easy to construct examples where DAG compression is exponentially worse than the other methods [56] because repeated patterns "inside" the tree are not exploited.

Top-tree compression [7] avoids this shortcoming by DAG compressing a *top tree* [2] of t instead of t itself. A top tree represents a hierarchy of clusters of the tree edges: leaves are individual edges, internal (binary) nodes are merging operations of child clusters. Top tree compression is presented for node-labeled ordinal trees, but can be applied to unlabeled trees, as well, and we formulate its properties for these here. Top trees of best possible worst-case size $O(n/\log n)$ and of height $O(\log n)$ can be computed in linear time [58, 14] from an ordinal tree t on n nodes. Furthermore, any top DAG (of arbitrary height) for an ordinal tree t of size n can be transformed with a constant multiplicative blow-up in linear time into a top DAG of height $O(\log n)$ for t [27].

We can write a tree t as a term (see also Section B), thus transforming it into a string.[8] Any DAG for t corresponds to a straight-line program (SLP) [54] for this string, but with the restriction

[8]For terms, it is natural to have node labels (functions in the term) imply a given degree (function arity); such trees are called *ranked*. When this is not the case, trees are called unranked. Working with ranked node labels does not preclude to study unlabeled trees; we can imagine nodes to be labeled with their degree for this purpose. Any tree code must necessarily store each node's degree, so this does not add additional information.

that every nonterminal produces (the term of) a fringe subtree of t. To allow better compression through exploiting repeated patterns inside the tree, one can either give up the correspondence of nonterminals to subtrees/tree patters or move to a more expressive grammar formalism.

The latter approach leads to (linear) **tree straight-line programs (TSLPs)** [56], which can be seen as a *multiple context-free grammar* [79, §2.8]: here, a rank-k nonterminal derives $k+1$ substrings separated by k gaps (instead of a single substring in context-free grammars). That gives us the flexibility to let nonterminals produce (the term of) a *context* c, a fringe subtree $t[v]$ with k holes, i.e., k nodes are removed together with their subtree from $t[v]$ to obtain c. Let r denote the maximal degree in t, then we can transform any TSLP into one with only rank-1 and rank-0 nonterminals with a blow-up of $O(r)$ in grammar size (the total size of all right-hand sides) [57]. Like for top-tree compression, a TSLP of size $O(n/\log n)$ and height $O(\log n)$ can be computed from an unlabeled ranked (constant maximal degree) tree of n nodes in linear time [24]; unlike for top-trees this result does not directly generalize to ordered trees with arbitrary degrees.

Unsurprisingly, TSLPs yield universal codes for all the classes of binary-tree sources for which the DAG-based code of [83] is universal [25] (the worst-case or average-case fringe-dominated sources); but they are also shown to be universal for the class of monotonic sources [25], which are not in general compressed optimally using DAGs, and achieve compression to the above mentioned kth-order empirical entropy for binary trees [46].

Unlike top DAGs, TSLPs cannot decompose trees “horizontally” (splitting the children of one node), which makes them less effective for trees of large degree. **Forest straight-line programs (FSLPs)** [30] add such an operation; they are shown to achieve the same compression up to constant factors as TSLPs for the first-child-next-sibling encoding of a tree and top DAGs (for unlabeled trees) [30]. (For labeled trees over an alphabet of size σ, it is shown in [30] that a top DAG can be transformed in $O(n)$ time into an equivalent FSLP with a constant multiplicative blow-up, whereas the transformation from an FSLP to a top DAG needs time $O(\sigma n)$ and a multiplicative blow-up of size $O(\sigma)$ is unavoidable.)

The other approach mentioned above – using unrestricted (string) **SLPs on a linearization** of a tree t – is investigated in [8]. They consider compressing the balanced-parenthesis (BP) encoding of an ordinal tree t on n nodes, and show that an SLP proportional in size to the smallest DAG can be computed from the DAG [8, Lem. 8.1].

A similar approach is taken in [26], focusing on ranked trees. It is shown there that an SLP for the depth-first degree sequence (DFDS) can be exponentially smaller than the smallest TSLP (but a TSLP with factor $O(h \cdot d)$, for h the height and d the maximal degree of t, can always be computed from an DFDS-SLP), and also exponentially smaller than the minimal SLP for the BP sequence of an ordinal tree. On the other hand, any TSLP (and hence DAG) can be transformed into an SLP for the DFDS with a factor $O(d)$ blowup, where d is the maximal degree in t. The latter can still be more desirable as many algorithmic problems are efficiently solvable for TSLP-compressed trees [56].

Other approaches include an LZ77-inspired methods for ranked trees [31]; it is not known to support operations on the compressed representation.

A.3. Succinct Trees

The survey of Raman and Rao [73] and Navarro’s book [64] give an overview of the various known succinct ordinal-tree data structures; cardinal trees and binary trees are covered also in [16, 12]. From a theoretical perspective, the tree-covering technique – initially suggested by Geary, Raman and Raman [34]; extended and simplified in [44, 16, 11] – might be seen as the most versatile representation [17].

A typical property of succinct data structures is that their space usage is determined only by the *size* of the input. For example, all of the standard tree representations use $2n + o(n)$ bits of space for *any* tree with n nodes. Notable exceptions are ultrasuccinct trees [52] that compresses ordinal trees (indeed, their DFDS) to the (zeroth-order) empirical node-degree entropy and otherwise employs the data structures designed for the depth-first *unary* degree sequeunce (DFUDS) representation. Gańczorz [29] recently extended this shape compression to labeled trees, in which the labels are also stored in compressed form, and Davoodi et al. [11] achieved space bounded by the empirical node-type entropy for binary trees. The latter two works are closest to ours in terms of their data structures; both are based on (variants) of tree covering.

"Four Russians and an American". Using a Huffman code for the lookup-level in a data structure is an arguably obvious idea, but to the last author's surprise, this trick does not seem to be part of the standard toolbox in the field. We refer to it as the "Four-Russians-One-American" trick. While explicitly mentioned in [64, §4.1.2] for higher-order-entropy-compressed bitvectors, a recent work on run-length compressed bitvectors [3] does not discuss four Russians and one American as an option, although it is competitive (asymptotically) with some of their results, e.g., [3, Thm. 4]. The survey [40] on compressed storage schemes for strings does not mention four Russians and one American as an option, although it yields the same time-space bounds as the (conceptually more complicated) methods discussed there (§3.2 and §3.3, based on [38] resp. [20]). Finally – closest to our work – compressing micro tree types in tree-covering data structures is used in several works [16, 11, 77, 29] – only Gańczorz [29] makes use of Four Russians and one American. Moreover, it does not seem to have been used before to compare against measures of compressibility other than (empirical) entropy.

A.4. Compressed Tree Data Structures

Some of the tree compression methods discussed above have also been turned into compressed data structures. Compressed tree data structures typically achieve $O(\log n)$ query times, which is in general close to optimal as discussed below. The exact set of supported operations for all discussed data structures is reported in Table 5, which also lists the main approaches for succinct data structures for comparison.

A DAG-compressed top tree of with d nodes can be augmented to a $O(d \log n)$ bit data structure [7, 45] for ordinal trees. Many more operations are supported by the data structure of [8], which uses the machinery developed in the same paper for providing random access to SLP-compressed strings to store an SLP for the BP string of an ordinal tree and simulate access to the excess sequence used in [68]. The data structure of [26] also uses a string SLP, but for the depth-first degree sequence instead of the BP, thus building on further indices for DFUDS-based succinct trees. In both cases, the size of the data structure becomes $O(g \log n)$ bits when g is the size of the SLP.

Lower bounds. Since all of the above methods are dictionary-based (in the sense of [53]), a recent lower bound [71] applies to them. It builds on earlier work for SLPs [78], which proved that if g is the size of an SLP G for a string T with $n = |T| = \Theta(g^{1+\varepsilon})$ for an $\varepsilon > 0$, random access to T requires $\Omega(\log n / \log \log n)$ time for any data structure using $O(g \operatorname{polylog}(n))$ space; ([78] has other

Table 5: Supported operations and their running time for different static-tree representations: balanced parentheses (`BP`), depth-first unary degree sequence (`DFUDS`), tree covering (`TC`), compression using top-DAGs (top directed acyclic graphs), forest straight-line programs (FSLP), and compression using straight-line programs for the `BP` sequence (SLP(BP)) resp. depth-first degree sequence (SLP(DFDS)). `BP` includes the range-min-max-tree based data structure of [68]; ultrasuccinct trees [52] are based on `DFUDS`; `TC` is used in [16, 11, 29, 77] and in the present work.

Operations	`BP`	`DFUDS`	`TC`	top DAG/FSLP	SLP(BP)	SLP(DFDS)
`parent`	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
`degree`	$O(1)$	$O(1)$	$O(1)$			$O(\log n)$
`first_child`, `next_sibling`	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
`last_child`	$O(1)$	$O(1)$	$O(1)$		$O(\log n)$	$O(\log n)$
`prev_sibling`	$O(1)$	$O(1)$	$O(1)$		$O(\log n)$	$O(\log n)$
`child`	$O(1)$	$O(1)$	$O(1)$			$O(\log n)$
`child_rank`	$O(1)$	$O(1)$	$O(1)$			$O(\log n)$
`depth`,	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	
`LCA`	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
`anc`	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	
`nbdesc`	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
`height`	$O(1)$		$O(1)$	$O(\log n)$	$O(\log n)$	
`leftmost_leaf`, `rightmost_leaf`	$O(1)$	$O(1)$	$O(1)$		$O(\log n)$	
`leaf_rank`, `leaf_select`	$O(1)$	$O(1)$	$O(1)$			
`level_leftmost`, `level_rightmost`	$O(1)$		$O(1)$		$O(\log n)$	
`level_pred`, `level_succ`	$O(1)$		$O(1)$		$O(\log n)$	
$\texttt{node\_rank}_{\texttt{PRE}}$, $\texttt{node\_select}_{\texttt{PRE}}$	$O(1)$	$O(1)$	$O(1)$		$O(\log n)$	
$\texttt{node\_rank}_{\texttt{IN}}$, $\texttt{node\_select}_{\texttt{IN}}$	$O(1)$		$O(1)$			
$\texttt{node\_rank}_{\texttt{POST}}$, $\texttt{node\_select}_{\texttt{POST}}$	$O(1)$		$O(1)$		$O(\log n)$	
$\texttt{node\_rank}_{\texttt{DFUDS}}$, $\texttt{node\_select}_{\texttt{DFUDS}}$		$O(1)$	$O(1)$			

tradeoffs for more compressible strings, too). Prezza [71] showed that also all operations required by tree data structures based on LOUDS, DFUDS or BP sequences require $\Omega(\log n/\log\log n)$ time on $O(\alpha\,\mathrm{polylog}(n))$-space data structures, where α is the size of any dictionary compressor (and $n = \Theta(\alpha^{1+\varepsilon})$).

Average-case behavior. While dictionary-based compression has the ability to dramatically compress some specific trees, simple information-theoretic arguments show that the vast majority are only slightly compressible. Clearly, this is true for uniformly chosen trees, but also for a vast variety of less balanced sources as those considered in this article. For such "average-case" trees, the compressed object (top dag, SLP) is of size $\alpha = O(n/\log n)$. While the above data structures then still use $O(n)$ bits of space, none is known to be succinct (have a constant of 2 in front of n).

Also, queries take $O(\log n)$ time, while the random-access lower bound no longer applies with $g = \Omega(n/\log n)$. Indeed, constant-time random access to SLPs is generally possible using $O(n^\varepsilon g^{1-\varepsilon}|\Sigma|\log n)$ bits of space [71] (setting $\tau = (n/g)^\varepsilon$), and that seems to be the best known bound. With $g = \Omega(n/\log n)$, that bound is $\Omega(|\Sigma| n \log^\varepsilon(n)) = \omega(n)$. It therefore seems not currently possible to build universally compressed data structures on top of any dictionary-based compressor that answers queries in constant time and has optimal space for the tree sources.

A.5. Range-Minimum Queries

Via the connection to lowest-common-ancestor (LCA) queries in Cartesian trees (see, e.g., [11]), we can formulate the RMQ problem as a task on trees: Any (succinct) data structure for binary trees that supports finding nodes by inorder index ($\texttt{node\_select}_{\texttt{IN}}$), `LCA`, and finding the inorder index of a node ($\texttt{node\_rank}_{\texttt{IN}}$) immediately implies a (succinct) solution for RMQ.

Worst-case optimal succinct data structures for the RMQ problem have been presented by Fischer and Heun [21], with subsequent simplifications by Ferrada and Navarro [18] and Baumstark et al. [5]. Implementations of (slight variants) of these solutions are part of widely-used programming libraries for succinct data structures, such as Succinct [1] and SDSL [35].

The above approaches use the same $2n + o(n)$ space on any input, but there are few attempts to exploit compressible instances. Fischer and Heun [21] show that range-minimum queries can still be answered efficiently when the array is compressed to kth order empirical entropy. For random permutations, the model we considered here, this does not result in significant savings. Barbay, Fischer and Navarro [4] used LRM-trees to obtain an RMQ data structure that adapts to presortedness in A, e.g., the number of (strict) runs by storing the tree as an ultrasuccinct tree. Again, for the random permutations considered here, this would not result in space reductions.

Recently, Gawrychowski et al. [32] designed RMQ solutions for grammar-compressed input arrays resp. DAG-compressed Cartesian trees. The amount of compression for random permutation is negligible for the former; for the latter it is less clear, but in both cases, they have to give up constant-time queries. The node-type entropy-compressed data structure for binary trees [11] is the first constant-time RMQ data structure that compresses random RMQ instances. They show that a node in the Cartesian tree has probability $\frac{1}{3}$ to be binary resp. a leaf, and probability $\frac{1}{6}$ to have a single left resp. right child. The resulting entropy is $\mathcal{H}(\frac{1}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{6}) \approx 1.91$ bit per node instead of the 2 bit for a trivial encoding.

Golin et al. [37] showed that $1.736n$ bits are (asymptotically) necessary and sufficient to encode a random RMQ instance, but they do not present a data structure that is able to make use of their encoding. The constant in the lower bound also appears in the entropy of BSTs build from random insertions [55], and indeed the shape distributions are the same [63, §3]. The encoding of Golin et al. has independently been described by Magner et al. [60] to compress trees (without attempts to combine it with efficient access to the stored object). Our result closes this gap between the lower bound and the best data structure with efficient queries, both for RMQ and for representing binary trees.

A.5.1. Applications

The RMQ problem is an elementary building block in many data structures. We discuss two exemplary applications here, in which a non-uniform distribution over the set of RMQ answers is to be expected.

Range searching. A direct application of RMQ data structures lies in 3-sided orthogonal 2D range searching. Given a set of points in the plane with coordinates (x, y), the goal is to report all points in x-range $[x_1, x_2]$ and y-range $(-\infty, y_1]$ for some $x_1, x_2, y_1 \in \mathbb{R}$. Given such a set of points in the plane, we maintain an array of the points sorted by x-coordinates and build a range-minimum data structure for the array of y-coordinates and a predecessor data structure for the set of x-coordinates. To report all points in x-range $[x_1, x_2]$ and y-range $(-\infty, y_1]$, we find the indices i and j of the outermost points enclosed in x-range, i.e., the ranks of (the successor of) x_1 resp. (the predecessor of) x_2. Then, the range-minimum in $[i, j]$ is the first candidate, and we compare its y-coordinate to y_1. If it is smaller than y_1, we report the point and recurse in both subranges; otherwise, we stop.

A natural testbed is to consider random point sets. When x- and y-coordinates are independent of each other, the ranking of the y-coordinates of points sorted by x form a random permutation, and we obtain the exact setting studied in this paper.

Longest-common extensions. A second application of RMQ data structures is the longest-common extension (LCE) problem on strings: Given a string T, the goal is to create a data structure that allows to answer LCE queries, i.e., given indices i and j, what is the largest length ℓ, so that $T_{i,i+\ell-1} = T_{j,j+\ell-1}$. LCE data structures are a building block, e.g., for finding tandem repeats in genomes; (see Gusfield's book [43] for many more applications).

A possible solution is to compute the suffix array $SA[1..n]$, its inverse SA^{-1}, and the longest common prefix array $LCP[1..n]$ for the string T, where $LCP[i]$ stores the length of the longest common prefix of the ith and $(i-1)$st suffixes of T in lexicographic order. Using an RMQ data structure on LCP, $lce(i, j)$ is found as $LCP\big[rmq_{LCP}(SA^{-1}(i) + 1, SA^{-1}(j))\big]$.

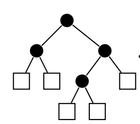
Since LCE effectively asks for lowest common ancestors of leaves in suffix trees, the tree shapes arising from this application are related to the shape of the suffix tree of T. This shape heavily depends on the considered input strings, but for strings generated by a Markov source, it is known that random suffix trees behave asymptotically similar to random tries constructed from independent strings of the same source [51, Chap. 8]. Those in turn have logarithmic height. This gives some hope that the RMQ instances arising from LCE are compressible; we could confirm this on example strings, but further study is needed here.

B. Preliminaries

In this section we introduce some basic definitions and notations; a comprehensive list of our notation is given in Section L. We write $[n..m] = \{n, \dots, m\}$ and $[n] = [1..n]$ for integers n, m. We use the standard Landau notation (i.e., O-notation etc.) and write lg for $\log_2$. We leave the basis of log undefined (but constant); (any occurrence of log outside a Landau-term should thus be considered a mistake). We make the convention that $0 \lg(0) = 0$ and $0 \lg(x/0) = 0$ for $x \geq 0$.

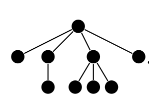
B.1. Trees

Let $\mathcal{T}$ denote the set of all *binary trees*, that is, of ordered rooted trees, such that each node has either (i) exactly two children, or (ii) a single left child, or (iii) a single right child, or (iv) is a leaf. For technical reasons, we also include the *empty tree* Λ (also called "null" in analogy of representing trees via left/right-child pointers), which consists of zero nodes, in the set of binary trees. A *fringe subtree* of a binary tree t is a subtree that consists of a node of t and all its descendants. With $t[v]$ we denote the fringe subtree rooted at node v and with $t_\ell[v]$ (resp. $t_r[v]$) we denote the fringe subtree rooted in v's left (resp. right) child: If v does not have a left (resp., right) child, then $t_\ell[v]$ (resp., $t_r[v]$) is the empty binary tree. If v is the root node of t, we shortly write t_ℓ and t_r instead of $t_\ell[v]$ and $t_r[v]$. With $|t|$ we denote the *size* (i.e., number of nodes) of t. Moreover, let $h(t)$ denote the *height* of t, which is inductively defined by $h(\Lambda) = 0$ and $h(t) = 1 + \max(h(t_\ell), h(t_r))$, for $t \neq \Lambda$. Let $\mathcal{T}_n$ denote the set of binary trees with n nodes and let $\mathcal{T}^h$ denote the set of binary trees of height h. We write trees inline as (unranked) terms with an anonymous function $\bullet$ representing a vertex; for example $t = \bullet(\bullet(\Lambda, \Lambda), \bullet(\bullet(\Lambda, \Lambda), \Lambda)) \in \mathcal{T}_4$ represents the binary tree

$t \;=$  .

(We followed the convention to draw empty subtrees as squares). A binary tree is called a *full binary tree*, if every node has either exactly two children or is a leaf, i.e., there are no unary nodes. Note that there is a natural one-to-one correspondence between the set $\mathcal{T}_n$ of binary trees of size n and the set of full binary trees with $n+1$ leaves. Every binary tree t of size n uniquely corresponds to a full binary tree t' with $n+1$ leaves by identifying the nodes of t with the internal nodes of t'. Thus, results from [25, 55, 76, 83] stated in the setting of full binary trees naturally transfer to our setting.

With $\mathfrak{T}$ we denote the set of *ordinal trees* (a.k.a. Catalan trees, planted plane trees); every node has a potentially empty sequence of children, each of which is a (nonempty) ordinal tree. Again, $\mathfrak{T}_n$ are ordinal trees with n nodes, $|t|$ denotes the size (number of nodes) of an ordinal tree $t \in \mathfrak{T}$, and $t[v]$ denotes the fringe subtree rooted in node v of $t \in \mathfrak{T}$. We use square brackets for writing ordinal trees (to distinguish from binary trees); for example $t = \bullet[\bullet[], \bullet[\bullet[]], \bullet[\bullet[], \bullet[], \bullet[]], \bullet[]] \in \mathfrak{T}_9$ stands for the ordinal tree

$t \;=$  .

Definition B.1 (BP encoding): *We define the balanced-parenthesis encoding of binary trees*

$BP : \mathcal{T} \to \{\mathtt{(},\mathtt{)}\}^\star$, *recursively as follows:*

$$BP(t) \quad = \quad \begin{cases} \varepsilon & \text{if } t = \Lambda \\ \mathtt{(} \cdot BP(t_\ell) \cdot \mathtt{)} \cdot BP(t_r) & \text{if } t = \bullet(t_l, t_r). \end{cases}$$

Similarly, we define for ordinal trees $BP_o : \mathfrak{T} \to \{\mathtt{(},\mathtt{)}\}^\star$ *recursively:*

$$BP_o(t) \quad = \quad \begin{cases} \varepsilon & \text{if } t = \Lambda \\ \mathtt{(} \cdot BP_o(t_1) \cdots BP_o(t_k) \cdot \mathtt{)} & \text{if } t = \bullet[t_1, \ldots, t_k],\ k \in \mathbb{N}_0. \end{cases}$$

Here ε denotes the empty sequence. For technical reasons, we also define *forests*, which are (possibly empty) sequences of trees from $\mathfrak{T}$: With $\mathfrak{F}$, we denote the set of all forests. We have $\mathfrak{F} = \mathfrak{T}^\star$. The balanced parenthesis mapping BP_o for ordinal trees naturally extends to a mapping $BP_o : \mathfrak{F} \to \{\mathtt{(},\mathtt{)}\}^\star$ by setting $BP_o(t_1 \cdots t_k) = BP_o(t_1) \cdots BP_o(t_k)$.

Definition B.2 (FCNS): *We define the first-child-next-sibling mapping* $\mathrm{fcns} : \mathfrak{F} \to \mathcal{T}$ *from ordinal forests to binary trees recursively as follows:*

$$\begin{aligned} \mathrm{fcns}(\varepsilon) &= \Lambda, \\ \mathrm{fcns}(t_1 = \bullet[c_1, \ldots, c_k], t_2, \ldots, t_j) &= \bullet(\mathrm{fcns}(c_1, \ldots, c_k), \mathrm{fcns}(t_2, \ldots, t_j)). \end{aligned}$$

Example B.3: Let $t = \bullet[\bullet[], \bullet[\bullet[]], \bullet[\bullet[], \bullet[], \bullet[]], \bullet[]]$. Then

$$\begin{aligned} \mathrm{fcns}(t) &= \bullet(\mathrm{fcns}(\bullet[], \bullet[\bullet[]], \bullet[\bullet[], \bullet[], \bullet[]], \bullet[]), \Lambda) \\ &= \bullet(\bullet(\Lambda, \mathrm{fcns}(\bullet[\bullet[]], \bullet[\bullet[], \bullet[], \bullet[]], \bullet[])), \Lambda) \\ &= \bullet(\bullet(\Lambda, \bullet(\bullet(\Lambda, \Lambda), \mathrm{fcns}(\bullet[\bullet[], \bullet[], \bullet[]], \bullet[]))), \Lambda) \\ &= \bullet(\bullet(\Lambda, \bullet(\bullet(\Lambda, \Lambda), \bullet(\mathrm{fcns}(\bullet[], \bullet[], \bullet[]), \mathrm{fcns}(\bullet[])))), \Lambda) \\ &= \bullet(\bullet(\Lambda, \bullet(\bullet(\Lambda, \Lambda), \bullet(\bullet(\Lambda, \bullet(\Lambda, \bullet(\Lambda, \Lambda))), \bullet(\Lambda, \Lambda)))), \Lambda). \end{aligned}$$

It is a folklore result that fcns is a bijection between ordinal forests and binary trees, which is easily seen by noting that:

$$\forall f \in \mathfrak{F} \ : \ \mathrm{fcns}(f) \ = \ BP^{-1}(BP_o(f)) \qquad \text{and} \qquad \forall t \in \mathcal{T} \ : \ \mathrm{fcns}^{-1}(t) \ = \ {BP_o}^{-1}(BP(t)).$$

An easy, uniquely decodable binary-tree code is obtained by storing the size plus one, $|t|+1$, of the binary tree in *Elias-gamma-code*, $\gamma(|t|+1)$, using $|\gamma(|t|+1)| = 2\lfloor \lg(|t|+1) \rfloor + 1$ many bits, followed by the balanced parenthesis encoding $BP(t)$ of the binary tree, using $2|t|$ many bits. (We store the size plus one of the binary tree, instead of its size, in order to take the case into account that t might be the empty binary tree). We can use this encoding to obtain a simple length-restricted version of any binary-tree code C as follows:

Definition B.4 (Worst-case bounding trick): *Let* $C : \mathcal{T} \to \{0,1\}^\star$ *denote a uniquely decodable encoding of binary trees. We define a simple* length-restricted version $\bar{C} : \mathcal{T} \to \{0,1\}^\star$ *of the binary-tree code* C *as follows:*

$$\bar{C}(t) \quad = \quad \begin{cases} \mathtt{0} \cdot \gamma(|t|+1) \cdot BP(t), & \text{if } |C(t)| > 2|t| + 2\lfloor \lg(|t|+1) \rfloor; \\ \mathtt{1} \cdot C(t), & \text{otherwise.} \end{cases}$$

The length-restricted code $\bar{C} : \mathcal{T} \to \{0,1\}^\star$ then uses

$$|\bar{C}(t)| \le \min\{|C(t)|, 2|t| + 2\lfloor \lg(|t|+1) \rfloor + 1\} + 1 \tag{2}$$

many bits in order to encode a binary tree t of size $|t|$, that is, by spending one extra bit to indicate the used encoding, we can get the best of both worlds. In a similar way, using the balanced parenthesis mapping $BP_o : \mathfrak{T} \to \{\texttt{(},\texttt{)}\}^\star$ for ordinal trees, we can obtain a *lenght-restricted version* of any ordinal-tree encoding.

B.2. Succinct Data Structures

We use the data structure of Raman, Raman, and Rao [72] for compressed bitvectors. They show the following result; we use it for more specialized data structures below.

Lemma B.5 (Compressed bit vector): *Let $\mathcal{B}$ be a bit vector of length n, containing m 1-bits. In the word-RAM model with word size $w = \Theta(\lg n)$ bits, there is a data structure of size*

$$\lg \binom{n}{m} \;+\; O\Bigl(\frac{n \log\log n}{\log n}\Bigr) \;\;\le\;\; m \lg\Bigl(\frac{n}{m}\Bigr) \;+\; O\Bigl(\frac{n \log\log n}{\log n} + m\Bigr)$$

bits that supports the following operations in $O(1)$ time, for any $i \in [1, n]$:

- *$access(\mathcal{B}, i)$: return the bit at index i in $\mathcal{V}$.*
- *$rank_\alpha(\mathcal{B}, i)$: return the number of bits with value $\alpha \in \{0,1\}$ in $\mathcal{B}[1..i]$.*
- *$select_\alpha(\mathcal{B}, i)$: return the index of the i-th bit with value $\alpha \in \{0,1\}$.*

Variable-cell arrays. A standard trick ("two-level index") allows us to store variable cell arrays: Let $o_1, \ldots, o_m$ be m objects where o_i needs s_i bits of space. The goal is to store an "array" O of the objects contiguously in memory, so that we can access the ith element in constant time as $O[i]$; in case $s_i > w$ (where w denotes the word size in the word-RAM model), we mean by "access" to find its starting position. We call such a data structure a variable-cell array.

Lemma B.6 (Variable-cell arrays): *There is a variable-cell array data structure for objects $o_1, \ldots, o_m$ of sizes $s_1, \ldots, s_m$ that occupies*

$$n \;+\; m \lg(\max s_i) \;+\; 2m \lg\lg n \;+\; O(m)$$

bits of space, where $n = \sum_{i=1}^m s_i$ is the total size of all objects.

Proof: Denote by $s = \min s_i$, $S = \max s_i$ and $\bar{s} = n/m$ the minimal, maximal and average size of the objects, respectively. We store the concatenated bit representation in a bitvector $B[1..n]$ and use a two-level index to find where the ith object begins. More in detail, we store the starting index of every bth object in an array *blockStart*$[1..\lceil m/b \rceil]$. The space usage is $\frac{m}{b} \lg n$ (ignoring ceilings around the logarithms). In a second array *blockLocalStart*$[1..m]$, we store for every object its starting index within its block. The space for this is $m \lg(bS)$ (again, ignoring ceilings around the logarithms): we have to prepare for the worst case of a block full of maximal objects.

It remains to choose the block size; $b = \lg^2 n$ yields the claimed bounds. Note that *blockStart* is $o(n)$ (for $b = \omega(\lg n/\bar{s})$), but *blockLocalStart* has, in general, non-negligible space overhead. The error term only comes from ignoring ceilings around the logarithms; its constant can be bounded explicitly. □

B.3. The Farzan-Munro Algorithm

We briefly recapitulate the Farzan-Munro algorithm [16, §3]. Recall that we have a parameter B governing the sizes of micro trees.

B.3.1. Ordinal Trees

The Farzan-Munro algorithm builds components bottom-up, through a recursive procedure which returns a component containing the root of the subtree it is called on, collecting nodes until a component contains at least B nodes: Let v be a node of the tree t and suppose that components for all children $u_1, \ldots, u_k$ of v have been computed recursively; the returned components will be called the *active* components $C_1, \ldots, C_k$ of the children, whereas some components might be already declared *permanent* and remain invariant. The normal mode of operations – "greedy packing" – is to start a new component C containing just v and to keep including the active components of v's children, left to right. If we reach $|C| \geq B$, C is declared permanent, and we start a new component $C \leftarrow \{v\}$. When all children are processed, we declare C permanent – except for the case when $|C| < B$ *and* it contains all children $u_1, \ldots, u_k$ of v. Finally, we return C.

This mode in isolation is not sufficient for our goal. An external edge of a component connects a *non-root node* of the component with the root of another component. Greedily packing leads to potentially many external edges per component. To achieve at most one external edge, the Farzan-Munro algorithm distinguishes heavy and light nodes; a node u is *heavy* if $|t[u]| \geq B$. The entire subtree of a light node fits into one component, so these do not have external edges and can be combined safely. For heavy children of v, there will be further connections, so we must avoid grouping several heavy children into one component to have at most one external edge per component. This leads to a problem since the active components of these nodes can be too small to remain ungrouped, and in general, there can be $\Theta(n)$ heavy nodes, so we cannot afford to keep that many components around.

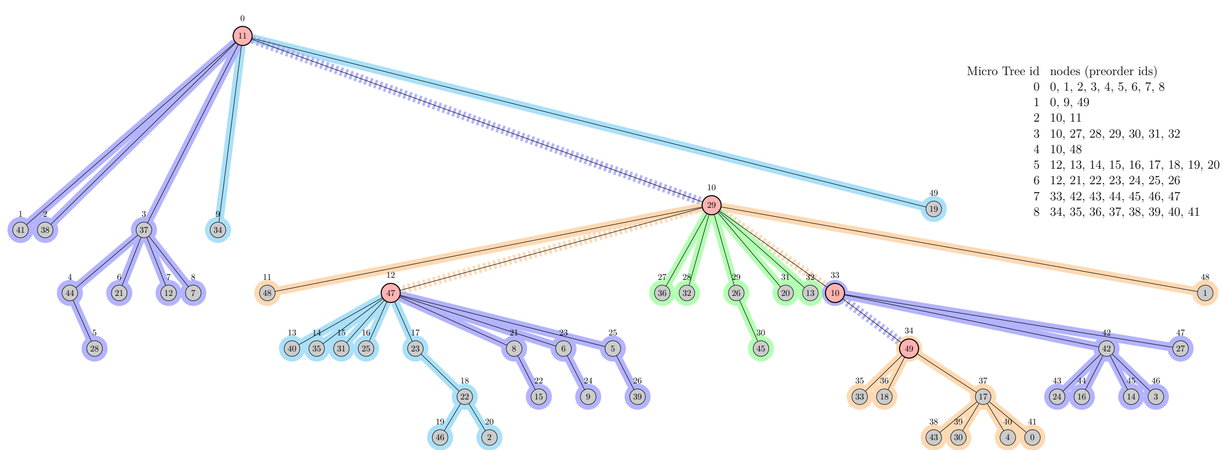


Figure 2: Example tree with $n = 50$ nodes, partitioned using $B = 8$. The root is a path node and the micro tree $\{0, 9, 49\}$ (preorder ids) shows a split of the children, omitting the heavy child 10. 10 itself is a branching node; note that here children in components are contiguous. (The leftmost and rightmost children here are *not* in the same micro tree despite the color).

However, the number of *branching nodes* – nodes with at least two heavy children – is always

$O(n/B)$. If v is a branching node, we can declare the active components of heavy children permanent and use greedy packing separately in the gaps between/outside heavy children. (This leads to some undersized components, but they can be charged to branching nodes, so remain bounded by $O(n/B)$ in number.)

The remaining, and only truly "abnormal" case, is that of a *path node*, happening when v has exactly one heavy child u_j. This makes two special treatments necessary. First, we cannot bound the number of path nodes, so we cannot afford to declare the active component of u_j permanent right away. But that is also not necessary, for there is only one heavy child anyways. So we just greedily pack as if all children were light. If, however, C_j was already declared permanent, we cannot add v to it without risking an *oversized* component – so C_j must stay untouched – but we also cannot pack the children left resp. right of u_j separately since that could lead to two external edges from v for the component that we pass up the tree. Therefore, we here – and only in this case – pack *across* the gap left by u_j, allowing a component that contains a range of v's children with one gap.

An example illustrating the special cases is shown in Figure 2.

B.3.2. Properties

From the procedure above, we immediately observe the following properties.

Fact B.7: Suppose we apply the Farzan-Munro algorithm with parameter B to a tree t with n nodes. For the resulting micro trees $\mu_1, \ldots, \mu_m$, we find:

(i) Every micro-tree root is heavy.

(ii) Every fringe micro tree has $\geq B$ nodes.

(iii) If v is a *heavy leaf* (v is heavy, but none of its children is), v is a micro-tree root (potentially shared among several components). All components with children of v contain *one interval* of children.

(iv) If v is a *branching node* (at least 2 heavy children), all components with children of v contain *one interval* of children.

(v) If v is a *path node* (exactly 1 heavy child), the components containing v also each contain *one interval* of children, except for the heavy child, which may be missing from the components of the surrounding interval.

(vi) Every node v appears in at least one micro tree; if v appears in several micro trees, then as the shared root of all of them.

B.3.3. Binary Trees

When applying the Farzan-Munro algorithm to binary trees, simplifications arise from the bounded degree of nodes; in particular, we never obtain components that share nodes. Figure 1 (page 9) shows an example.

It is illustrative to consider the possible cases that can arise. Let v be a node with children u_1 and u_2 (potentially null) whose active components are C_1 and C_2, respectively. If u_1 and u_2 are light $|C_1|, |C_2| < B$, greedy packing yields a single component $\{v\} \cup C_1 \cup C_2$. If both children are heavy – v is a branching node – we keep $C = \{v\}$ and declare C, C_1 and C_2 permanent.

If only one child, say u_1, is heavy, there are two cases depending on whether C_1 is permanent. If it is, we keep it and pass $\{v\} \cup C_2$ up the tree. If C_1 is not permanent, it must be small, $|C_1| < B$, and greedy packing yields a single component $\{v\} \cup C_1 \cup C_2$.

Part I.
Binary Trees

We now present our results on binary trees. We begin by describing our code and data structure (Section C), then define the various classes of sources, state properties, list concrete examples and state and prove universality of our hypersuccinct code for the classes of sources introduced (Section D – F). For an overview over the classes of sources sources and the concrete examples considered in our paper, see Table 2 and Table 3.

C. Hypersuccinct Binary Trees

Here, we describe our compressed tree code resp. data structure. Both are based on the Farzan-Munro algorithm [16] to decompose a tree into connected subtrees (so-called *micro trees*). It was originally designed for ordinal trees; we state its properties here when applied on binary trees. The results follow directly from the result proven in [16] and the fact that node degrees are at most two. For the reader's convenience, we describe the relevant details of the method in Section B.3.

Lemma C.1 (Binary tree decomposition, [16, Theorem 1]): *For any parameter $B \geq 1$, a binary tree with n nodes can be decomposed, in linear time, into $\Theta(n/B)$ pairwise disjoint subtrees (so-called micro trees) of $\leq 2B$ nodes each. Moreover, each of these micro trees has at most three connections to other micro trees:*

(i) *an edge from a parent micro tree to the root of the micro tree,*

(ii) *an edge to another micro tree in the left subtree of the micro tree root,*

(iii) *an edge to another micro tree in the right subtree of the micro tree root.*

(iv) *At least one of the edges to a child micro tree (if both of them exist) emanates from the root itself.*

In particular, contracting micro trees into single nodes yields again a binary tree. □

If a node v's parent u belongs to a different micro tree, u will have a "null pointer" within its micro tree, i.e., it loses its child there. To recover these connections between micro trees, we do not only need the information which micro tree is a child of which other micro tree, but also which null pointer inside a micro tree leads to the lost child. We refer to this null pointer as the *portal* of the (parent) micro tree (to the child micro tree).

An additional property that we need is stated in the following lemma; it follows directly from the construction of micro trees.

Lemma C.2 (Micro-tree roots are heavy): *Let v be the root of a micro tree constructed using the tree parameter B, respectively, any ancestor of a micro tree root. Then $|t[v]| \geq B$.* □

C.1. Hypersuccinct Code

Based on the above properties of this tree partitioning algorithm, we design a universal code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ for binary trees: Given a binary tree t of size n, we apply the Farzan-Munro algorithm with parameter $B = \lceil \frac{1}{8}\lg(n)\rceil$ to decompose the tree into micro trees $\mu_1, \ldots, \mu_m$, where $m = \Theta(n/\log n)$. The size of the micro trees $\mu_1, \ldots, \mu_m$ is thus upper-bounded by $\mu = \lceil \frac{1}{4}\lg(n)\rceil$. With Υ we denote the *top tier* of the tree t, which is obtained from t by contracting each micro tree μ_i into a single node (it forms a graph minor of t in the graph-theoretic sense). In particular, as each micro tree μ_i has at most 3 connections to other micro trees (a parent micro tree and (up to) two child micro trees, see Lemma C.1), Υ is again a binary tree, and the size of Υ equals the number m of micro trees. With $\Sigma_\mu \subseteq \bigcup_{s\le\mu} \mathcal{T}_s$ we denote the set of shapes of micro trees that occur in the tree t: We observe that because of the limited size of micro trees, there are fewer different possible shapes of binary trees than we have micro trees. The crucial idea of our hypersuccinct encoding is to treat each shape of a micro tree as a letter in the alphabet Σ_μ and to compute a Huffman code $C : \Sigma_\mu \to \{0,1\}^\star$ based on the frequency of occurrences of micro tree shapes in the sequence $\mu_1, \ldots, \mu_m \in \Sigma_\mu^m$: For our hypersuccinct code, we then use a *length-restricted* version $\bar{C} : \Sigma_\mu \to \{0,1\}^\star$ obtained from C using the simple cutoff technique from Definition B.4. Finally, for each micro tree, we have to encode which null pointers (external leaves) are portals to left and right child components (if they exist). For that, we store the portals' rank in the micro-tree-local in left-to-right order of the null pointers using $\lceil \lg(\mu+1)\rceil$ bits each. We can thus encode t as follows:

1. Store n and m in Elias gamma code,
2. followed by the balanced-parenthesis (BP) bitstring for Υ (see Definition B.1).
3. Next comes an encoding for $\bar{C}$; for simplicity, we simply list all possible codewords and their corresponding binary trees by storing the size (in Elias code) followed by their BP sequence.
4. Then, we list the length-restricted Huffman codes $\bar{C}(\mu_i)$ of all micro trees in DFS order (of Υ).
5. Finally, we store 2 $\lceil \lg(\mu+1)\rceil$-bit integers to encode the portal nulls for each micro tree, again in DFS order (of Υ).

Altogether, this yields our *hypersuccinct encoding* $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ for binary trees. Decoding is obviously possible by first recovering n, m, and Υ from the BP, then reading the Huffman code and finally replacing each node in Υ by its micro tree in a depth-first traversal, using the information about portals to identify nodes from components that are adjacent in Υ. With respect to the length of the hypersuccinct code, we find the following:

Lemma C.3 (Hypersuccinct binary tree code): *Let $t \in \mathcal{T}_n$ be a binary tree of n nodes, decomposed into micro trees $\mu_1, \ldots, \mu_m$ by the Farzan-Munro algorithm. Let C be an ordinary Huffman code for the string $\mu_1 \ldots \mu_m$. Then, the hypersuccinct code encodes t with a binary codeword of length*

$$|\mathsf{H}(t)| \;\le\; \sum_{i=1}^{m} |C(\mu_i)| + O\Big(n \frac{\log\log n}{\log n}\Big).$$

Proof: We first show that, among the five parts of the hypersuccinct binary-tree code for $t \in \mathcal{T}_n$, all but the second to last one contribute $O(n \log\log n/\log n)$ bits. Part 1 clearly needs $O(\log n)$ bits and Part 2 requires $2m = \Theta(n/\log n)$ bits. For Part 3, observe that

$$|\Sigma_\mu| \le \sum_{s \le \lceil \lg n/4 \rceil} 4^s < \frac{4}{3} \cdot 4^{\lg n/4+1} = \frac{16}{3}\sqrt{n}.$$

With the worst-case cutoff technique from Definition B.4, $\bar{C}(\mu_i) \le 2 + 2\lg(\mu+1) + 2\mu \le O(\mu)$, so we need asymptotically $O(\sqrt{n})$ entries / codewords in the table, each of size $O(\mu) = O(\log n)$, for an overall table size of $O(\sqrt{n}\log n)$. Part 5 uses $m \cdot 2\lceil \lg(\mu+1)\rceil = \Theta(n \cdot \frac{\log\log n}{\log n})$ bits of space. It remains to analyze Part 4. We note that by applying the worst-case pruning scheme of Definition B.4, we waste 1 bit per micro tree compared to a pure, non-restricted Huffman code. But these wasted bits amount to $m = O(n/\log n)$ bits in total, and so are again a lower-order term:

$$\begin{aligned}\sum_{i=1}^{m} \bar{C}(\mu_i) &= \sum_{i=1}^{m} \min\{|C(\mu_i)| + 1, 2|\mu_i| + 2\lfloor \lg|\mu_i| + 1\rfloor + 2\} \\ &\le \sum_{i=1}^{m}(|C(\mu_i)| + 1) = \sum_{i=1}^{m} |C(\mu_i)| + O(n/\log n),\end{aligned}$$

where the first equality comes from (2). This finishes the proof. □

C.2. Tree Covering Data Structures

What sets hypersuccinct code apart from other known codes is that it can be turned into a universally compressed tree data structure with constant-time queries. For that, we use a well-known property of tree covering that can be formalized as follows.

Theorem C.4 (Tree-covering index [16]): *Given a binary tree $t \in \mathcal{T}_n$, decomposed into micro trees $\mu_1, \ldots, \mu_m$ with tree covering. Assuming access to a data structure that maps i to $BP(\mu_i)$ in constant-time (for any $i \in [m]$), there is a data structure occupying $o(n)$ additional bits of space that supports all operations from Table 1 in constant-time.*

We will use this to turn our hypersuccinct code into a full-blown tree data structure; any results proven about the space of the former via Lemma C.3 can then be transferred to this data structure. To realize the mapping of micro tree ids to shapes, we will keep a variable-cell bitvector (Lemma B.6) storing $i \mapsto \bar{C}(\mu_i)$ where $\bar{C}(\mu_i)$ is the (length-restricted) Huffman code of $\mu_1, \ldots, \mu_m$. To get the balanced-parenthesis strings, we additionally store a lookup-table for $\bar{C}(\mu_i) \mapsto BP(\mu_i)$. The space for the former is $\sum_{i=1}^{m} |\bar{C}(\mu_i)| + O(n\log\log n/\log n)$ by Lemma B.6 and the latter is $O(\sqrt{n}\,\mathrm{polylog}\, n)$ because of the length restriction (see the proof of Lemma C.3).

D. Memoryless and Higher-Order Binary-Tree Sources

Let $t \in \mathcal{T}$ be a binary tree. We define the *type* of a node v as

$$\mathrm{type}(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf,} \\ 1 & \text{if } v \text{ has a single left child (and no right child),} \\ 2 & \text{if } v \text{ is a binary node,} \\ 3 & \text{if } v \text{ has a single right child (and no left child).} \end{cases}$$

For a node v of a binary tree t, we inductively define the *history* of v, $h(v) \in \{1,2,3\}^\star$, as follows: If v is the root node, we set $h(v) = \varepsilon$, (i.e., the empty string). If v is the child node of node w of t, we set $h(v) = h(w)\,\mathrm{type}(w)$, i.e., in order to obtain $h(v)$, we concatenate the types of v's ancestors. Note that $\mathrm{type}(v)$ is not part of the history of v. Moreover, we define the k-history of v, $h_k(v)$, as the length-k-suffix of $1^k h(v)$, i.e., if $|h(v)| \geq k$, $h_k(v)$ equals the last k characters of $h(v)$, and if $|h(v)| < k$, we pad this too short history with 1's in order to obtain a string $h_k(v)$ of length k.[9]

Let $k \geq 0$, let $z \in \{1,2,3\}^k$ and let $i \in \{0,1,2,3\}$. With n_z^t we denote the number of nodes of t with k-history z and with $n_{z,i}^t$ we denote the number of nodes of type i of t and k-history z . A *kth-order type process* $\tau = (\tau_z)_{z\in\{1,2,3\}^k}$ is a tuple of probability distributions $\tau_z : \{0,1,2,3\} \to [0,1]$. A kth order type process assigns a probability $\mathbb{P}[t]$ to a binary tree t by

$$\mathbb{P}[t] \;=\; \prod_{v\in t} \tau_{h_k(v)}(\mathrm{type}(v)) \;=\; \prod_{z\in\{1,2,3\}^k} \prod_{i=0}^{3} (\tau_z(i))^{n_{z,i}^t}. \tag{3}$$

If $k = 0$, we call such a kth-order type process a *memoryless binary-tree source*: in this case, the probability distribution on the node types is independent of the node's ancestors' node types. If $k > 0$, we call the kth-order type process a *higher-order binary-tree source.*

A kth-order type process randomly constructs a binary tree t as follows: In a top-down way, starting at the root node, we determine for each node v its type, where this decision depends on the k-history $h_k(v)$ of the node: The probability that a node v is of type i is given by $\tau_{h_k(v)}(i)$. If $i = 0$, then this node becomes a leaf and the process stops at this node. If $i = 1$, we attach a single left child to the node, if $i = 2$, we attach a left and a right child to the node, and if $i = 3$, we attach a single right child to the node. The process then continues at these child nodes. Note that this process might produce infinite trees with non-zero probability.

We define the following higher-order empirical entropy for binary trees:

Definition D.1 (Empirical type entropy): *Let $k \geq 0$ be an integer, and let $t \in \mathcal{T}$ be a binary tree. The (unnormalized)* kth-order type entropy $H_k^{\mathrm{type}}(t)$ *of t is defined as*

$$H_k^{\mathrm{type}}(t) \;=\; \sum_{z\in\{1,2,3\}^k} \sum_{i=0}^{3} n_{z,i}^t \lg\left(\frac{n_z^t}{n_{z,i}^t}\right).$$

The corresponding normalized tree entropy is obtained by dividing by the tree size. The zeroth order empirical type entropy is a slight variant of the degree entropy defined for ordinal trees by Jansson, Sadakane, and Sung [52] and occurs implicitly in [11].

We say that the kth-order type process $(\tau_z)_z$ is the *empirical kth-order type process of a tree* t, if $\tau_z(i) = \frac{n_{z,i}^t}{n_z^t}$ for all $z \in \{1,2,3\}^k$ and $i \in \{0,1,2,3\}$. In particular, if $(\tau_z)_z$ is the empirical kth-order type process of a binary tree $t \in \mathcal{T}$, we have

$$\lg\left(\frac{1}{\mathbb{P}[t]}\right) \;=\; \sum_{z\in\{1,2,3\}^k} \sum_{i=0}^{3} n_{z,i}^t \lg\left(\frac{1}{\tau_z(i)}\right) \;=\; \sum_{z\in\{1,2,3\}^k} \sum_{i=0}^{3} n_{z,i}^t \lg\left(\frac{n_z^t}{n_{z,i}^t}\right) \;=\; H_k^{\mathrm{type}}(t).$$

This shows that the empirical entropy is precisely the number of bits an optimal code can achieve for this source.

[9] This is an ad-hoc decision: Alternatively, we could allow histories of length smaller than k.

Example D.2 (Uniform binary trees): In order to encode a (uniformly random) binary tree of size n, $2n$ bits are necessary [16]. Let τ denote the memoryless type process defined by $\tau(0) = \tau(1) = \tau(2) = \tau(3) = \frac{1}{4}$, then for every binary tree t of size n we have $\mathbb{P}[t] = 4^{-n}$ and in particular, $\lg(1/\mathbb{P}[t]) = 2n$.

Example D.3 (Full binary trees): Probability distributions over full binary trees are obtained from type processes $(\tau_z)_{z\in\{1,2,3\}^k}$ with $\tau_z(1) = \tau_z(3) = 0$ for all $z \in \{1, 2, 3\}^k$. Recall that every full binary tree consists of an odd number $n = 2\nu + 1$ of nodes: ν binary nodes and $\nu + 1$ leaves for some integer ν. If τ is a memoryless type process, we thus have $\mathbb{P}[t] = \tau(0)^{\nu+1}\tau(2)^{\nu}$ for every $t \in \mathcal{T}_n$. Setting $\tau(0) = \tau(2) = \frac{1}{2}$ yields

$$\lg\left(\frac{1}{\mathbb{P}[t]}\right) \quad = \quad (\nu+1)\log(2) + \nu\log(2) \quad = \quad n,$$

and $n = 2\nu + 1$ is the minimum number of bits needed to represent a (uniformly chosen) full binary tree $t \in \mathcal{T}_n$ [46].

Example D.4 (Unary paths): Type processes $(\tau_z)_{z\in\{1,2,3\}^k}$ with $\tau_z(2) = 0$ yield probability distributions over unary-path trees, i.e., trees only consisting of unary nodes and one leaf. In order to encode a unary-path tree of size $n + 1$, we need n bits (to encode the n "directions" left/right). For a fixed integer n, let τ^n denote the memoryless type process with $\tau^n(1) = \tau^n(3) = 1/(2 + \varepsilon_n)$ and $\tau^n(0) = \varepsilon_n/(2 + \varepsilon_n)$, for $\varepsilon_n = 2/n$. We have

$$\begin{aligned}\lg\left(\frac{1}{\mathbb{P}[t]}\right) \quad &= \quad n\lg(2+\varepsilon_n) + \lg\left(1 + \frac{2}{\varepsilon_n}\right) \quad = \quad n\lg\left(2 + \frac{2}{n}\right) + \lg(n+1) \\ &\leq \quad n + \lg(n+1) + \frac{1}{\ln(2)},\end{aligned}$$

for every unary-path tree $t \in \mathcal{T}_{n+1}$.

Example D.5 (Motzkin trees): Motzkin trees are binary trees with only one type of unary nodes: Probability distributions over Motzkin trees can be modeled by type processes $(\tau_z)_{z\in\{1,2,3\}^k}$ with $\tau_z(3) = 0$ for every $z \in \{1, 2, 3\}^k$. For encoding (uniformly random) Motzkin trees of size n, asymptotically $\lg(3)n \approx 1.58496n$ bits are necessary [75, Theorem 6.16]: Let τ denote the memoryless type process with $\tau(0) = \tau(1) = \tau(2) = \frac{1}{3}$, then $\mathbb{P}(t) = 3^{-n}$ for every Motzkin tree of size n. In particular, we have $\lg(1/\mathbb{P}[t]) = \lg(3)n$.

Example D.6: Let $(\tau_z)_{z\in\{1,2,3\}^k}$ denote a higher-order type process with $\tau_{z'1}(1) = \tau_{z'3}(1) = \tau_{z'1}(3) = \tau_{z'3}(3) = 0$ for every $z' \in \{1, 2, 3\}^{k-1}$, then τ only generates binary trees with non-zero probability, in which children of unary nodes are either binary nodes or leaves, i.e., on each path from the root node to a leaf of the tree, we do not pass two consecutive unary nodes. For example, a first-order type process $(\tau_z)_{z\in\{1,2,3\}}$ which satisfies this property is obtained by setting $\tau_1(2) = \tau_1(0) = 1/2$, $\tau_3(2) = \tau_3(0) = 1/2$, $\tau_2(0) = \tau_2(1) = \tau_2(2) = \tau_2(3) = 1/4$.

Example D.7: The *random binary search tree model* assigns a probability to a binary tree of size n by setting

$$\mathbb{P}_{bst}(t) = \prod_{v \in t} \frac{1}{|t[v]|},$$

where the product ranges over all nodes v of t, see Example E.1 and Section 5.2 for more information. This distribution over binary trees arises for binary search trees (BST)s, when they are built by successive insertions from a uniformly random permutation. In [37], it was shown that the average numbers of node types in a random binary search tree t of size n satisfy $\sum_{t \in \mathcal{T}_n} \mathbb{P}_{bst}[t] n_0^t \sim \sum_{t \in \mathcal{T}_n} \mathbb{P}_{bst}[t] n_2^t \sim \frac{1}{3} n$ and $\sum_{t \in \mathcal{T}_n} \mathbb{P}_{bst}[t] n_1^t = \sum_{t \in \mathcal{T}_n} \mathbb{P}_{bst}[t] n_3^t \sim \frac{1}{6} n$. Thus, it seems natural to consider the memoryless type process given by $\tau(0) = \tau(2) = \frac{1}{3}$ and $\tau(1) = \tau(3) = \frac{1}{6}$: In [11], a data structure supporting RMQ in constant time using

$$\begin{aligned}
&\sum_{t \in \mathcal{T}_n} \mathbb{P}_{bst}(t) \lg\left(\frac{1}{\mathbb{P}_\tau(t)}\right) + o(n) \\
&= \sum_{t \in \mathcal{T}_n} \mathbb{P}_{bst}(t) \Big(n_0^t \lg(3) + n_2^t \lg(3) + n_1^t \lg(6) + n_3^t \lg(6) \Big) + o(n) \\
&= \frac{1}{3} \lg(3) n + \frac{1}{3} \lg(3) n + \frac{1}{6} \lg(6) n + \frac{1}{6} \lg(6) n \\
&\approx 1.919n + o(n)
\end{aligned}$$

many bits in expectation is introduced. However, to achieve the asymptotically optimal $\sum_{t \in \mathcal{T}_n} \mathbb{P}_{bst}(t) \lg(\frac{1}{\mathbb{P}_{bst}[t]}) + o(n) \approx 1.736n + o(n)$ bits on average (see Section 5.2), it is necessary to consider a different kind of binary-tree sources.

D.1. Universality of Memoryless and Higher-Order Sources

In order to show universality of the hypersuccinct code from Section C.1 with respect to memoryless and higher-order binary tree sources, we first derive a source-specific encoding (a so-called depth-first arithmetic code) with respect to the memoryless/higher-order source, against which we will then compare our hypersuccinct code. An overview of the strategy is given in Section 4.1.

The formula for $\mathbb{P}[t]$, Equation (3), suggests a route for an (essentially) optimal *source-specific* encoding of any binary tree t with $\mathbb{P}[t] > 0$ that, given a kth-order type process $(\tau_z)_z$, spends $\lg(1/\mathbb{P}[t])$ (plus lower-order terms) many bits in order to encode a binary tree $t \in \mathcal{T}$ with $\mathbb{P}[t] > 0$: Such an encoding may spend $\lg(1/\tau_z(i))$ many bits per node v of type i and of k-history z of t. (Note that as $\mathbb{P}[t] > 0$ by assumption, we have $\tau_{h_k(v)}(\text{type}(v)) > 0$ for every node v of t). Assuming that we "know" the kth-order type process $(\tau_z)_z$ – i.e., that it need not be stored as part of the encoding – we can use *arithmetic coding* [82] in order to encode the type of node v in that many bits. A simple (source-dependent) encoding D_τ, dependent on a given kth-order type process $(\tau_z)_z$, thus stores a tree t as follows: While traversing the tree in depth-first order, we always know the k-history of each node v we pass, and encode type(v) of each node v, using arithmetic coding: To encode type(v), we feed the arithmetic coder with the model that the next symbol is a number $i \in \{0,1,2,3\}$ with probability $\tau_z(i)$, where z is the k-history of v. We refer to this (source-dependent) code D_τ as the *depth-first arithmetic code* for the type process τ. We can reconstruct the tree t recursively from its code $D_\tau(t)$, as we always know the node types of nodes we have already visited in the depth-first order traversal of the tree, and the k-history of the node which we will visit next. As arithmetic coding needs $\lg\Big(1/\tau_{h_k(v)}(\mathrm{type}(v))\Big)$ many bits per node v, plus at most 2 bits of overhead, the total number of bits needed to store a binary tree $t \in \mathcal{T}$ is thus

$$|D_\tau(t)| \;\leq\; \sum_{v\in t} \lg\left(\frac{1}{\tau_{h_k(v)}(\mathrm{type}(v))}\right) + 2 \;=\; \lg\left(\frac{1}{\mathbb{P}[t]}\right) + 2. \tag{4}$$

Note that D_τ is a single prefix-free code for the set of all binary trees which satisfy $\mathbb{P}[t] > 0$ with respect to the kth-order type process $(\tau_z)_z$. We now start with the following lemma:

Lemma D.8: *Let $(\tau_z)_z$ be a kth-order type process and let $t \in \mathcal{T}$ be a binary tree of size n with $\mathbb{P}[t] > 0$. Then*

$$\sum_{i=1}^{m} |C(\mu_i)| \;\leq\; \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{nk}{\log n} + \frac{n \log\log n}{\log n}\right),$$

where C is a Huffman code for the sequence of the micro trees $\mu_1, \ldots, \mu_m$ obtained from the tree covering scheme (see Section C.1).

Proof: Let v be a node of t and let μ_i denote the micro tree of t that contains v. For the sake of clarity, let $\mathrm{type}^t(v)$ denote the type of v viewed as a node of t, and let $\mathrm{type}^{\mu_i}(v)$ denote the type of v in μ_i. We find that $\mathrm{type}^{\mu_i}(v) = \mathrm{type}^t(v)$, unless v is a parent of a portal null: In this case, the degree of v in μ_i is strictly smaller than the degree of v in t. By definition of the tree covering scheme (Lemma C.1), there are at most two parents of portal nulls per micro tree μ_i. If a tree μ_i contains two parents of portal nulls, one of those two nodes is the root node by Lemma C.1. Let $\pi_{i,1}$ denote the root node of μ_i and let $\pi_{i,2}$ denote the parent node of the portal null in μ_i which is not the root node, if it exists. Moreover, let $\mathrm{pos}(\pi_{i,2})$ denote the preorder index of node $\pi_{i,2}$ in μ_i.

Again for the sake of clarity, let $h_k^t(v)$ denote the k-history of v in t, and let $h_k^{\mu_i}(v)$ denote the k-history of v in micro tree μ_i. If v is of depth smaller than k (within μ_i), then its k-history $h_k^{\mu_i}(v)$ in μ_i might not coincide with its k-history $h_k^t(v)$ in t, and if v is a descendant of order smaller than k of node $\pi_{i,2}$ (i.e., v is of depth smaller than k in the subtree of μ_i rooted in $\pi_{i,2}$), then its k-history in μ_i does not coincide with its k-history in t, as $\pi_{i,2}$ changes its node type.

However, if we know the k-history $h_k^t(\pi_{i,1})$ of the root node $\pi_{i,1}$ of μ_i, the type $\mathrm{type}^t(\pi_{i,1})$, and the preorder position (in μ_i) and type (in t) of the node $\pi_{i,2}$, we are able to recover the k-history $h_k^t(v)$ of every node $v \in \mu_i$. We define the following modification of D_τ (i.e., the depth-first arithmetic code defined at the beginning of Section D.1), under the assumption that we know $h_k^t(\pi_{i,1})$, $\mathrm{type}^t(\pi_{i,1})$, $\mathrm{type}^t(\pi_{i,2})$ and $\mathrm{pos}(\pi_{i,2})$: While traversing the micro-tree μ_i in depth-first order, we encode $\mathrm{type}^{\mu_i}(v)$ (i.e., $\mathrm{type}^t(v)$) for every node v of μ_i except for nodes $\pi_{i,1}$ and $\pi_{i,2}$ (if it exists), for which we encode $\mathrm{type}^t(\pi_{i,1})$ and $\mathrm{type}^t(\pi_{i,2})$ (which we know, by assumption, as well as the preorder position of $\pi_{i,2}$); as we know $h_k^t(\pi_{i,1})$ by assumption, as well as the node types of $\pi_{i,1}$ and $\pi_{i,2}$, we know $h_k^t(v)$ at every node v we pass: we therefore encode $\mathrm{type}^t(v)$ using arithmetic coding by feeding the arithmetic coder with the model that the next symbol is a number

$i \in \{0,1,2,3\}$ with probability $\tau_{h_k^t(v)}(i)$. We denote this modification of $D_\tau(\mu_i)$ with $D_\tau^{h_k^t(\pi_{i,1})}(\mu_i)$ and find that it spends at most

$$D_\tau^{h_k^t(\pi_{i,1})}(\mu_i) \le \sum_{v\in\mu_i} \lg\left(\frac{1}{\tau_{h_k^t(v)}(\mathrm{type}^t(v))}\right) + 2 \tag{5}$$

many bits in order to encode a micro tree μ_i.

Furthermore, let $S : \{0,1,2,3\}^\star \to \{0,1\}^\star$ denote any uniquely decodable binary encoding which spends $2|z|$ bits in order to encode $z \in \{0,1,2,3\}^\star$. Let $\mathcal{I}_0$ denote the set of indices $i \in [m]$ for which μ_i is fringe, let $\mathcal{I}_1$ denote the set of indices $i \in [m]$ for which the root node of μ_i is a parent of a portal null, but no other portal null exists, let $\mathcal{I}_3$ denote the set of indices $i \in [m]$, for which the root node of μ_i is not a parent of a portal null, but node $\pi_{i,2}$ is a parent of a portal null, and let $\mathcal{I}_3 = [m] \setminus (\mathcal{I}_0 \cup \mathcal{I}_1 \cup \mathcal{I}_2)$. We define a modified encoding of μ_i as follows:

$$\tilde{D}_\tau(\mu_i) \quad = \quad \begin{cases} 00 \cdot S(h_k^t(\pi_{i,1})) \cdot D_\tau^{h_k^t(\pi_{i,1})}(\mu_i), & \text{if } i \in \mathcal{I}_0; \\ 01 \cdot S(h_k^t(\pi_{i,1})) \cdot \gamma(\mathrm{type}^t(\pi_{i,1})+1) \cdot D_\tau^{h_k^t(\mu_i)}(\mu_i), & \text{if } i \in \mathcal{I}_1; \\ 10 \cdot S(h_k^t(\pi_{i,1})) \cdot \gamma(\mathrm{pos}(\pi_{i,2})) \cdot \gamma(\mathrm{type}^t(\pi_{i,2}+1)) \\ \quad \cdot D_\tau^{h_k^t(\mu_i)}(\mu_i), & \text{if } i \in \mathcal{I}_1; \\ 11 \cdot S(h_k^t(\pi_{i,1})) \cdot \gamma(\mathrm{type}^t(\pi_{i,1}+1)) \cdot \gamma(\mathrm{pos}(\pi_{i,2})) \\ \quad \cdot \gamma(\mathrm{type}^t(\pi_{i,2})+1) \cdot D_\tau^{h_k^t(\pi_{i,1})}(\mu_i) & \text{if } i \in \mathcal{I}_3. \end{cases}$$

Note that formally, $\tilde{D}_\tau$ is *not* a prefix-free code over Σ_μ, as there can be micro tree shapes that are assigned several codewords by $\tilde{D}_\tau$, depending on which and how many nodes are portals to other micro trees. But $\tilde{D}_\tau$ is uniquely decodable to local shapes of micro trees, and can thus be seen as a *generalized prefix-free code*, where more than one codeword per symbol is allowed. In terms of encoding length, assigning more than one codeword is not helpful – removing all but the shortest one never makes the code worse – so a Huffman code minimizes the encoding length over the larger class of *generalized* prefix-free codes. Thus, as a Huffman code minimizes the encoding length over the class of *generalized* prefix-free codes, we find

$$\begin{aligned} \sum_{i=1}^m |C(\mu_i)| \quad &\le \quad \sum_{i=1}^m |\tilde{D}_\tau(\mu_i)| \quad = \quad \sum_{j=0}^3 \sum_{i\in\mathcal{I}_j} |\tilde{D}_\tau(\mu_i)| \\ &\le \quad \sum_{i=1}^m |S(h_k^t(\pi_{i,1}))| \; + \; \sum_{j=0}^3 \sum_{i\in\mathcal{I}_j} |D_\tau^{h_k^t(\pi_{i,1})}(\mu_i)| \; + \; O(m\log\mu), \end{aligned}$$

as $\mathrm{pos}(\pi_{i,2}) \le \mu$. With the estimate (5), this is upper-bounded by

$$\le \quad \sum_{i=1}^m |S(h_k^t(\pi_{i,1}))| \; + \; \sum_{j=0}^3 \sum_{i\in\mathcal{I}_j} \sum_{v\in\mu_i} \lg\left(\frac{1}{\tau_{h_k^t(v)}(\mathrm{type}^t(v))}\right) \; + \; O(m\log\mu).$$

Finally, as $|S(h_k^t(\pi_{i,1}))| = 2k$, we have

$$\begin{aligned}\sum_{i=1}^{m} |C(\mu_i)| &\leq \sum_{v\in t} \lg\left(\frac{1}{\tau_{h_k^t(v)}(\mathrm{type}^t(v))}\right) + O(m\log\mu) + O(km) \\ &= \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{nk}{\log n} + \frac{n\log\log n}{\log n}\right), \qquad \square\end{aligned}$$

as $m = \Theta(n/\log n)$ and $\mu = \Theta(\log n)$ (see Section C.1). This finishes the proof of the lemma.

From Lemma D.8 and Lemma C.3, we find that our hypersuccinct code is universal with respect to memoryless/higher-order type processes of order k, if $k = o(\log n)$:

Theorem D.9: *Let $(\tau_z)_z$ be a kth-order type process. The hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies*

$$|\mathsf{H}(t)| \leq \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{nk + n\log\log n}{\log n}\right)$$

for every $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$. In particular, if $(\tau_z)_z$ is the empirical kth-order type process of the binary tree t, we have

$$|\mathsf{H}(t)| \leq H_k^{\mathrm{type}}(t) + O\left(\frac{nk + n\log\log n}{\log n}\right).$$

From Theorem D.9 and Example D.2, Example D.3, Example D.4 and Example D.5 we obtain the following corollary:

Corollary D.10: *The hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ optimally compresses*

(i) ***binary trees*** *t of size n, drawn uniformly at random from the set of all binary trees of size n, using $|\mathsf{H}(t)| \leq 2n + O(n\log\log n/\log n)$ many bits,*

(ii) ***full binary trees*** *t of size n, drawn uniformly at random from the set of all full binary trees of size n, using $|\mathsf{H}(t)| \leq n + O(n\log\log n/\log n)$ many bits,*

(iii) ***unary-path trees*** *t of size $n+1$, drawn uniformly at random from the set of all unary-path trees of size $n+1$, using $|\mathsf{H}(t)| \leq n + O(n\log\log n/\log n)$ many bits, and*

(iv) ***Motzkin trees*** *t of size n, drawn uniformly at random from the set of all Motzkin trees of size n, using $|\mathsf{H}(t)| \leq \lg(3)n + O(n\log\log n/\log n)$ many bits.*

Remark D.11 (Shape entropy): Another notion of empirical entropy for unlabeled full binary trees was defined in [46]: The authors define the k-history of a node v of a full binary tree t as the string consisting of the last k directions (left/right) on the path from the root node of the tree to node v, and define the (normalized) kth order empirical entropy $\mathcal{H}_k^s(t)/|t|$ of the full binary tree as the expected uncertainty of the node types conditioned on the k-history of the node. In particular, it is then shown in [46], that the length of the binary encoding of full binary trees based on TSLPs from [25] can be upper-bounded in terms of this empirical entropy plus lower-order terms. As this notion of empirical entropy $\mathcal{H}_k(t)$ for full binary trees is conceptually quite similar to the empirical entropy of the node types $H_k^{\mathrm{type}}(t)$, the main ideas of our proof that $|\mathsf{H}(t)| \leq H_k^{\mathrm{type}}(t) + o(n)$ can be transferred to the setting from [46] in order to show that $|\mathsf{H}(t)| \leq \mathcal{H}_k(t) + o(n)$ holds for full binary trees of size n, as well, if $k = o(\log n)$. For a formal definition and further details on shape entropy, see Section K.

E. Fixed-Size and Fixed-Height Binary Tree Sources

A general concept to model probability distributions on various sets of binary trees was introduced by Zhang, Yang, and Kieffer in [83] (see also [55]), where the authors extend the classical notion of an information source on finite sequences to so-called structured binary-tree sources, or binary-tree sources for short: So-called leaf-centric binary-tree sources induce probability distributions on the set of full binary trees with n leaves and correspond to fixed-size binary-tree sources which we will introduce below, while so-called depth-centric binary tree souces induce probability distributions on the set of full binary trees of height h and correspond to fixed-height binary-tree sources, also to be introduced below in this section. For a formal introduction of structure sources and underlying concepts, see [83].

E.1. Fixed-Size Binary Tree Sources

A fixed-size binary tree source $\mathcal{S}_{fs}(p)$ is defined by a function $p : \mathbb{N}_0^2 \to [0, 1]$, such that

$$\sum_{\ell=0}^{n} p(\ell, n-\ell) \;=\; 1 \qquad \text{for all } n \in \mathbb{N}_0.$$

A fixed-size tree source $\mathcal{S}_{fs}(p)$ induces a probability distribution over the set of all binary trees of size n by

$$\mathbb{P}[t] \;=\; \prod_{v\in t} p(|t_\ell[v]|, |t_r[v]|), \tag{6}$$

where the product ranges over all nodes v of the binary tree t. If t is the empty tree, we set $\mathbb{P}[t] = 1$. Intuitively, this corresponds to generating a binary tree by a (recursive) depth-first traversal as follows: Given a target tree size n, ask the source for a left subtree size $\ell \in \{0, ..., n-1\}$: The probability of a left subtree size ℓ is $p(\ell, n-1-\ell)$. Create a node and recursively generate its left subtree of size ℓ and its right subtree of size $n-1-\ell$. The random choices in the left and right subtree are independent conditional on their sizes. An inductive proof over n verifies that $\sum_{t\in\mathcal{T}_n} \mathbb{P}[t] = 1$ for every $n \in \mathbb{N}_0$.

Note that the concept of fixed-size binary-tree sources is equivalent to the concept of leaf-centric binary-tree sources considered in [25, 55, 76, 83] in the setting of full binary trees.

Example E.1 (Random binary search tree model): The (arguably) simplest example of a fixed-size tree source is the *random binary search tree (BST) model* $\mathcal{S}_{fs}(p_{bst})$. This corresponds to setting $p_{bst}(\ell, n-\ell) = \frac{1}{n+1}$ for all $\ell \in \{0, \ldots, n\}$ and $n \in \mathbb{N}_0$. The very same distribution over binary trees arises for (unbalanced) binary search trees (BSTs), when they are build by successive insertions from a uniformly random permutation ("random BSTs"), and also for the shape of Cartesian trees build from a uniformly random permutation (a.k.a. random increasing binary trees [22, Ex. II.17 & Ex. III.33]); see Section 5.2.

Example E.2 (Uniform model): Perhaps the most elementary distribution on the set $\mathcal{T}_n$ is the *uniform probability distribution*, i.e., $\mathbb{P}(t) = \frac{1}{|\mathcal{T}_n|}$ for every $t \in \mathcal{T}_n$. This distribution corresponds to the fixed-size tree source $\mathcal{S}_{fs}(p_{uni})$ defined by

$$p_{uni}(\ell, n-\ell) \;=\; \frac{|\mathcal{T}_\ell||\mathcal{T}_{n-\ell}|}{|\mathcal{T}_{n+1}|} \qquad \text{for every } \ell \in \{0,\dots,n-1\} \text{ and } n \in \mathbb{N}_0.$$

Example E.3 (Binomial random tree model): Fix a constant $0 < \alpha < 1$. The *binomial random tree model* $\mathcal{S}_{fs}(p_{bin})$ is defined by

$$p_{bin}(\ell, n-\ell) \;=\; \alpha^\ell (1-\alpha)^{n-\ell} \binom{n}{\ell}$$

for every $\ell \in \{0,\dots,n\}$ and $n \in \mathbb{N}_0$. It is a slight variant of the digital search tree model, studied in [61] (see also [55, 83, 76]), and corresponds to (simple) *tries* built from n bitstrings generated by a Bernoulli(α) (memoryless) source.

Example E.4 (Almost paths): Setting $p(0,n) = p(n,0) = \frac{1}{2}$ for $n \geq 2$ yields a fixed-size source which produces unary paths; (this is a special case of [83, Ex. 6]). One can generalize the example so that $p(\ell, r) > 0$ implies $\min\{\ell, r\} \leq K$ for some constant K by setting

$$p_{path}(\ell, r) \;=\; \begin{cases} \min\left\{\dfrac{1}{\ell+r+1}, \dfrac{1}{2(K+1)}\right\} & \text{if } \ell \leq K \text{ or } r \leq K, \\ 0 & \text{otherwise.} \end{cases}$$

A fixed-size source $\mathcal{S}_{fs}(p_{path})$ only generates binary trees for which at each node, the left or right subtree has at most K nodes. Unary paths correspond to $K = 0$.

Example E.5 (Random fringe-balanced BSTs): Let $t \in \mathbb{N}_0$ be a parameter, and define

$$p_{bal}(k, n-k-1) \;=\; \begin{cases} \dbinom{k}{t}\dbinom{n-k-1}{t} \Big/ \dbinom{n}{2t+1} & \text{if } n \geq 2t+1, \\ \dfrac{1}{n} & \text{otherwise.} \end{cases}$$

This is the shape of a random $(2t+1)$-fringe-balanced BST; (see [81, §4.3] and the references therein for background on these trees).

E.2. Fixed-Height Binary-Tree Sources

A fixed-height binary tree source $\mathcal{S}_{fh}(p)$ is defined by a function $p : \mathbb{N}_0^2 \to [0,1]$, such that

$$\sum_{\substack{i,j \in \mathbb{N}_0 \\ \max(i,j)=h}} p(i,j) \;=\; 1 \qquad \text{for all } h \in \mathbb{N}_0.$$

A fixed-height tree source $\mathcal{S}_{fh}(p)$ induces a probability distribution over the set of all binary trees of height h by

$$\mathbb{P}[t] \quad = \quad \prod_{v \in t} p(h(t_\ell[v]), h(t_r[v])), \tag{7}$$

where the product ranges over all nodes of the binary tree t. If t is the empty tree, we set $\mathbb{P}[t] = 1$. Intuitively, this corresponds to generating a binary tree by a (recursive) depth-first traversal as follows: Given a target height h of the tree, ask the source for the height ℓ of the left subtree and the height r of the right subtree conditional on $\max(\ell, r) = h - 1$. The probability of a pair of heights (ℓ, r) with $\max(\ell, r) = h - 1$ is $p(\ell, r)$. Create a node and recursively generate its left subtree of height ℓ and its right subtree of height r. The random choices in the left and right subtree are independent conditional on their heights. An inductive proof over h verifies that $\sum_{t \in \mathcal{T}^h} \mathbb{P}[t] = 1$ for every $h \in \mathbb{N}_0$. Note that the concept of fixed-height binary-tree sources is equivalent to the concept of depth-centric binary-tree sources considered in [25, 55] in the setting of full binary trees.

Example E.6 (AVL trees by height): An AVL tree is a binary tree t, such that for every node v of t, we have $|h(t_\ell[v]) - h(t_r[v])| \le 1$. Let $\mathcal{T}^h(\mathcal{A})$ denote the set of AVL trees of height h. The number of AVL trees of height h satisfies the following recurrence relation:

$$|\mathcal{T}^h(\mathcal{A})| \quad = \quad 2|\mathcal{T}^{h-1}(\mathcal{A})||\mathcal{T}^{h-2}(\mathcal{A})| \; + \; |\mathcal{T}^{h-1}(\mathcal{A})||\mathcal{T}^{h-1}(\mathcal{A})|.$$

Set

$$p(j,k) \quad = \quad \begin{cases} \dfrac{|\mathcal{T}^j(\mathcal{A})||\mathcal{T}^k(\mathcal{A})|}{|\mathcal{T}^h(\mathcal{A})|} & \text{for } (j,k) \in \{(h-2,h-1),(h-1,h-1),(h-1,h-2)\} \\ 0 & \text{otherwise,} \end{cases}$$

for every $h \ge 2$. Then $\mathcal{S}_{fh}(p)$ corresponds to a uniform probability distribution on the set $\mathcal{T}^h(\mathcal{A})$ of AVL trees of *height* h for every $h \in \mathbb{N}$.

E.3. Entropy of Fixed-Size and Fixed-Height Sources

Given a fixed-size tree source $\mathcal{S}_{fs}(p)$ or fixed-height tree source $\mathcal{S}_{fh}(p)$, we write $H_n(\mathcal{S}_{fs}(p))$, respectively $H_h(\mathcal{S}_{fh}(p))$ for the entropy of the distribution it induces over the set of binary trees $\mathcal{T}_n$, respectively, $\mathcal{T}^h$: If $\mathcal{S}_{fs}(p)$ is a fixed-size tree source, we have

$$\begin{aligned} H_n(\mathcal{S}_{fs}(p)) \quad &= \quad \sum_{t \in \mathcal{T}_n} \mathbb{P}[t] \lg\!\left(\frac{1}{\mathbb{P}[t]}\right) \\ &= \quad \sum_{t \in \mathcal{T}_n} \left(\prod_{v \in t} p(|t_\ell[v]|, |t_r[v]|)\right) \cdot \sum_{v \in t} \lg\!\left(\frac{1}{p(|t_\ell[v]|, |t_r[v]|)}\right). \end{aligned}$$

Similarly, if $\mathcal{S}_{fh}(p)$ is a fixed-height tree source, we have

$$H_h(\mathcal{S}_{fh}(p)) \quad = \quad \sum_{t \in \mathcal{T}^h} \mathbb{P}[t] \lg\!\left(\frac{1}{\mathbb{P}[t]}\right)$$

$$= \sum_{t\in\mathcal{T}^h} \biggl(\prod_{v\in t} p(h(t_\ell[v]), h(t_r[v]))\biggr) \cdot \sum_{v\in t} \lg\biggl(\frac{1}{p(h(t_\ell[v]), h(t_r[v]))}\biggr).$$

(Recall our convention $0\lg(1/0) = 0$).

In [55], the growth of H_n was examined with respect to several types of fixed-size binary-tree sources, like the uniform model $\mathcal{S}_{fs}(p_{uni})$ from Example E.2 and the binomial random tree model $\mathcal{S}_{fs}(p_{bin})$ from Example E.3. In particular, for the random BST model $\mathcal{S}_{fs}(p_{bst})$ from Example E.1, it was shown in [55] that

$$H_n(\mathcal{S}_{fs}(p_{bst})) \quad\sim\quad 2n\sum_{i=2}^{\infty}\frac{\lg i}{(i+2)(i+1)} \quad\approx\quad 1.7363771n;$$

see Section 5.2 (page 13) for more discussion of this example.

In the following, we present several properties of fixed-size and fixed-height binary-tree sources, for which we will be able to derive universal codes.

E.4. Monotonic Tree Sources

The first property was introduced in [25], where it was shown that a certain binary encoding of binary trees based on tree straight-line programs yields universal codes with respect to fixed-size and fixed-height sources satisfying this property:

Definition E.7 (Monotonic source): *A fixed-size or fixed-height binary tree source is monotonic if $p(\ell, r) \ge p(\ell + 1, r)$ and $p(\ell, r) \ge p(\ell, r + 1)$ for all $\ell, r \in \mathbb{N}_0$.*

Clearly, the binary search tree model $\mathcal{S}_{fs}(p_{bst})$ from Example E.1 is a monotonic fixed-size tree source, and one can easily show that the uniform model $\mathcal{S}_{fs}(p_{uni})$ from Example E.2 is another one. Furthermore, the fixed-size source $\mathcal{S}_{fs}(p_{path})$ from Example E.4 is monotonic. In contrast, the binomial random tree model $\mathcal{S}_{fs}(p_{bin})$ from Example E.3 and the fringe-balanced BSTs (Example E.5) are not monotonic.

For monotonic tree sources, we find the following:

Lemma E.8 (Monotonicity implies submultiplicativity): *Let $t \in \mathcal{T}$, and let $\mu_1, \ldots, \mu_m$ be a partition of t into disjoint subtrees, in the sense that every node of t belongs to exactly one subtree μ_i. If p corresponds to a monotonic fixed-size or monotonic fixed-height tree source, then*

$$\mathbb{P}[t] \quad\le\quad \prod_{i=1}^{m}\mathbb{P}[\mu_i]$$

Proof: Let v be a node of t and let μ_i denote the subtree that v belongs to. As μ_i is a subtree of t, we find $|\mu_{i\ell}[v]| \le |t_\ell[v]|$, $|\mu_{ir}[v]| \le |t_r[v]|$, $h(\mu_{i\ell}[v]) \le h(t_\ell[v])$ and $h(\mu_{ir}[v]) \le h(t_r[v])$. From the definition of monotonicity, we thus have $p(|\mu_{i\ell}[v]|, |\mu_{ir}[v]|) \ge p(|t_\ell[v]|, |t_r[v]|)$, if p corresponds to a fixed-size source, respectively, $p(h(\mu_{i\ell}[v]), h(\mu_{ir}[v])) \ge p(h(t_\ell[v]), h(t_r[v]))$, if p corresponds to a fixed-height source. As every node of t belongs to exactly one subtree μ_i, we find for monotonic fixed-size sources p:

$$\mathbb{P}[t] \quad=\quad \prod_{v\in t} p\bigl(|t_\ell[v]|, |t_r[v]|\bigr) \quad\le\quad \prod_{i=1}^{m}\prod_{v\in\mu_i} p\bigl(|\mu_{i\ell}[v]|, |\mu_{ir}[v]|\bigr) \quad=\quad \prod_{i=1}^{m}\mathbb{P}[\mu_i].$$

For monotonic fixed-height sources, we similarly find

$$\mathbb{P}[t] \;=\; \prod_{v\in t} p(h(t_\ell[v]), h(t_r[v])) \;\leq\; \prod_{i=1}^{m}\prod_{v\in\mu_i} p(h(\mu_{i\ell}[v]), h(\mu_{ir}[v])) \;=\; \prod_{i=1}^{m}\mathbb{P}[\mu_i]. \qquad \square$$

Lemma E.8 depicts the crucial property of monotonic sources, based on which we will be able prove universality of our hypersuccinct encoding from Section C.1.

E.5. Fringe-Dominated Tree Sources

A second class of tree sources, for which we will be able to show universality of our encoding, is the following: Let $n_b(t)$ be the number of nodes v in t with $|t[v]| = b$ and let $n_{\geq b}(t)$ likewise be the number of nodes v in t with $|t[v]| \geq b$.

Definition E.9 (Average-case fringe-dominated): *We call a fixed-size binary tree source average-case B-fringe dominated for a function B with $B(n) = \Theta(\log(n))$, if*

$$\sum_{t\in\mathcal{T}_n} \mathbb{P}[t] n_{\geq B(n)}(t) \;=\; o\biggl(\frac{n}{\log(B(n))}\biggr).$$

Definition E.10 (Worst-case fringe-dominated): *We call a fixed-size or fixed-height binary tree source worst-case B-fringe dominated for a function B with $B(n) = \Theta(\log(n))$, if*

$$n_{\geq B(n)}(t) \;=\; o(n/(\log B(n)))$$

for every tree $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$.

Note that Definition E.10 treats fixed-size and fixed-height binary tree sources, but Definition E.9 only covers fixed-size binary tree sources (to avoid averaging over trees of different sizes). Moreover, a fixed-size tree source that is worst-case B-fringe-dominated is clearly average-case B-fringe-dominated as well.

Sufficient conditions for fixed-size sources to be average-case fringe-dominated are given in [76] in the context of DAG-compression of trees. The classes for which our hypersuccinct code from Section C.1 is universal happen to be exactly the classes for which the DAG-based compression provably yields best possible compression:

Definition E.11 (ψ-nondegenerate [76]): *Let $\psi : \mathbb{R} \to (0, 1]$ denote a monotonically decreasing function. A fixed-size tree source $\mathcal{S}_{fs}(p)$ is called* ψ-nondegenerate, *if $p(\ell, n-\ell) \leq \psi(n)$ for every $\ell \in \{0, \ldots, n\}$ and sufficiently large n.*

Definition E.12 (φ-weakly-weight-balanced [76]): *Let $\varphi : \mathbb{R} \to (0, 1]$ denote a monotonically decreasing function and let $c \geq 3$ denote a constant. A fixed-size tree source $\mathcal{S}_{fs}(p)$ is called* φ-weakly-weight-balanced, *if*

$$\sum_{\frac{n}{c}\leq\ell\leq n-\frac{n}{c}} p(\ell-1, n-\ell-1) \;\geq\; \varphi(n)$$

for every $n \in \mathbb{N}$.

The following two lemmas follow from results shown in [76] (note that in [76], the authors consider full binary trees with n leaves, whereas we consider (not necessarily full) binary trees with n nodes, so there is an off-by-one in the definition of the tree size n):

Lemma E.13 (ψ-nondegeneracy implies fringe dominance, [76, Lemma 4]): *Let $\mathcal{S}_{fs}(p)$ be a ψ-nondegenerate fixed-size tree source, then*

$$\sum_{t \in \mathcal{T}_n} \mathbb{P}[t] \cdot n_{\geq B(n)}(t) \quad \leq \quad O(n\psi(B(n))),$$

for every function B with $B(n) = \Theta(\log n)$.

Lemma E.14 (φ-balance implies fringe dominance, [76, Lemma 14]): *Let $\mathcal{S}_{fs}(p)$ be a φ-weakly-weight-balanced fixed-size tree source, then*

$$\sum_{t \in \mathcal{T}_n} \mathbb{P}[t] \cdot n_{\geq B(n)}(t) \quad \leq \quad O\biggl(\frac{cn}{\varphi(n)B(n)}\biggr),$$

for every function B with $B(n) = \Theta(\log n)$.

Thus, if a fixed-size tree source $\mathcal{S}_{fs}(p)$ is ψ-nondegenerate for a function ψ with $\psi(n) \in o(1/\log(n))$, or φ-weakly-weight-balanced for a function φ with $\varphi(n) \in \omega(\log\log n/\log n)$ (under the assumption that $B = \Theta(\log n)$), then it is average-case fringe dominated. For the binary search tree model $\mathcal{S}_{fs}(p_{bst})$ (Example E.1), Lemma E.13 and Lemma E.14 both yield $\sum_{t\in\mathcal{T}_n} \mathbb{P}[t] n_{\geq B(n)}(t) \in O(n/B(n))$, by choosing $\psi(n) \in \Theta(1/n)$ and $\varphi(n) \in \Theta(1)$. Moreover, for the binomial random tree model $\mathcal{S}_{fs}(p_{bin})$ from Example E.3, we find $\sum_{t\in\mathcal{T}_n} \mathbb{P}[t] n_{\geq B(n)}(t) \in O(n/B(n))$ from Lemma E.14 (see also [76, Ex. 16]). Additionally, for random fringe-balanced BSTs from Example E.5, it is easy to show that $\sum_{t\in\mathcal{T}_n} \mathbb{P}[t] n_{\geq B(n)}(t) \in O(n/B(n))$ by choosing $\psi(n) = \Theta(1/n)$ in Lemma E.13 (see also [80, Lemma 2.38]).

Intuitively, φ-weakly-weight-balanced fixed-size tree sources lower-bound the probability of balanced binary trees in terms of the function φ. They generalize a class of tree sources considered in [25, Lemma 4 and Theorem 2], as well as so-called leaf-balanced (called weight-balanced below) tree sources introduced in [83] and further analyzed in [25]:

Definition E.15 (Weight-balanced): *A weight-balanced tree source is a φ-weakly-weight-balanced tree source with $\varphi = 1$, that is, there is a constant $c \geq 3$, such that*

$$\sum_{\frac{n}{c} \leq \ell \leq n - \frac{n}{c}} p(\ell - 1, n - \ell - 1) \quad = \quad 1$$

for every $n \in \mathbb{N}$.

Weight-balanced tree sources constitute an example of fixed-size tree sources which are worst-case fringe-dominated:

Lemma E.16 (Weight-balance implies fringe dominance): *Let $\mathcal{S}_{fs}(p)$ be a weight-balanced fixed-size tree source. Then*

$$n_{\geq B(n)}(t) \quad = \quad O\biggl(\frac{n}{B(n)}\biggr)$$

for every tree $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$ and function B, i.e., $\mathcal{S}_{fs}(p)$ is worst-case B-fringe dominated.

Proof: Lemma E.16 follows from results shown in [24] (see also [25, Lemma 3]): Let $0 < \beta \le 1$. In [24], the authors introduce so-called β-balanced binary trees: A node v of a binary tree t is called β-balanced, if $|t_\ell[v]| + 1 \ge \beta(|t_r[v]| + 1)$ and $|t_r[v]| + 1 \ge \beta(|t_\ell[v]| + 1)$ (note that in [24], the authors count leaves of full binary trees, such that there is an off-by-one in the definition of β-balanced nodes). A binary tree is called β-balanced, if for all internal nodes u, v of t such that u is the parent node of v, we have that u is β-balanced or v is β-balanced. In the proof of [24, Lemma 10], it is shown in the context of DAG-compression of trees that for every β-balanced tree $t \in \mathcal{T}_n$, we have $n_{\ge b}(t) \le 4\alpha n/b$ for every constant $b \in \mathbb{N}$, where $\alpha = 1 + \log_{1+\beta}(\beta^{-1})$. Now let $\mathcal{S}_{fs}(p)$ be a weight-balanced fixed-size tree source and let t be a binary tree with $\mathbb{P}[t] > 0$. It remains to show that t is β-balanced for some constant β: Let v be a node of t. As $\mathbb{P}[t] > 0$, we find that $p(|t_\ell[v]|, |t_r[v]|) > 0$, and thus, there is a constant c, such that $n/c \le |t_\ell[v]|+1, |t_r[v]|+1 \le n - n/c$: In particular, we find that $|t_\ell[v]| + 1 \ge (|t_r[v]| + 1)/c$ and $|t_r[v]| + 1 \ge (|t_\ell[v]| + 1)/c$. Thus, t is β-balanced with $\beta = 1/c$. $\square$

Finally, we will present a class of fixed-height binary tree sources that generalizes AVL-trees and is worst-case B-fringe dominated (and thus amenable to compression using our techniques).

Definition E.17 (δ-height-balanced): *A fixed-height tree source $\mathcal{S}_{fh}(p)$ is called* δ-height-balanced*, if there is a monotonically increasing function $\delta : \mathbb{N} \to \mathbb{N}_0$, such that for all $(i, j) \in \mathbb{N}_0 \times \mathbb{N}_0$ with $p(i, j) > 0$ and $\max(i, j) = k - 1$ we have $|i - j| \le \delta(k)$.*

For δ-height-balanced tree sources, we find the following:

Lemma E.18 (Height balance implies fringe dominance): *Let $\mathcal{S}_{fh}(p)$ be a δ-height-balanced fixed-height tree source, then*

$$n_{\ge B(n)}(t) \quad \le \quad O\left(\frac{\delta(n) n \log B(n)}{B(n)}\right)$$

for every tree $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$ and function B.

In particular, under the assumption that $B(n) = \Theta(\log n)$, $\mathcal{S}_{fh}(p)$ is worst-case fringe-dominated if $\delta(k) \in o(\log k/(\log\log k)^2)$. The class of δ-height-balanced fixed-height tree sources generalizes so-called depth-balanced tree sources introduced in [25]. The fixed-height binary tree source from Example E.6 is an example of a 1-height-balanced fixed-height tree source. Lemma E.18 follows from combining, respectively, generalizing known results from [25, Lemma 7] and [45, Lemma 2], the latter presented in the context of top-tree compression; in the following, we give a self-contained proof in our notation: We start with showing the following lemma based upon [45, Lemma 2], which is wider interest for establishing fringe dominance.

Lemma E.19 (Log-height implies fringe dominance): *Let t be a binary tree and let $b \in \mathbb{N}$. If there is a constant $c > 1$, such that $h(t[v]) \le \log_c(|t[v]| + 1) = \frac{1}{\lg(c)} \cdot \lg(|t[v]| + 1)$ for every node v of t, then the number $n_{\ge b}(t)$ of nodes v with $|t[v]| \ge b$ in t satisfies*

$$n_{\ge b}(t) \quad \le \quad \frac{4|t|(\lg b + 2)}{b \lg c} + \frac{2|t|}{b}.$$

Proof: We call a node v of t *heavy*, if $|t[v]| \ge b$, otherwise, we call the node v *light*. Furthermore, we call the empty binary tree *light*. Thus, our goal is to upper-bound the number of heavy nodes in t. The total number of heavy nodes consists of

(i) the number of heavy nodes with only light children plus

(ii) the number of heavy nodes with one heavy child and one light child (which might be the empty tree), plus

(iii) the number of heavy nodes with two heavy children.

We start with upper-bounding the number (i) of heavy nodes with only light children: These nodes are not in an ancestor-descendant relationship with each other, and as they are heavy, the subtrees rooted in those nodes are of size at least b: Thus, there are at most $|t|/b$ many of those nodes.

In order to upper-bound number (ii) of heavy nodes with one heavy child and one light child, we adapt the following definition from [45]: We say that a node v is in *class* i for an integer $i \in \mathbb{N}_0$, if $|t[v]| \in [2^i, 2^{i+1}-1]$. Moreover, we call a node a *top-class* i node, if its parent belongs to class $j > i$ and we say that a node is a *bottom-class* i node, if its children both belong to classes $i_1, i_2 < i$.

We find that if a node is heavy, then it is in class i for an integer $\lfloor \lg b \rfloor \leq i \leq \lfloor \lg |t| \rfloor$. Moreover, if a node v is in class i, then at most one of its children u, w is in class i as well: If both nodes u, w belonged to class i, then $|t[v]| = 1 + |t[u]| + |t[w]| \geq 1 + 2^i + 2^i > 2^{i+1}$, a contradiction to the fact that v belongs to class i.

Let v be a top-class i node. By the above considerations, there is exactly one path of class i nodes in $t[v]$, which leads from v to a bottom-class i node w, and there are no other class i nodes in $t[v]$. We upper-bound the length of this path from node v to node w as follows: By assumption, we find that $h(t[v]) \leq \lg(|t[v]| + 1)(\lg c)^{-1} \leq \lg(2^{i+1})(\lg c)^{-1} = (i+1)(\lg c)^{-1}$. Thus, $h(t[v]) - h(t[w]) \leq (i+1)(\lg c)^{-1}$. Hence, $t[v]$ contains at most $(i+1)(\lg c)^{-1}$ many class i nodes and in particular, $t[v]$ contains at most $(i+1)(\lg c)^{-1}$ many class i heavy nodes with one heavy child and one light child.

As top-class i nodes are not in an ancestor-descendant relationship with each other, there are at most $|t|/2^i$ many top-class i nodes in t. Thus, there are at most $|t|/2^i \cdot (i+1)(\lg c)^{-1}$ class i heavy nodes with one heavy child and one light child, respectively, only one heavy child, in t. Altogether, there are at most

$$\sum_{i=\lfloor \lg b \rfloor}^{\lfloor \lg |t| \rfloor} \frac{|t|(i+1)}{2^i(\lg c)} \quad \leq \quad \frac{4|t|(\lg b + 2)}{b \lg c}$$

many heavy nodes with one heavy child and one light child in t.

It remains to upper-bound number (iii) of heavy nodes with two heavy children: For this, note that all heavy nodes of t form a (non-fringe) subtree t' of t rooted in the root of t. Heavy nodes of type (i), i.e., heavy nodes with only light children, are the leaves of this subtree t', while nodes of type (ii) are unary nodes in t' and heavy nodes of type (iii) are binary nodes in t'. Thus, the number (iii) of heavy nodes with two heavy children is upper-bounded by the number (i), which is upper-bounded by $|t|/b$. This finishes the proof. □

With Lemma E.19, we are able to prove Lemma E.18:

Lemma E.18: Let $\beta \in \mathbb{N}$. We call a binary tree t *β-height-balanced*, if for every node v of t, we have $|h(t_\ell[v]) - h(t_r[v])| \leq \beta$. This property of trees was called β-depth-balanced trees in [25]. Note that every subtree of a β-height-balanced tree is β-height-balanced as well. In [25, Lemma 7], it is shown that for every β-height-balanced tree t, we have $|t|+1 \geq c^{h(t)}$ with $c = 1 + 1/(1+\beta)$ (note that in [25], the authors consider full binary trees and measure size as the number of leaves, such that there is an off-by-one in the meaning of $|t|$). Thus, Lemma E.19 applies to β-height-balanced trees.

Now let $\mathcal{S}_{fh}(p)$ be a fixed-height tree source, and let $\delta : \mathbb{N} \to \mathbb{N}_0$ be a monotonically increasing function, such that for all $(i,j) \in \mathbb{N}_0 \times \mathbb{N}_0$ with $p(i,j) > 0$ and $\max(i,j) = k-1$, we have $|i-j| \leq \delta(k)$. Moreover, let $t \in \mathcal{T}_n$ be a binary tree of size n with $\mathbb{P}[t] > 0$. Then $|h(t_\ell[v]) - h(t_r[v])| \leq \delta(h(t[v]))$ for every node v of t. In particular, as δ is monotonically increasing, we find that t is β-height balanced with $\beta = \delta(h(t))$ and as $h(t) \leq |t| = n$, t is $\delta(n)$-height-balanced. By Lemma E.19, we thus find that

$$n_{\geq B(n)}(t) \quad \leq \quad \frac{4n(\lg B(n) + 2)}{B(n)\lg c} + \frac{2n}{B(n)},$$

with $c = 1 + 1/(1+\delta(n))$. By the mean-value theorem, we find

$$\begin{aligned} \lg\Big(1 + \frac{1}{1+\delta(n)}\Big) \quad &= \quad \lg\Big(\frac{2+\delta(n)}{1+\delta(n)}\Big) \quad = \quad \lg(2+\delta(n)) - \lg(1+\delta(n)) \\ &\geq \quad \frac{1}{(2+\delta(n))\ln(2)}. \end{aligned}$$

Thus

$$n_{\geq B(n)}(t) \quad \leq \quad \frac{4\ln(2)(2+\delta(n))n(\lg B(n)+2)}{B(n)} + \frac{2n}{B(n)} \quad = \quad O\Big(\frac{\delta(n) n \log B(n)}{B(n)}\Big).$$

This proves the lemma. □

E.6. Universality of Fixed-Size and Fixed-Height Sources

In order to show universality of the hypersuccinct code from Section C.1 with respect to fixed-size and fixed-height sources, we proceed in a similar way as in the case of memoryless and higher-order sources: An overview of the strategy is given in Section 4.1. First, we derive a source-specific encoding (a so-called depth-first order arithmetic code) with respect to the fixed-size or fixed-height source, against which we will then compare our hypersuccinct code:

The formulas for $\mathbb{P}[t]$, Equation (6) and Equation (7), immediately suggest a route for an (essentially) optimal *source-specific* encoding of any binary tree t with $\mathbb{P}[t] > 0$ that, given a fixed-size or fixed-height source p, spends $\lg(1/\mathbb{P}[t])$ (plus lower-order terms) many bits in order to encode a binary tree $t \in \mathcal{T}$ with $\mathbb{P}[t] > 0$: For a given fixed-size source, such an encoding may spend $-\lg\big(p(|t_\ell[v]|, |t_r[v]|)\big)$ many bits per node v, while for a fixed-height source, it may spend $-\lg\big(p(h(t_\ell[v]), h(t_r[v]))\big)$ many bits per node v. (Note that as $\mathbb{P}[t] > 0$ by assumption, we have $p(|t_\ell[v]|, |t_r[v]|) > 0$, respectively, $p(h(t_\ell[v]), h(t_r[v])) > 0$ for every node v of t.) Assuming that we "know" p – i.e., assuming it is "hard-wired" into the code and need not be stored as part of the encoding – and assuming that we have already stored $|t[v]|$, if p corresponds to a fixed-size source, respectively, $h(t[v])$, if p corresponds to a fixed-height source, we can use *arithmetic coding* [82] to store $|t_\ell[v]|$ (from which we will then be able to determine $|t_r[v]|$), if p corresponds to a fixed-size source, respectively, $h(t_\ell[v])$ and $h(t_r[v])$, if p corresponds to a fixed-height source.

First, let us assume that p corresponds to a fixed-size binary tree source. A simple (source-dependent) encoding D_p thus stores a tree $t \in \mathcal{T}_n$ as follows: We initially encode the size of the tree in Elias gamma code: If the tree consists of n nodes, we store the Elias gamma code of $n+1$, $\gamma(n+1)$, in order to take the case into account that t is the empty binary tree. Additionally, while traversing the tree in depth-first order, we encode $|t_\ell[v]|$ for each node v, using arithmetic coding: To encode $|t_\ell[v]|$, we feed the arithmetic coder with the model that the next symbol is a number $\ell \in \{0, \ldots, |t[v]| - 1\}$ with respective probabilities $p(\ell, |t[v]| - 1 - \ell)$.

If p corresponds to a fixed-height binary tree source, we proceed similarly: A (source-dependent) encoding D_p with respect to a fixed-height source $\mathcal{S}_{fh}(p)$ stores a tree $t \in \mathcal{T}^h$ by initially encoding $h+1$, i.e., the height of the tree plus one, in Elias gamma code, $\gamma(h+1)$, followed by an encoding of $(h(t_\ell[v]), h(t_r[v]))$ for every node v in depth-first order, stored using arithmetic encoding: Note that there are $2h(t[v]) - 1$ many different possibilities for $(h(t_\ell[v]), h(t_r[v]))$, thus, we can represent a pair $(h(t_\ell[v]), h(t_r[v]))$ by a number $i \in \{0, 2h(t[v]) - 2\}$, (e.g., by letting i represent the pair $(i, h(t[v]) - 1)$ if $i \le h(t[v]) - 1$ and $(h(t[v]) - 1, 2h(t[v]) - 2 - i)$, otherwise). To encode $(h(t_\ell[v]), h(t_r[v]))$, we feed the arithmetic coder with the model that the next symbol is a number $i \in \{0, 2h(t[v]) - 2\}$ with respective probabilities $p(i, h(t[v]) - 1)$, if $i \le h(t[v]) - 1$, and $p(h(t[v]) - 1, 2h(t[v]) - 2 - i)$, otherwise.

We refer to this (source-dependent) code D_p as the *depth-first arithmetic code* for the binary tree source with probabilities p. We can reconstruct the tree t recursively from its code $D_p(t)$: Since we always know the subtree size, respectively, subtree height, we know how many and what size the bins for the next left subtree size, respectively, pair of subtree heights, uses in the arithmetic code. Finally, if a subtree size or height is 1 or 0, we know the subtree itself. Recalling that arithmetic coding compresses to the entropy of the given input plus at most 2 bits of overhead, we need at most $\lg(1/\mathbb{P}[t]) + 2$ bits to store t when we know $|t|$, respectively $h(t)$ (depending on the type of tree source). With $h(t) \le |t|$, and as the Elias-gamma code satisfies $|\gamma(n)| \le 2\lfloor \lg(n) \rfloor + 1$, we find that the total encoding length is upper-bounded by

$$|D_p(t)| \;\; \le \;\; \lg(1/\mathbb{P}[t]) \; + \; 2\lfloor \lg(|t|+1) \rfloor \; + \; 3. \tag{8}$$

If p corresponds to a fixed-size tree source, taking expectations over the tree t to encode, depth-first arithmetic coding thus stores a binary tree with n nodes using $H_n(\mathcal{S}_{fs}(p)) + O(\log n)$ bits on average.

E.6.1. Universality for Monotonic Fixed-Size and Fixed-Height Sources

In this subsection, we show universality of our hypersuccinct code from Section C.1 with respect to *monotonic* fixed-size and fixed-height sources, as defined in Definition E.7. We start with the following lemma:

Lemma E.20 (Monotonic bounds micro-tree code): *Let $\mathcal{S}_{fs}(p)$, respectively, $\mathcal{S}_{fh}(p)$, be a fixed-size or fixed-height tree source and let $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$. If $\mathcal{S}_{fs}(p)$, respectively, $\mathcal{S}_{fh}(p)$ is monotonic, then*

$$\sum_{i=1}^{m} |C(\mu_i)| \;\; \le \;\; \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{n \log\log n}{\log n}\right),$$

where C is a Huffman code for the sequence of micro trees $\mu_1, \ldots, \mu_m$ obtained from our tree covering scheme (see Section C.1).

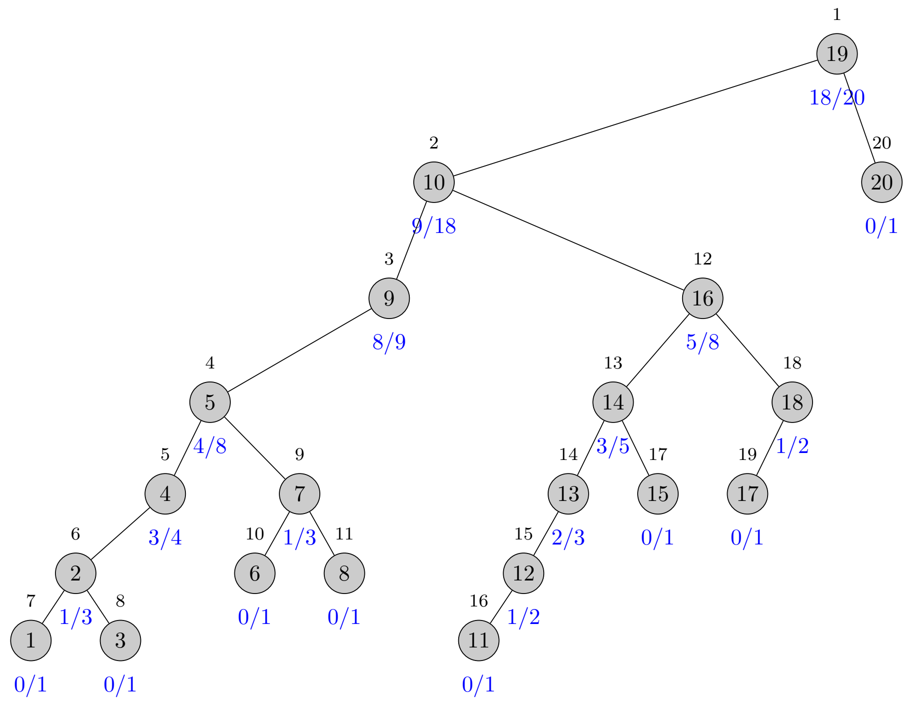


Figure 3: Example of a binary tree t with 20 nodes. Each node shows the inorder number (in the node), its preorder index (above the node) and the sizes of left subtree and its total subtree (blue, below the node). Assuming the random BST model, we have $\lg(1/\mathbb{P}[t]) \approx 28.74$, slightly below the expectation $H_{20}(\mathcal{S}_{fs}(p_{bst})) \approx 29.2209$. The arithmetic DFS code for the left tree sizes is 111011010111101011110101011111, i.e., 30 bits. This compares very favorably to a balanced-parenthesis representation ((((((()()))(()())))((((()))())(())))()) which would use 40 bits.

Proof: Let us denote by $D_p : \mathcal{T} \to \{0,1\}^\star$ the depth-first arithmetic code as introduced in the beginning of Section E.6. In particular, by Lemma E.8, we find that $\mathbb{P}[\mu_i] \geq \mathbb{P}[t]$ for all micro trees μ_i of t, and thus, $D_p(\mu_i)$ is well-defined for every micro tree μ_i. Restricting D_p to Σ_μ yields a prefix-free code for Σ_μ, so we know by the optimality of Huffman codes that

$$\sum_{i=1}^{m} |C(\mu_i)| \;\; \leq \;\; \sum_{i=1}^{m} |D_p(\mu_i)|.$$

By our estimate (8) for $|D_p|$, we find that

$$\begin{aligned}\sum_{i=1}^{m} |D_p(\mu_i)| \;\; &\leq \;\; \sum_{i=1}^{m} \biggl(\lg\biggl(\frac{1}{\mathbb{P}[\mu_i]}\biggr) + 3 + 2\lfloor \lg(|\mu_i|+1) \rfloor \biggr) \\ &\leq \;\; \sum_{i=1}^{m} \lg\biggl(\frac{1}{\mathbb{P}[\mu_i]}\biggr) + O(m \log \mu).\end{aligned}$$

Note that the subtrees $\mu_1, \ldots, \mu_m$ form a partition of t in the sense that every node of t belongs to exactly one subtree μ_i: Thus, and as p corresponds to a monotonic fixed-size or fixed-height source, we find by Lemma E.8:

$$\sum_{i=1}^{m} \lg\Big(\frac{1}{\mathbb{P}[\mu_i]}\Big) + O(m\log\mu) \quad\leq\quad \lg\Big(\frac{1}{\mathbb{P}[t]}\Big) + O(m\log\mu).$$

Altogether, with $m = \Theta(n/\log n)$ and $\mu = \Theta(\log n)$ (see Section C.1), we thus obtain

$$\sum_{i=1}^{m} |C(\mu_i)| \quad\leq\quad \lg\Big(\frac{1}{\mathbb{P}[t]}\Big) \;+\; O\Big(\frac{n\log\log n}{\log n}\Big). \qquad \square$$

From Lemma E.20 and Lemma C.3, we obtain the following result for monotonic tree sources (defined in Definition E.7):

Theorem E.21 (Universality for monotonic sources): *Let $\mathcal{S}_{fs}(p)$, respectively, $\mathcal{S}_{fh}(p)$, be a monotonic fixed-size or fixed-height tree source. Then the hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies*

$$|\mathsf{H}(t)| \quad\leq\quad \lg\Big(\frac{1}{\mathbb{P}[t]}\Big) + O\Big(\frac{n\log\log n}{\log n}\Big)$$

for every $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$.

The binary tree sources from Example E.1, Example E.2, and Example E.4 are monotonic fixed-size binary tree sources. Thus, together with Theorem E.21, we obtain the following corollary:

Corollary E.22: *The hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies the following:*

(i) A ***(random) binary search tree*** *(BST) (see Example E.1) t of size n is encoded using*

$$|\mathsf{H}(t)| \quad\leq\quad \lg(1/\mathbb{P}[t]) + O(n\log\log n/\log n)$$

many bits. In particular, we need on average

$$\begin{aligned}\sum_{t\in\mathcal{T}_n} \mathbb{P}[t]|\mathsf{H}(t)| \quad&\leq\quad H_n(\mathcal{S}_{fs}(p_{bst})) + O(n\log\log n/\log n)\\ &\approx\quad 1.736n + O(n\log\log n/\log n)\end{aligned}$$

many bits (see [55]) in order to encode a random BST of size n.

(ii) ***Almost-path binary trees*** *(for arbitrary $K \geq 0$) from Example E.4 are encoded using $|\mathsf{H}(t)| \leq \lg\big(\frac{1}{\mathbb{P}[t]}\big) + O(n\log\log n/\log n)$ many bits.*

As the uniform probability distribution on the set $\mathcal{T}_n$ of binary trees of size n can be modeled as a monotonic fixed-size binary tree source (see Example E.2), we find moreover that Corollary D.10, part (i) follows from Theorem E.21.

E.6.2. Universality for Fringe-Dominated Fixed-Size and Fixed-Height Sources

Recall that our hypersuccinct code from Section Section C.1 decomposes t into micro trees $\mu_1, \ldots, \mu_m$ using Lemma C.1 and uses a Huffman code C for $\mu_1, \ldots, \mu_m$. Some of these micro trees might be *"fringe"*, i.e., correspond to fringe subtrees of t and leaves in the top tier tree Υ, but many will be *internal* micro trees, i.e., have child micro trees in the top tier tree Υ. That means, micro-tree-local subtree sizes, resp. heights, and global subtree sizes, resp., heights, differ for nodes that are ancestors of the portal to the child micro tree – and only for those nodes do they differ: This will be the crucial observation in order to show that our hypersuccinct code is universal with respect to fringe-dominated sources.

Formally, let v be a node of t. If v is contained in a fringe micro tree μ_i, respectively, in a non-fringe micro tree μ_i but not an ancestor of a portal node, then $\mu_i[v] = t[v]$, and thus $p(|\mu_{i\ell}[v]|, |\mu_{ir}[v]|) = p(|t_\ell[v]|, |t_r[v]|)$, respectively, $p(h(\mu_{i\ell}[v]), h(\mu_{ir}[v])) = p(h(t_\ell[v]), h(t_r[v]))$. On the other hand, if v is an ancestor of a portal node in a non-fringe subtree μ_i, then $\mu_i[v] \neq t[v]$. In order to take this observation into consideration, we make the following definitions: Let μ_i be an internal (non-fringe) micro tree. By $\mathit{bough}(\mu_i)$, we denote the subtree of μ_i induced by the set of nodes that are ancestors of μ_i's child micro trees (ancestors of the portals); the boughs of a micro tree are the paths from the portals to the micro tree root. In particular, if v denotes a node of t contained in a subtree μ_i, then $t[v] \neq \mu_i[v]$ if and only if μ_i is not fringe and v is contained in $\mathit{bough}(\mu_i)$. Hanging off the boughs of μ_i are (fringe) subtrees $f_{i,1}, \ldots, f_{i,|\mathit{bough}(\mu_i)|+1}$, listed in depth-first order of the bough nulls these subtrees are attached to. In particular, some of these subtrees might be the empty tree. Recall that the portal nodes themselves are not part of μ_i and hence not part of $\mathit{bough}(\mu_i)$. We now find the following:

Lemma E.23 (bough decomposition): *Let $\mathcal{S}_{fs}(p)$, respectively, $\mathcal{S}_{fh}(p)$, be a fixed-size, respectively, fixed-height binary tree source. Furthermore, let $\mathcal{I}_0 = \{i \in [m] : \mu_i$ is a fringe micro tree in $t\}$ and let $\mathcal{I}_1 = [m] \setminus \mathcal{I}_0$. Then*

$$\sum_{i\in\mathcal{I}_0} \lg\left(\frac{1}{\mathbb{P}[\mu_i]}\right) \;+\; \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|\mathit{bough}(\mu_i)|+1} \lg\left(\frac{1}{\mathbb{P}[f_{i,j}]}\right) \;\leq\; \lg\left(\frac{1}{\mathbb{P}[t]}\right).$$

Proof: The statement follows immediately from the facts that (i) all the subtrees μ_i for $i \in \mathcal{I}_0$ and $f_{i,j}$ for $i \in \mathcal{I}_1$ and $j \in \{1, \ldots, |\mathit{bough}(\mu_i)| + 1\}$ are fringe subtrees of t, and (ii) every node v of t occurs in at most one of these fringe subtrees. Assume that p corresponds to a fixed-size tree source, then we find:

$$\begin{aligned}
&\sum_{i\in\mathcal{I}_0} \lg\left(\frac{1}{\mathbb{P}[\mu_i]}\right) + \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|\mathit{bough}(\mu_i)|+1} \lg\left(\frac{1}{\mathbb{P}[f_{i,j}]}\right) \\
&\quad= \; -\sum_{i\in\mathcal{I}_0}\sum_{v\in\mu_i} \lg(p(|\mu_{i\ell}[v]|, |\mu_{ir}[v]|)) - \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|\mathit{bough}(\mu_i)|+1} \sum_{v\in f_{i,j}} \lg(p(|f_{i,j\ell}[v]|, |f_{i,jr}[v]|)) \\
&\quad\overset{(i)}{=} \; -\sum_{i\in\mathcal{I}_0}\sum_{v\in\mu_i} \lg(p(|t_\ell[v]|, |t_r[v]|)) - \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|\mathit{bough}(\mu_i)|+1} \sum_{v\in f_{i,j}} \lg(p(|t_\ell[v]|, |t_r[v]|)) \\
&\quad\overset{(ii)}{\leq} \; -\sum_{v\in t} \lg(p(|t_\ell[v]|, |t_r[v]|)) \\
&\quad= \; \lg\left(\frac{1}{\mathbb{P}[t]}\right).
\end{aligned}$$

The proof for fixed-height sources is similar. $\square$

We now find the following:

Lemma E.24 (Great-branching lemma): *Let $\mathcal{S}_{fs}(p)$, respectively, $\mathcal{S}_{fh}(p)$, be a fixed-size, respectively, fixed-height tree source and let $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$. Then*

$$\sum_{i=1}^{m} |C(\mu_i)| \quad \le \quad \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O(n_{\ge B}(t) \log B),$$

where C is a Huffman code for the sequence of micro trees $\mu_1, \ldots, \mu_m$ from our tree covering scheme and $B = B(n) \in \Theta(\log n)$ is the parameter of the tree covering scheme (see Section C.1).

Proof: We construct a new encoding for micro trees against which we can compare the hypersuccinct code, the "great-branching" code, G_B, as follows:

$$G_B(\mu_i) \quad = \quad \begin{cases} \mathtt{0} \cdot D_p(\mu_i), & \text{if } \mu_i \text{ is a fringe micro tree;} \\ \mathtt{1} \cdot \gamma(|\mathit{bough}(\mu_i)|) \cdot \mathit{BP}(\mathit{bough}(\mu_i)) \cdot \\ \qquad D_p(f_{i,1}) \cdots D_p(f_{i,|\mathit{bough}(\mu_i)|+1}), & \text{otherwise,} \end{cases}$$

where $D_p : \mathcal{T} \to \{0,1\}^\star$ is the depth-first order arithmetic code as introduced in the beginning of Section E.6. Note that G_B is well-defined, as the encoding D_p is only applied to fringe subtrees μ_i and $f_{i,j}$ of t, for which $\mathbb{P}[\mu_i], \mathbb{P}[f_{i,j}] > 0$ follows from $\mathbb{P}[t] > 0$. Moreover, note that formally, G_B is *not* a prefix-free code over Σ_μ: there can be micro tree shapes that are assigned *several* codewords by G_B, depending on which nodes are portals to other micro trees (if any). But G_B is uniquely decodable to local shapes of micro trees, and can thus be seen as a *generalized prefix-free code*, where more than one codeword per symbol is allowed. In terms of the encoding length, assigning more than one codeword is not helpful – removing all but the shortest one never makes the code worse – so a Huffman code minimizes the encoding length over the larger class of *generalized* prefix-free codes. In particular, the Huffman code C for micro trees used in the hypersuccinct code achieves no worse encoding length than the great-branching code:

$$\sum_{i=1}^{m} |C(\mu_i)| \quad \le \quad \sum_{i=1}^{m} |G_B(\mu_i)|.$$

With $\mathcal{I}_0 = \{i \in [m] : \mu_i \text{ is a fringe micro tree in } t\}$, and $\mathcal{I}_1 = [m] \setminus \mathcal{I}_0$, we have

$$\begin{aligned} \sum_{i=1}^{m} |G_B(\mu_i)| \quad &= \quad \sum_{i \in \mathcal{I}_0} |G_B(\mu_i)| + \sum_{i \in \mathcal{I}_1} |G_B(\mu_i)| \\ &\le \quad \sum_{i \in \mathcal{I}_0} (1 + |D_p(\mu_i)|) \; + \; \sum_{i \in \mathcal{I}_1} \left(2 + 2\lg(|\mathit{bough}(\mu_i)|) + 2|\mathit{bough}(\mu_i)| + \sum_{j=1}^{|\mathit{bough}(\mu_i)|+1} |D_p(f_{i,j})| \right). \end{aligned}$$

With the estimate (8), this is upper-bounded by

$$\begin{aligned} &\sum_{i \in \mathcal{I}_0} \left(4 + \lg \frac{1}{\mathbb{P}[\mu_i]} + 2\lg(|\mu_i| + 1) \right) \; + \; \sum_{i \in \mathcal{I}_1} \sum_{j=1}^{|\mathit{bough}(\mu_i)|+1} \left(3 + \lg \frac{1}{\mathbb{P}[f_{i,j}]} + 2\lg(|f_{i,j}| + 1) \right) \\ &\qquad + \; \sum_{i \in \mathcal{I}_1} (2 + 2\lg(|\mathit{bough}(\mu_i)|) + 2|\mathit{bough}(\mu_i)|) \\ &\le \quad \sum_{i \in \mathcal{I}_0} \lg \frac{1}{\mathbb{P}[\mu_i]} \; + \; \sum_{i \in \mathcal{I}_1} \sum_{j=1}^{|\mathit{bough}(\mu_i)|+1} \lg \frac{1}{\mathbb{P}[f_{i,j}]} \; + \; O(m \log \mu) + O\left(\sum_{i \in \mathcal{I}_1} |\mathit{bough}(\mu_i)| \log \mu \right). \end{aligned}$$

By Lemma E.23, we have

$$\sum_{i\in\mathcal{I}_0} \lg\frac{1}{\mathbb{P}[\mu_i]} \;+\; \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|bough(\mu_i)|+1} \lg\frac{1}{\mathbb{P}[f_{i,j}]} \;\;\le\;\; \lg\!\left(\frac{1}{\mathbb{P}[t]}\right).$$

It remains to upper-bound the error terms: Lemma C.2 implies that any node v in the bough of a micro tree satisfies $|t[v]| \ge B$. Thus, the total number of nodes of t which belong to a bough of t is therefore upper-bounded by $n_{\ge B}(t)$. Altogether, we thus obtain

$$\sum_{i=1}^{m} |C(\mu_i)| \;\;\le\;\; \lg\!\left(\frac{1}{\mathbb{P}[t]}\right) \;+\; O(m\log\mu) \;+\; O(n_{\ge B}(t)\log\mu).$$

By a pigeon-hole argument, we find $n_{\ge B}(t) = \Omega(n/B)$. As $\mu = \Theta(B(n)) = \Theta(\log n)$ and $m = \Theta(n/B(n)) = \Theta(n/\log n)$ (see Section C.1), we have

$$\sum_{i=1}^{m} |C(\mu_i)| \;\;\le\;\; \lg\!\left(\frac{1}{\mathbb{P}[t]}\right) \;+\; O\Big(n_{\ge B(n)}(t)\log\log n\Big). \qquad \square$$

For average-case fringe-dominated fixed-size binary tree sources (defined in Definition E.9), we obtain the following result from Lemma E.24 and Lemma C.3:

Theorem E.25 (Universality from average-case fringe dominance): *Let $\mathcal{S}_{fs}(p)$ be an average-case fringe-dominated fixed-size binary tree source. Then the hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies*

$$\sum_{t\in\mathcal{T}_n} \mathbb{P}[t]|\mathsf{H}(t)| \;\;\le\;\; H_n(\mathcal{S}_{fs}(p)) + o(n).$$

For worst-case fringe-dominated fixed-size, respectively, fixed-height binary tree sources (defined in Definition E.10), we get the following result from Lemma E.24 and Lemma C.3:

Theorem E.26 (Universality from worst-case fringe dominance): *Let $\mathcal{S}_{fs}(p)$, respectively, $\mathcal{S}_{fh}(p)$ be a worst-case fringe-dominated fixed-size or fixed-height binary tree source. Then the hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies*

$$|\mathsf{H}(t)| \;\;\le\;\; \lg\!\left(\frac{1}{\mathbb{P}[t]}\right) + o(n)$$

for every binary tree $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$.

In Section E.5, we have presented several general classes of fixed-size and fixed-height tree sources, which are average-case or worst-case fringe-dominated. For these classes, we now obtain the following universality results of our hypersuccinct encoding from Lemma E.24 and Lemma C.3. With Lemma E.13 we find for ψ-nondegenerate fixed-size binary tree sources (defined in Definition E.11):

Corollary E.27 (Universality from ψ-nondegeneracy): *Let $\mathcal{S}_{fs}(p)$ be a ψ-nondegenerate fixed-size binary tree source. Then the hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies*

$$\sum_{t\in\mathcal{T}_n} \mathbb{P}[t]|\mathsf{H}(t)| \;\;\le\;\; H_n(\mathcal{S}_{fs}(p)) + O(n\psi(\log n)\log\log n).$$

With Lemma E.14, we obtain for φ-weakly-weight-balanced fixed-size binary tree sources (defined in Definition E.12):

Corollary E.28 (Universality from $\boldsymbol{\varphi}$-balance): *Let $\mathcal{S}_{fs}(p)$ be a φ-weakly-weight-balanced fixed-size binary tree source. Then the hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies*

$$\sum_{t \in \mathcal{T}_n} \mathbb{P}[t] |\mathsf{H}(t)| \quad \leq \quad H_n(\mathcal{S}_{fs}(p)) + O\left(\frac{n \log\log n}{\varphi(n) \log n}\right).$$

Moreover, with Lemma E.16, we find for weight-balanced fixed-size binary tree sources (defined in Definition E.15):

Corollary E.29 (Universality from weight-balance): *Let $\mathcal{S}_{fs}(p)$ be a weight-balanced fixed-size binary tree source. Then the hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies*

$$|\mathsf{H}(t)| \quad \leq \quad \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{n \log\log n}{\log n}\right)$$

for every binary tree $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$.

Finally, with Lemma E.18, we obtain for δ-height-balanced fixed-height binary tree sources (defined in Definition E.17):

Corollary E.30 (Universality from height-balance): *Let $\mathcal{S}_{fh}(p)$ be a δ-height-balanced fixed-height binary tree source. Then the hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies*

$$|\mathsf{H}(t)| \quad \leq \quad \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{\delta(n) n \log\log n}{\log n}\right)$$

for every binary tree $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$.

As the fixed-size and fixed-height tree sources from Example E.3, Example E.5, Example E.6 and Example F.4 are (average-case or worst-case) fringe dominated, we obtain the following corollary from Theorem E.25 and Theorem E.26:

Corollary E.31: *The hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies the following:*

(i) A binary tree of size n randomly generated by the ***binomial random tree model*** *$\mathcal{S}_{fs}(p_{bin})$ from Example E.3 is average-case optimally encoded:*

$$\sum_{t \in \mathcal{T}_n} \mathbb{P}[t] |\mathsf{H}(t)| \quad \leq \quad H_n(\mathcal{S}_{fs}(p_{bin})) + o(n).$$

(ii) A binary tree of size n randomly generated by the ***random fringe-balanced BST model*** *$\mathcal{S}_{fs}(p_{bal})$ from Example E.5 is average-case optimally encoded:*

$$\sum_{t \in \mathcal{T}_n} \mathbb{P}[t] |\mathsf{H}(t)| \quad \leq \quad H_n(\mathcal{S}_{fs}(p_{bal})) + o(n).$$

(iii) An ***AVL tree*** *t of size n and height h, drawn uniformly at random from the set $\mathcal{T}^h(\mathcal{A})$ of all AVL trees of height h, is optimally compressed using $|\mathsf{H}(t)| \leq \lg(|\mathcal{T}^h(\mathcal{A})|) + o(n)$ many bits (see Example E.6).*

*(iv) An α-**weight-balanced BST** of size n, drawn uniformly at random from the set $\mathcal{T}_n(\mathcal{W}_\alpha)$ of all α-weight-balanced binary trees of size n, is optimally compressed using $|\mathsf{H}(t)| \le \lg(|\mathcal{T}_n(\mathcal{W}_\alpha)|) + o(n)$ many bits (see Example F.4).*

We remark that using Lemma E.14 and Lemma E.13, it is possible to determine a more precise redundancy term for the results from Corollary E.31, part (i) and part (ii). Moreover, we remark that the average-case result from Corollary E.22, part (i), also follows from Theorem E.25.

F. Uniform-Subclass Sources

Finally, another class for which we will be able to prove universality of our code are so-called *uniform-subclass sources*. Let $\mathcal{T}(\mathcal{P})$ (resp. $\mathcal{T}_n(\mathcal{P})$) denote the subset of binary trees $t \in \mathcal{T}$ (resp. $t \in \mathcal{T}_n$), which satisfy a certain *property* $\mathcal{P}$ (examples will be given below). A uniform subclass source $\mathcal{U}_\mathcal{P}$ with respect to a property $\mathcal{P}$ assigns a probability to a binary tree $t \in \mathcal{T}_n$ by

$$\mathbb{P}[t] \;=\; \begin{cases} |\mathcal{T}_n(\mathcal{P})|^{-1} & \text{if } t \in \mathcal{T}_n(\mathcal{P}); \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

That is, a uniform subclass source $\mathcal{U}_\mathcal{P}$ induces a uniform probability distribution on the sets $(\mathcal{T}_n(\mathcal{P}))_n$ of all binary trees of size n which satisfy property $\mathcal{P}$. For technical reasons, we include the empty binary tree Λ in the set $\mathcal{T}(\mathcal{P})$ and set $\mathbb{P}[\Lambda] = 1$. We cannot hope to obtain universal codes for uniform-subclass sources in full generality. We therefore restrict our attention to *tame uniform subclass sources* $\mathcal{U}_\mathcal{P}$, which we define to mean the following four conditions:

(i) *Fringe-hereditary:* We call a property $\mathcal{P}$ *fringe-hereditary*, if every fringe subtree of a binary tree $t \in \mathcal{T}(\mathcal{P})$ belongs to $\mathcal{T}(\mathcal{P})$ as well. Furthermore, we call a uniform subclass source $\mathcal{U}_\mathcal{P}$ *fringe-hereditary*, if the property $\mathcal{P}$ is fringe-hereditary.

(ii) *Worst-case fringe dominated:* Recall that $n_{\ge b}(t)$ denotes the number of nodes v of a binary tree t, for which $|t[v]| \ge b$, where b is a parameter. We call a uniform subclass source $\mathcal{U}_\mathcal{P}$ *worst-case B-fringe-dominated* for a function $B = B(n)$ with $B(n) = \Theta(\log n)$, if $n_{\ge B(n)}(t) \in o(n/\log B(n))$ for every binary tree t in $\mathcal{T}_n(\mathcal{P})$.

(iii) *Log-linear*: A uniform subclass source $\mathcal{U}_\mathcal{P}$ is called *log-linear*, if there is a constant $c > 0$ and a function ϑ with $\vartheta(n) \in o(n)$, such that

$$\lg(|\mathcal{T}_n(\mathcal{P})|) \;=\; c \cdot n + \vartheta(n).$$

(iv) *Heavy twigged*: A property $\mathcal{P}$ is called *B-heavy twigged* for a function $B = B(n)$ with $B(n) \in \Theta(\log(n))$, if every t in $\mathcal{T}_n(\mathcal{P})$ satisfies the following condition: If v is a node of t with $|t[v]| \ge B = B(n)$, then both its subtrees satisfy $|t_\ell[v]|, |t_r[v]| \in \omega(1)$. A uniform subclass source $\mathcal{U}_\mathcal{P}$ is called *B-heavy twigged*, if $\mathcal{P}$ is B-heavy-twigged.

Definition F.1 (Tame uniform-subclass sources): *A uniform-subclass source $\mathcal{U}_\mathcal{P}$ is called tame, if it is fringe-hereditary, worst-case fringe dominated, log-linear, and heavy twigged.*

Example F.2 (AVL trees): An example of a property which satisfies all of these four conditions is being an AVL tree: An AVL tree is a binary tree t which is 1-height-balanced, that is, for every node v of t, we have $|h(t_\ell[v]) - h(t_r[v])| \le 1$. Let $\mathcal{A}$ denote this property of being an AVL tree, then $\mathcal{U}_\mathcal{A}$ yields the uniform probability distribution on the set of AVL trees of a given *size*. By definition, we find that $\mathcal{A}$ is fringe-hereditary. Moreover, from Lemma E.19 and [25, Lemma 7], we find that $\mathcal{U}_\mathcal{A}$ is worst-case fringe-dominated for any function B with $B(n) = \Theta(\log n)$.

A precise asymptotic for the number a_n of AVL trees of size n is reported by Odlyzko [70]: $a_n \sim \alpha^{-n} n^{-1} u(\ln n)$ as $n \to \infty$, where $\alpha = 0.5219024\ldots$ is a numerically known constant and $u(x)$ is a fixed, continuous periodic function. (Curiously, a detailed proof does not seem to have been published.) We obtain $\lg a_n \sim cn$ with $c \approx 0.938148$, that is, $\mathcal{U}_\mathcal{A}$ is log-linear.

Finally, $\mathcal{A}$ is heavy-twigged: Let v be a node of $t \in \mathcal{T}(\mathcal{A})$ with $|t[v]| \ge B$. As $t[v]$ is a binary tree, we have $h(t[v]) \ge \lg(B)$. Moreover, as t is an AVL tree, we have $h(t_\ell[v]), h(t_r[v]) \ge \lg(B) - 2$ and thus $|t_\ell(v)|, |t_r[v]| \ge \lg(B) - 2$, which is in $\omega(1)$ for $B = \Theta(\log(n))$.

Example F.3 (Red-black trees): Another example is the property $\mathcal{R}$, which holds if t is the shape of a red-black tree: A (left-leaning) red-black tree is a binary tree in which the edges are (implicitly) colored red and black, so that the following conditions hold:

(a) The number of black edges on any root-to-leaf path is the same.

(b) No root-to-leaf path contains two consecutive red edges.

(c) If a node has only one red child edge, it must be the left child edge.

It is easy to check that $\mathcal{R}$ is fringe-hereditary. One can show inductively that the height of a red-black tree is at most $2\lg n + O(1)$, which together with fringe-hereditary and Lemma E.19 implies that $\mathcal{U}_\mathcal{R}$ is worst-case fringe-dominated.

For the log-linearity, we have to determine $\lg r_n$, for r_n the number of left-leaning red-black trees of size n. Since left-leaning red-black trees are in bijection with 2-3-4-trees [74], we can also count the latter. The similar 2-3 trees are enumerated (where the size is the number of external leaves) in [62, 70] and the same technique allows to determine the exponential growth rate. We obtain $\lg r_n \sim cn$ with $c \approx 0.879146$.

For the heavy-twigged property, let $t[v]$ be a fringe subtree in a red-black tree with $|t[v]| \ge B$. We have $h(t[v]) \ge \lg(B)$ (as for any binary tree). Moreover, since black-heights must be equal and at most every other edge can be red, we have $h(t_\ell[v]), h(t_r[v]) \ge \frac{1}{2}h(t[v]) - 1 \ge \frac{1}{2}\lg(B) - 1$, which also lower bounds the size of $t_\ell[v]$ and $t_r[v]$. So $t_\ell[v]|, |t_r[v]| = \omega(1)$ as $B \to \infty$.

Example F.4 (Weight-balanced BSTs): Let $\mathcal{T}(\mathcal{W}_\alpha)$ denote the set of *α-weight-balanced binary trees* (in the sense of BB[α], [69]): A binary tree is α-weight-balanced, if for every node v of t, we have $|t_\ell[v]| + 1 \ge \alpha(|t[v]| + 1)$ and $|t_r[v]| + 1 \ge \alpha(|t[v]| + 1)$ (note that this is a special case of β-balanced binary trees considered in the proof of Lemma E.16). The property $\mathcal{W}_\alpha$ is fringe-hereditary by definition and it is easy to see that $\mathcal{W}_\alpha$ is heavy-twigged.

From the proof of Lemma E.16, we furthermore find that α-weight-balanced binary trees are worst-case fringe dominated. Unfortunately, we are not aware of a counting result for these trees, and so it remains a conjecture that α-weight-balanced binary trees are log-linear and thus amenable to the same treatment.

However, the uniform subclass source $\mathcal{U}_{\mathcal{W}_\alpha}$ can be modeled as a worst-case fringe dominated fixed-size source: If we set

$$p(\ell, n-\ell) \quad = \quad \begin{cases} \dfrac{|\mathcal{T}_\ell(\mathcal{W}_\alpha)||\mathcal{T}_{n-\ell}(\mathcal{W}_\alpha)|}{|\mathcal{T}_{n+1}(\mathcal{W}_\alpha)|} & \text{if } \ell+1, n-\ell+1 \geq \alpha(n+2), \\ 0 & \text{otherwise} \end{cases}$$

for every $n \in \mathbb{N}$, then the corresponding fixed-size tree source $\mathcal{S}_{fs}(p)$ corresponds to a uniform probability distribution on $\mathcal{T}_n(\mathcal{W}_\alpha)$ for every $n \in \mathbb{N}$.

F.1. Universality for Uniform-Subclass Sources

In order to show universality of the hypersuccinct code from Section C.1 with respect to uniform subclass sources, we first derive a source-specific encoding with respect to the uniform subclass source, against which we will then compare our hypersuccinct code:

An encoding $E_{\mathcal{P}}(t)$ that stores a given binary tree $t \in \mathcal{T}_n(\mathcal{P})$ in $\lg(|\mathcal{T}_n(\mathcal{P}|) + O(\log n)$ many bits is obtained as follows: Let $t_1, \ldots, t_{|\mathcal{T}_n(\mathcal{P})|}$ denote an enumeration of all elements in $\mathcal{T}_n(\mathcal{P})$. In order to encode a binary tree $t \in \mathcal{T}_n(\mathcal{P})$, we first encode its size (plus one, in order to incorporate the case that t is the empty binary tree), in gamma code, $\gamma(n+1)$, followed by its number $i \in [|\mathcal{T}_n(\mathcal{P})|]$ in the enumeration of all binary trees in $\mathcal{T}_n(\mathcal{P})$, using $\lfloor \lg(|\mathcal{T}_n(\mathcal{P})|) \rfloor + 1$ many bits. Thus, such an encoding $E_{\mathcal{P}} : \mathcal{T}(\mathcal{P}) \to \{0,1\}^\star$ spends at most

$$|E_{\mathcal{P}}(t)| \quad \leq \quad \lg(|\mathcal{T}_n(\mathcal{P})|) + 2\lg(n+1) + 2 \quad = \quad \lg\left(\frac{1}{\mathbb{P}[t]}\right) + 2\lg(n+1) + 2 \tag{10}$$

many bits in order to encode $t \in \mathcal{T}_n(\mathcal{P})$. We remark that $E_{\mathcal{P}} : \mathcal{T}(\mathcal{P}) \to \{0,1\}^\star$ is a single prefix-free code on $\mathcal{T}(\mathcal{P})$. We find the following:

Lemma F.5 (Great-branching lemma for $\mathcal{U}_{\mathcal{P}}$): *Let $\mathcal{U}_{\mathcal{P}}$ be a fringe-hereditary, worst-case fringe-dominated, log-linear, heavy-twigged uniform subclass source and let $t \in \mathcal{T}_n$ with $\mathbb{P}[t] > 0$. Then*

$$\sum_{i=1}^{m} |C(\mu_i)| \leq \lg\left(\frac{1}{\mathbb{P}[t]}\right) + o(n),$$

where C is a Huffman code for the sequence of micro trees $\mu_1, \ldots, \mu_m$ from our tree covering scheme (see Section C.1).

Proof: The proof works in a similar way as the proof of Lemma E.24: Let μ_i be an internal (non-fringe) micro tree. By *bough*(μ_i), we again denote the subtree of μ_i induced by the set of nodes that are ancestors of μ_i's child micro trees (ancestors of the portals); the boughs of a micro tree are the paths from the portals to the micro tree root. Hanging off the boughs of μ_i are (fringe) subtrees $f_{i,1}, \ldots, f_{i,|bough(\mu_i)|+1}$, listed in depth-first order of the bough nulls these subtrees are attached to. In general, some of these subtrees might be the empty tree – however, as the uniform subclass source $\mathcal{U}_{\mathcal{P}}$ we consider is *heavy-twigged*, and as every node v that belongs to *bough*(μ_i) satisfies $|t[v]| \geq B$ by Lemma C.2 (where $B = B(n)$ is the parameter from the tree covering algorithm), we find that $|f_{i,j}| \in \omega(1)$, except for possibly two exceptions, as the portals are replaced by null pointers in μ_i. Recall that the portal nodes themselves are not part of μ_i and hence not part of $bough(\mu_i)$. As in the proof of Lemma E.24, we construct a new encoding for micro trees against which we can compare the hypersuccinct code, another " great-branching" code, $\tilde{G}_B$, as follows:

$$\tilde{G}_B(\mu_i) \;=\; \begin{cases} \mathtt{0} \cdot E_{\mathcal{P}}(\mu_i), & \text{if } \mu_i \text{ is a fringe micro tree;} \\ \mathtt{1} \cdot \gamma(|bough(\mu_i)|) \cdot BP(bough(\mu_i)) \cdot & \\ \quad E_{\mathcal{P}}(f_{i,1}) \cdots E_{\mathcal{P}}(f_{i,|bough(\mu_i)|+1}), & \text{otherwise.} \end{cases}$$

Note that $\tilde{G}_B$ is well-defined: As the encoding $E_{\mathcal{P}}$ is only applied to fringe subtrees μ_i and $f_{i,j}$ of t, which satisfy property $\mathcal{P}$ as $\mathcal{U}(\mathcal{P})$ is fringe-hereditary, we find that $\mathbb{P}[\mu_i], \mathbb{P}[f_{i,j}] > 0$. Moreover, note that formally, $\tilde{G}_B$ is *not* a prefix-free code over Σ_μ: there can be micro tree shapes that are assigned *several* codewords by $\tilde{G}_B$, depending on which nodes are portals to other micro trees (if any). But $\tilde{G}_B$ is uniquely decodable to local shapes of micro trees, and can thus be seen as a *generalized* prefix-free code. In terms of the encoding length, assigning more than one codeword never makes the code worse, thus a Huffman code minimizes the encoding length over the larger class of generalized prefix-free codes. In particular, the Huffman code C for micro trees used in the hypersuccinct code achieves no worse encoding length than the great-branching code:

$$\sum_{i=1}^{m} |C(\mu_i)| \;\le\; \sum_{i=1}^{m} |\tilde{G}_B(\mu_i)|.$$

With $\mathcal{I}_0 = \{i \in [m] : \mu_i \text{ is a fringe micro tree in } t\}$, and $\mathcal{I}_1 = [m] \setminus \mathcal{I}_0$, we have

$$\begin{aligned} \sum_{i=1}^{m} |\tilde{G}_B(\mu_i)| \;&=\; \sum_{i\in\mathcal{I}_0} |\tilde{G}_B(\mu_i)| + \sum_{i\in\mathcal{I}_1} |\tilde{G}_B(\mu_i)| \\ &\le\; \sum_{i\in\mathcal{I}_0} (1 + |E_{\mathcal{P}}(\mu_i)|) \\ &\quad + \sum_{i\in\mathcal{I}_1} \Biggl(2 + 2\lg(|bough(\mu_i)|) + 2|bough(\mu_i)| + \sum_{j=1}^{|bough(\mu_i)|+1} |E_{\mathcal{P}}(f_{i,j})|\Biggr). \end{aligned}$$

With estimate (10) this is upper-bounded by

$$\begin{aligned} &\sum_{i\in\mathcal{I}_0} \biggl(3 + \lg\frac{1}{\mathbb{P}[\mu_i]} \;+\; 2\lg(|\mu_i|+1)\biggr) \;+\; \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|bough(\mu_i)|+1} \biggl(2 + \lg\frac{1}{\mathbb{P}[f_{i,j}]} + 2\lg(|f_{i,j}|+1)\biggr) \\ &\qquad + \sum_{i\in\mathcal{I}_1} (2 + 2\lg(|bough(\mu_i)|) + 2|bough(\mu_i)|) \\ &\le\; \sum_{i\in\mathcal{I}_0} \lg\frac{1}{\mathbb{P}[\mu_i]} \;+\; \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|bough(\mu_i)|+1} \lg\frac{1}{\mathbb{P}[f_{i,j}]} \;+\; O(m\log\mu) + O\Biggl(\sum_{i\in\mathcal{I}_1} |bough(\mu_i)|\log\mu\Biggr). \end{aligned}$$

By the log-linearity of the uniform subclass source $\mathcal{U}_{\mathcal{P}}$, we find $\lg(1/\mathbb{P}[\mu_i]) = \lg\Bigl(|\mathcal{T}_{|\mu_i|}(\mathcal{P})|\Bigr) = c|\mu_i| + \vartheta(|\mu_i|)$ and $\lg(1/\mathbb{P}[f_{i,j}]) = \lg\Bigl(|\mathcal{T}_{|f_{i,j}|}(\mathcal{P})|\Bigr) = c|f_{i,j}| + \vartheta(|f_{i,j}|)$, with $\vartheta(n) \in o(n)$ and $c > 0$ constant, for the fringe subtrees μ_i and $f_{i,j}$ (if $f_{i,j}$ is the empty binary tree, we simply have

$\lg(1/\mathbb{P}[f_{i,j}]) = 0$ by assumption). As $\mathcal{U}_{\mathcal{P}}$ is heavy-twigged, we have $|f_{i,j}| \in \omega(1)$ for all subtrees $f_{i,j}$ which are not the empty tree. Furthermore, we find $|\mu_i| \in \omega(1)$ for all fringe micro trees μ_i of t by Lemma C.2. Hence, as $\vartheta(n) \in o(n)$, and as the trees μ_i for $i \in \mathcal{I}_0$ and $f_{i,j}$ are disjoint subtrees of t, we have

$$\sum_{i\in\mathcal{I}_0} \vartheta(|\mu_i|) \;+\; \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|bough(\mu_i)|+1} \vartheta(|f_{i,j}|) \;=\; o(n).$$

Thus, we find

$$\begin{aligned}
&\sum_{i\in\mathcal{I}_0} \lg\frac{1}{\mathbb{P}[\mu_i]} \;+\; \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|bough(\mu_i)|+1} \lg\frac{1}{\mathbb{P}[f_{i,j}]} \\
&\quad= \; c\sum_{i\in\mathcal{I}_0}(|\mu_i| + \vartheta(|\mu_i|)) \;+\; c\sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|bough(\mu_i)|+1} (|f_{i,j}| + \vartheta(|f_{i,j}|)) \\
&\quad\le \; c|t| + o(n) \;=\; \lg(|\mathcal{T}_n(\mathcal{P})|) + o(n) \;=\; \lg\left(\frac{1}{\mathbb{P}[t]}\right) + o(n).
\end{aligned}$$

It remains to upper-bound the error terms: Lemma C.2 implies that any node v in the bough of a micro tree satisfies $|t[v]| \ge B$. Thus, the total number of nodes of t which belong to a bough of t is therefore upper-bounded by $n_{\ge B}(t)$. Altogether, we thus obtain

$$\sum_{i=1}^{m} |C(\mu_i)| \;\le\; \lg\left(\frac{1}{\mathbb{P}[t]}\right) \;+\; O(m\log\mu) \;+\; O(n_{\ge B}(t)\log\mu) \;+\; o(n).$$

By a pigeon-hole argument, we find $n_{\ge B}(t) = \Omega(n/B)$ and as $\mathcal{P}$ is worst-case fringe-dominated, we have $n_{\ge B(n)}(t) \in o(n/\log(B(n)))$. Furthermore, as $\mu = \Theta(\log n)$ and $m = \Theta(n/\log n)$ (see Section C.1), we have

$$\sum_{i=1}^{m} |C(\mu_i)| \;\le\; \lg\left(\frac{1}{\mathbb{P}[t]}\right) + o(n). \qquad \square$$

Theorem F.6 (Universality for tame uniform sources): *Let $\mathcal{U}_{\mathcal{P}}$ be a fringe-hereditary, worst-case fringe-dominated, log-linear, heavy-twigged uniform subclass source. The hypersuccinct code $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ satisfies*

$$|\mathsf{H}(t)| \;\le\; \lg\left(\frac{1}{\mathbb{P}[t]}\right) + o(n)$$

for every binary tree t of size n with $\mathbb{P}[t] > 0$.

Theorem F.6 follows from Lemma F.5 and Lemma C.3. In particular, we obtain the following corollary from Theorem F.6 (see Example F.2 and Example F.3):

Corollary F.7: *The hypersuccinct code* $\mathsf{H} : \mathcal{T} \to \{0,1\}^\star$ *optimally compresses*

(i) ***AVL trees*** *of size* n*, drawn uniformly at random from the set* $\mathcal{T}_n(\mathcal{A})$ *of all AVL trees of size* n*, using*

$$|\mathsf{H}(t)| \quad \le \quad \lg(|\mathcal{T}_n(\mathcal{A})|) + o(n) \quad \approx \quad 0.938148n + o(n)$$

many bits and

(ii) ***red-black trees*** *of size* n*, drawn uniformly at random from the set* $\mathcal{T}_n(\mathcal{R})$ *of all red-black trees of size* n*, using*

$$|\mathsf{H}(t)| \quad \le \quad \lg(|\mathcal{T}_n(\mathcal{R})|) + o(n) \quad \approx \quad 0.879146n + o(n)$$

many bits.

G. Range-Minimum Queries With Runs

In this appendix, we give the proofs of the results from Section 5.3.

G.1. Lower Bound

In this section, we proof Theorem 5.2.

We refer to a run of length one as a singleton run. The types of nodes in the Cartesian tree (whether or not their left and right children exist, see Section D) directly reflect their role in runs: A binary node is a run head of a non-singleton run, a leaf is the last node of non-singleton run, a right-unary node (i.e., unary node with a right child) is a middle node of run and a left-unary node is a singleton run. The leftmost node, i.e., the node with smallest inorder rank, is the only exception to this rule: if the leftmost run is a singleton run, the leftmost node is a leaf; otherwise it is right-unary.

In any case, a Cartesian tree for an array with r runs that has b binary nodes and u_ℓ left-unary nodes thus satisfies $r = b + u_\ell + 1$: every binary node represents the non-singleton run that begins with it, every left-unary node represents the singleton run at that position, and the leftmost run is counted separately. (Note that we do not double count the latter because the leftmost node is by definition neither binary nor left-unary.) We therefore obtain a lower bound for the number of equivalence classes among length-n arrays with r runs under range-minimum queries by counting binary trees with a given number of nodes n and a given number of nodes of certain types.

That r is the sum of two quantities is inconvenient, hence we instead consider the following sequence of bijections (see Figure 4). First, we map Cartesian trees t of n nodes bijectively to balanced-parenthesis (BP) strings of n pairs of parentheses as follows: The empty tree corresponds to the empty string. For a nonempty tree, we recursively compute the BP strings of the (potentially empty) left resp. right subtrees of the root; let these be denoted by L and R. Then the BP string for the entire tree is obtained as $L\texttt{(}R\texttt{)}$. (This is a variation of the canonical BP representation used in Part I.)

It is easy to check that the resulting sequence is indeed the push/pop sequence of a max-stack [21, 33] where '`(`' means push and '`)`' means pop. We map this sequence to a lattice path by replacing '`(`' by step vector $(1, +1)$ and '`)`' by $(1, -1)$; the resulting lattice path is a mountain-valley diagram (Dyck paths).

The important property of the above bijections is that they *preserve runs*: A *run end* is an index where the next number is smaller (or nonexistent). In the Cartesian tree, these are the leaves *and* left-unary nodes, in the BP string, these are the occurrences of '`()`' and in the mountain-valley representations, these are the *peaks*. The latter is known to be counted by the *Narayana numbers*: There are

$$N_{n,r} \quad = \quad \frac{1}{n}\binom{n}{r}\binom{n}{r-1} \quad = \quad \frac{r}{n(n+1-r)}\binom{n}{r}^2 \tag{11}$$

mountain-valley diagrams of length $2n$ with exactly r peaks [49]. This concludes the proof of Theorem 5.2; the asymptotic approximation for $\lg N_{n,r}$ immediately follows from the above closed form.

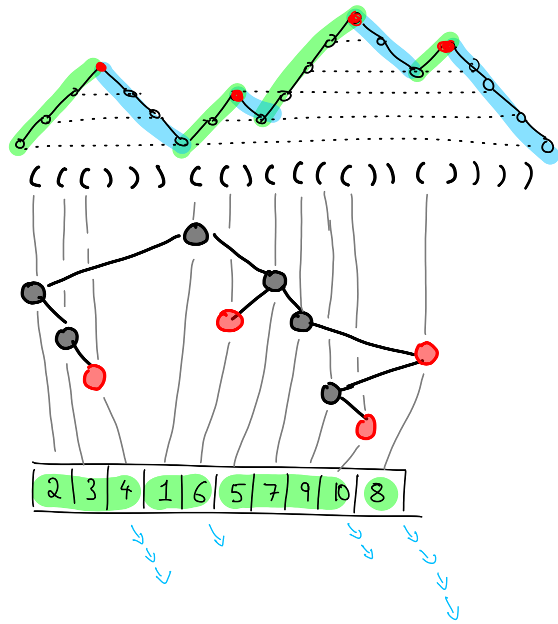


Figure 4: An example illustrating the bijections: The input array is $A = (2, 3, 4, 1, 6, 5, 7, 9, 10, 8)$, the min-oriented Cartesian tree is shown above with run ends highlighted in red. The BP string for the Cartesian tree is shown above the tree, with tree nodes connected to the corresponding opening parenthesis (note that nodes appear in inorder in the BP string). The maintain-valley (excess, Dyck path) representation of the BP string is on top; run ends correspond to peaks there.

G.2. Hypersuccinct RMQ with Runs

In this section, we prove Corollary 5.3. To this end, we show that using a hypersuccinct tree to represent the Cartesian tree of an array $A[1..n]$ with r increasing runs has a space usage that is bounded by $\lg N_{n,r} + o(n)$ bits. By Theorem 5.2, this space usage is optimal up to the $o(n)$ term.

As noted in Section G.1, the correspondence between runs and node types in the Cartesian tree can be made more specific by also specifying the number $s \in [r]$ of singleton runs: Singleton runs correspond to the left-unary nodes in the (min-oriented) Cartesian tree t, except possibly for a leftmost singleton run (which corresponds to a leaf). In either case, we will have $u_\ell = s \pm 1$ left-unary nodes and $\ell = r - s \pm 1$ leaves. That implies a number of binary nodes of $b = \ell - 1 \pm 1$; the remaining $u_r = n - b - \ell - u_\ell = n - 2r + s \pm 1$ nodes are right-unary nodes.

By Corollary 2.2, the hypersuccinct representation of the Cartesian tree t for $A[1..n]$ supports LCA-queries on t in $O(1)$ time and uses

$$|\mathsf{H}(t)| + o(n) \quad \leq \quad H_0^{\text{type}}(t) \; + \; o(n)$$

bits of space. We show that $H_0^{\text{type}}(t) \leq \lg N_{n,r} + o(n)$. Let

$$p \quad = \quad \left(\frac{b}{n}, \frac{u_l}{n}, \frac{u_r}{n}, \frac{\ell}{n} \right)$$

denote the empirical distribution of node types in t, and let $H(p)$ denote the entropy of this distribution. (For probability distribution $d = (d_1, d_2, \ldots, d_k)$, its entropy is defined by

$$H(d) \;=\; \sum_{i=1}^{k} d_i \lg\left(\frac{1}{d_i}\right),$$

as usual.)

By definition of the type-entropy H_0^{type} (cf. Definition D.1), we find $H_0^{\text{type}}(t) = nH(p)$. By our previous observations, p differs from

$$p' \;=\; \left(\frac{r-s}{n}, \frac{s}{n}, \frac{n-2r+s}{n}, \frac{r-s}{n}\right)$$

only by $\|p - p'\|_\infty \le \frac{2}{n}$. Using [80, Prop. 2.42], we thus find $H(p) \le H(p') + O(n^{-0.9})$ (this follows from Hölder-continuity of $x \mapsto x \ln x$). It thus remains to show that $n\,H(p') \le \lg N_{n,r} + o(n)$. By the grouping property of H, we have

$$n\,H(p') \;=\; n\biggl(H\Bigl(\frac{r}{n}, \frac{n-r}{n}\Bigr) + \frac{r}{n}H\Bigl(\frac{s}{r}, \frac{r-s}{r}\Bigr) + \frac{n-r}{n}H\Bigl(\frac{r-s}{n-r}, \frac{(n-r)-(r-s)}{n-r}\Bigr)\biggr).$$

In order to estimate the right-hand side, observe that it follows from [39, Eq. (5.22)] that $\sum_{s=0}^{r}\binom{r}{s}\binom{n-r}{r-s} = \binom{n}{r}$. Since all summands are positive, we have $\binom{r}{s}\binom{n-r}{r-s} \le \binom{n}{r}$ and hence

$$\lg\binom{r}{s} + \lg\binom{n-r}{r-s} \;\le\; \lg\binom{n}{r}, \qquad \text{for all } s \in [r]. \tag{12}$$

For a number $q \in [0,1]$, we set $h(q) = q\lg(1/q) + (1-q)\lg(1/(1-q))$. Using the standard inequality

$$\frac{2^{nh(q)}}{n+1} \;\le\; \binom{n}{qn} \;\le\; 2^{nh(q)}, \qquad nq \in [0..n], \tag{13}$$

we find

$$\begin{aligned}
rh\Bigl(\frac{s}{r}\Bigr) \;+\; (n-r)h\Bigl(\frac{r-s}{n-r}\Bigr) &\underset{(13)}{\le} \lg\binom{r}{s} + \lg\binom{n-r}{r-s} \;+\; \lg(r+1) + \lg(n-r+1)\\
&\underset{(12)}{\le} \lg\binom{n}{r} \;+\; \lg(r+1) + \lg(n-r+1)\\
&\underset{(13)}{\le} nh\Bigl(\frac{r}{n}\Bigr) \;+\; \lg(r+1) + \lg(n-r+1).
\end{aligned}$$

We thus have

$$\begin{aligned}
nH(p') &= n\biggl(H\Bigl(\frac{r}{n}, \frac{n-r}{n}\Bigr) + \frac{r}{n}H\Bigl(\frac{s}{r}, \frac{r-s}{r}\Bigr) + \frac{n-r}{n}H\Bigl(\frac{r-s}{n-r}, \frac{(n-r)-(r-s)}{n-r}\Bigr)\biggr)\\
&= n\biggl(h\Bigl(\frac{r}{n}\Bigr) + \frac{r}{n}h\Bigl(\frac{s}{r}\Bigr) + \frac{n-r}{n}h\Bigl(\frac{r-s}{n-r}\Bigr)\biggr)\\
&\le 2nh\Bigl(\frac{r}{n}\Bigr) \;+\; O(\log n)\\
&\underset{(13)}{\le} 2\lg\binom{n}{r} \;+\; O(\log n)\\
&\underset{(11)}{\le} \lg N_{n,r} \;+\; O(\log n).
\end{aligned}$$

So in total, we have shown that

$$|\mathsf{H}(t)| \;\leq\; \lg N_{n,r} \;+\; o(n),$$

which implies Corollary 5.3.

Part II. Ordinal Trees

Most results for binary trees can be extended to ordinal trees, but some additional arguments resp. restrictions are necessary because of large-degree nodes. Our results with respect to ordinal trees are presented in this part.

H. Hypersuccinct Ordinal Trees

The Farzan-Munro tree decomposition algorithm [16] is used to decompose an ordinal tree into subtrees, so-called *micro trees*. In the following, we recall the properties of this tree covering method (for more details, see Section B.3):

Lemma H.1 (Tree covering, [16, Thm. 1]): *For any parameter $B \geq 1$, an ordinal tree with n nodes can be decomposed, in linear time, into connected subtrees (so-called* micro trees*) with the following properties:*

(i) *Micro trees are pairwise disjoint except for (potentially) sharing a common micro tree root.*

(ii) *Each micro tree contains at most $2B$ nodes.*

(iii) *The overall number of micro trees is $\Theta(n/B)$.*

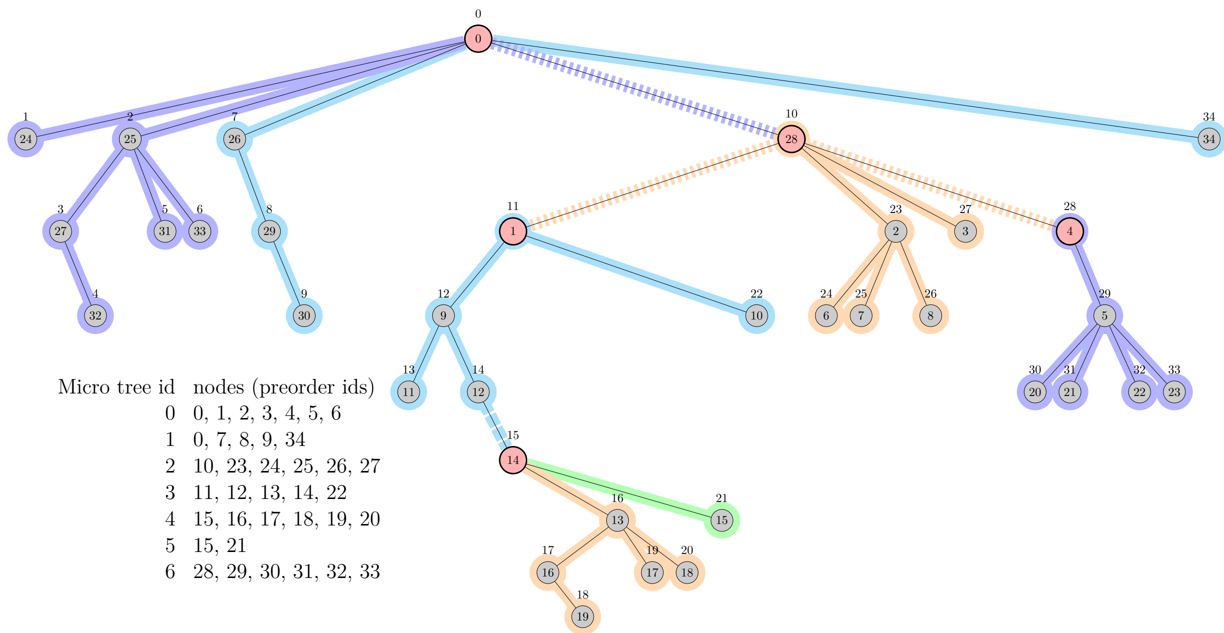
(iv) *Apart from edges leaving the micro tree root, at most one other edge leads to a node outside of this micro tree. This edge is called the "external edge" of the micro tree.*

By inspection of the proof in [16], we can say a bit more: If v is a node in the tree and is also the root of several micro trees of the decomposition, then the way that v's children (in the entire tree) are divided among the micro trees is into *consecutive* blocks. Each micro tree contains at most two of these blocks. (This case arises when the micro tree root has exactly one heavy child in the decomposition algorithm.) In binary trees, a micro tree is always an entire fringe subtree except for at most two entire subtrees, which are removed from it. In ordinal trees, the possibility of large node degrees makes such a decomposition impossible: here an arbitrary number of children (and their subtrees) can be missing in a micro tree root, and a single node in the original tree can be the (shared) root of many micro trees.

H.1. Hypersuccinct Code

In this section, we describe a universal code for ordinal trees based on the Farzan-Munro algorithm using just one level of micro trees. The purpose is to give a self-contained description of the mere representation of an ordinal tree (as opposed to a succinct data structure) that admits compression as a universal code. The exposition in [16] mixes this description with the details of the data structures needed for navigation.

We fix the parameter B, so that the maximal micro tree size is $\mu = \lceil \frac{1}{4} \lg n \rceil$ i.e., we set $B = \lceil \frac{1}{8} \lg n \rceil$. The code of the ordinal tree $t \in \mathfrak{T}_n$ is then obtained as follows: Decompose the tree into micro trees $\mu_1, \ldots, \mu_m$ where $m = \Theta(n/B) = \Theta(n/\log n)$. Recall that each micro tree μ_i can have the following connections to other micro trees:



Micro tree id	nodes (preorder ids)
0	0, 1, 2, 3, 4, 5, 6
1	0, 7, 8, 9, 34
2	10, 23, 24, 25, 26, 27
3	11, 12, 13, 14, 22
4	15, 16, 17, 18, 19, 20
5	15, 21
6	28, 29, 30, 31, 32, 33

Figure 5: Example tree with $n = 34$ nodes, partitioned using $B = 6$.

- an edge to one parent micro tree,
- an external edge to one child micro tree, leaving from some node of the micro tree (and inserted at some child rank),
- an arbitrary number of other subtrees of the shared root; these micro trees can contain the shared root or not.

The *top-tier* Υ of the tree is obtained by contracting each micro tree into a single node; shared roots are copied to each micro tree. Two micro trees are connected by an edge in Υ if there is an edge between some nodes in these micro trees in t. Since several micro trees can contain the root of the tree, we add a dummy root to Υ to turn it into a single tree. Figure 6 shows an example.

To be able to distinguish the different forms of interactions listed above, additional information for parent-child edges in Υ is stored. By construction, edges between micro trees always lead to the root of the child micro tree, but the other endpoint will have to be encoded. We observe that there are the following types of edges between a parent micro tree P and its child C:

(i) *new leftmost root child*
The root of C is a child of the root of P and comes before all children of P's root that lie inside P in the left-to-right order of the children. Moreover, there is no other child component C' of P that shares the root with C and comes before C in the child order.

(ii) *continued leftmost root child*
The root of C is a child of the root of P and comes before all children of P's root that lie inside P in the left-to-right order of the children, but it shares its root with the child component immediately before C in the child order.

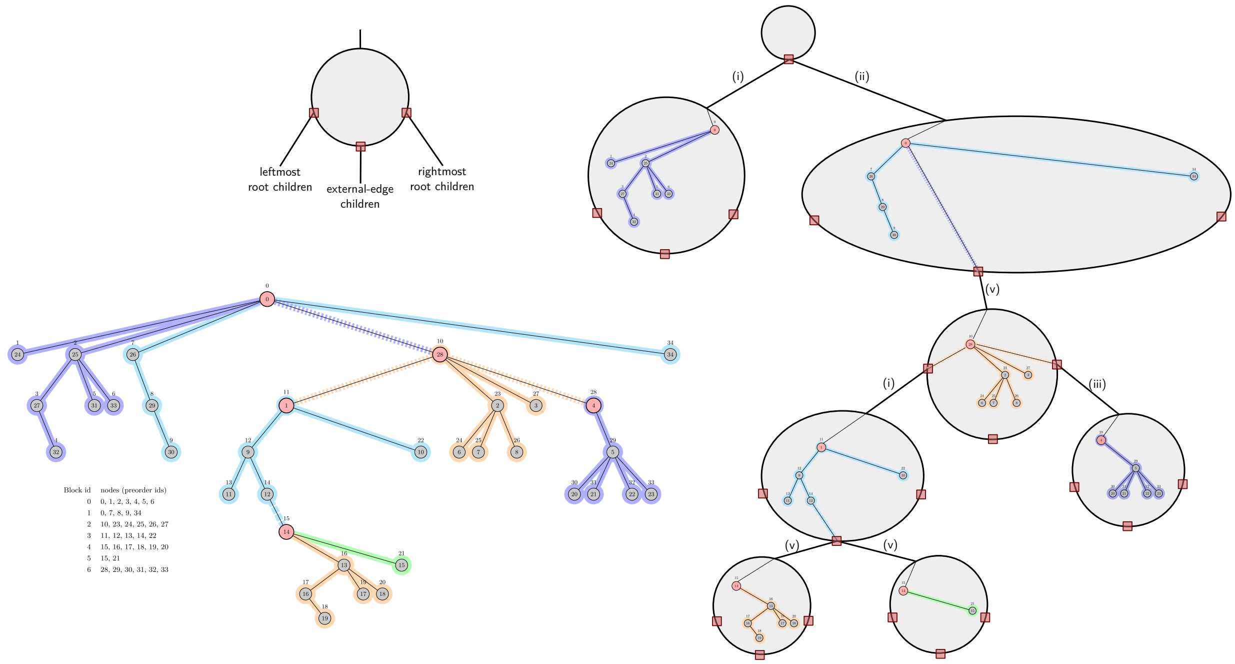


Figure 6: The tree from Figure 5 (left) and the top-tier tree Υ (right) corresponding to the covering with the edge types. Edge types are also visualized through different exit points for leftmost, rightmost, and external edges for illustration purposes.

(iii) *new rightmost root child*
The root of C is a child of the root of P and C's root comes after all root children included in P. Moreover, there is no other child component C' of P that shares the root with C.

(iv) *continued rightmost root child*
The root of C is a child of the root of P and C's root comes after all root children included in P, but it shares its root with the child component immediately before C in the child order.

(v) *external-edge child*
Any other edge. By construction, all external-edge child components of P share a common root, so there is no need to distinguish new and continued external edges.

We note that path nodes can give rise to an external-edge child C whose root is a child of P's root. This happens only when we greedily pack across the gap left by the permanent component of a single heavy child. P cannot have another external edge then, so we are free to use P's external-edge "slot" to link to C.

The top tier is again an ordinal tree, $\Upsilon \in \mathfrak{T}_{m+1}$. For the micro trees, we observe that because of their limited size, there are fewer different possible shapes of ordinal trees than we have micro trees. The crucial idea of our hypersuccinct encoding is again to treat each shape of a micro tree as a letter in the alphabet $\Sigma_\mu \subseteq \bigcup_{s \le \mu} \mathfrak{T}_s$ of micro tree shapes and to compute a Huffman code $C : \Sigma_\mu \to \{0,1\}^\star$ based on the frequency of occurrences of micro tree shapes in the sequence $\mu_1, \ldots, \mu_m \in \Sigma_\mu^m$. For our hypersuccinct code, we then use a *length-restricted* version $\bar{C} : \Sigma_\mu \to \{0,1\}^\star$ obtained from C using a variant of the simple cutoff technique from Definition B.4

for ordinal trees (using the balanced parenthesis encoding for ordinal trees). Furthermore, for each micro tree, we have to encode the portal for the external edges (if they exist) and the type of its parent edge (i)–(v). For that, we store the micro-tree-local preorder rank of the node and the child rank at which the external edges have to be inserted using $\lceil \lg(\mu + 1) \rceil$ bits each.

We can thus encode an ordinal tree $t \in \mathfrak{T}_n$ as follows:

1. Store n and m in Elias gamma code,
2. followed by the balanced-parenthesis (BP) bitstring for Υ.
3. Next comes an encoding for $\bar{C}$; for simplicity, we simply list all possible codewords and their corresponding ordinal trees by storing the size (in Elias-gamma code) followed by their BP sequence.
4. Then, we list the Huffman codes $\bar{C}(\mu_i)$ of all micro trees in DFS order (of Υ).
5. Then, we store 2 $\lceil \lg(\mu + 1) \rceil$-bit integers to encode the portal of each micro tree in DFS order (of Υ).
6. Finally, we encode the type of the parent edge using 3 bits of each micro tree, again in DFS order.

Altogether, this yields our *hypersuccinct code* $\mathsf{H} : \mathfrak{T} \to \{0,1\}^\star$ for ordinal trees. Decoding is possible by first recovering n, m, and Υ from the BP, then reading the Huffman code. We then replace each node in Υ by its micro tree in a depth-first traversal. Herein, we use the information about edge types in Υ to correctly connect the micro trees: partitioning children into leftmost and rightmost root children places them in the appropriate order into the list of children of the parent component's root. For type (ii) and (iv) children, we delete the component root and instead add its children to the next type (i) resp. (iii) siblings component's root. Finally, for type (v) children, we use the information about portals to find their place in a node's child list, and for all but the leftmost of them, also merge their roots with the left sibling component. With respect to the length of the hypersuccinct code, we find the following:

Lemma H.2 (Hypersuccinct ordinal tree code): *Let $t \in \mathfrak{T}_n$ be an ordinal tree of n nodes, decomposed into micro trees $\mu_1, \ldots, \mu_m$ by the Farzan-Munro algorithm. Let C be an ordinary Huffman code for the string $\mu_1 \ldots \mu_m$, the local shapes of the micro trees. Then, the hypersuccinct code encodes t with a binary codeword of length*

$$|\mathsf{H}(t)| \;\le\; \sum_{i=1}^{m} |C(\mu_i)| + O\Bigl(n \frac{\log\log n}{\log n}\Bigr).$$

Proof: It is easy to check that all parts of the hypersuccinct ordinal-tree code except Part 4 require $O(n \log\log n / \log n)$ bits of space. Let $t \in \mathfrak{T}_n$. The analysis of the number of bits needed to store parts 1–5 is identical to the binary-tree case: Part 1 needs $O(\log n)$ bits and Part 2 requires $2m + 2 = \Theta(n/\log n)$ bits. For Part 3, observe that

$$|\Sigma_\mu| \;\le\; \sum_{s \le \lceil \lg n/4 \rceil} 4^s \;<\; \frac{4}{3} \cdot 4^{\lg n/4+1} \;=\; \frac{16}{3}\sqrt{n}.$$

With the worst-case cutoff technique (adapted to ordinal trees) from Definition B.4, $\bar{C}(\mu_i) \le 1 + 2\mu \sim \frac{1}{2}\lg n$, so we need asymptotically $O(\sqrt{n})$ entries / codewords in the table, each of size $O(\mu) = O(\log n)$, for an overall table size of $O(\sqrt{n}\log n)$. Part 5 uses $m \cdot 2\lceil \lg(\mu+1)\rceil = \Theta(\frac{n}{B}\log B) = \Theta(n \cdot \frac{\log\log n}{\log n}) = o(n)$ bits of space. Part 6 uses $3m = \Theta(n/\log n)$ bits. It remains to analyze Part 4, which is again similar to the binary-tree case: We note that by applying the worst-case pruning scheme of Definition B.4, we waste 1 bit per micro tree compared to a pure, non-restricted Huffman code. But the wasted bits amount to $m = O(n/\log n)$ bits in total:

$$\begin{aligned}\sum_{i=1}^{m} \bar{C}(\mu_i) &= \sum_{i=1}^{m} \min\{|C(\mu_i)| + 1, 2|\mu_i| + 2\lceil \lg|\mu_i| + 1\rceil + 2\} \\ &\le \sum_{i=1}^{m} (|C(\mu_i)| + 1) \\ &= \sum_{i=1}^{m} |C(\mu_i)| + O(n/\log n).\end{aligned}$$

This finishes the proof. □

`parent`(v)	the parent of v, same as `anc`$(v, 1)$
`degree`(v)	the number of children of v
`child`(v, i)	the ith child of node v ($i \in \{1, \dots, \texttt{degree}(v)\}$)
`child_rank`(v)	the number of siblings to the left of node v plus 1
`depth`(v)	the depth of v, i.e., the number of edges between the root and v
`anc`(v, i)	the ancestor of node v at depth `depth`$(v) - i$
`nbdesc`(v)	the number of descendants of v
`height`(v)	the height of the subtree rooted at node v
`LCA`(v, u)	the lowest common ancestor of nodes u and v
`leftmost_leaf`(v)	the leftmost leaf descendant of v
`rightmost_leaf`(v)	the rightmost leaf descendant of v
`level_leftmost`(ℓ)	the leftmost node on level ℓ
`level_rightmost`(ℓ)	the rightmost node on level ℓ
`level_pred`(v)	the node immediately to the left of v on the same level
`level_succ`(v)	the node immediately to the right of v on the same level
`node_rank`$_X(v)$	the position of v in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}, \texttt{DFUDS}\}$, i.e., in a preorder, postorder, inorder, DFUDS order, or level-order traversal of the tree
`node_select`$_X(i)$	the ith node in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}, \texttt{DFUDS}\}$
`leaf_rank`(v)	the number of leaves before and including v in preorder
`leaf_select`(i)	the ith leaf in preorder

Table 6: Navigational operations on succinct ordinal trees. (v denotes a node and i an integer).

As for binary trees, the representation of ordinal trees based on the hypersuccinct code can be turned into a data structure:

Theorem H.3 (Tree covering index for ordinal trees [16]): *Let $t \in \mathfrak{T}_n$ denote an ordinal tree, decomposed into micro trees $\mu_1, \dots, \mu_m$ with the tree covering algorithm. Assuming access to a data structure that maps i to $BP(\mu_i)$ in constant-time, there is a data structure occupying $o(n)$ additional bits of space that supports all operations from Table 6 in constant time.*

I. Memoryless Ordinal Tree Sources

For an ordinal tree $t \in \mathfrak{T}$ and a node v of t, let $\deg_t(v)$ denote the (out-)degree of v. We leave out the subscript t, if the tree t is clear from the context. With ν_i^t we denote the number of nodes of degree i of t. A *degree distribution* $d = (d_i)_{i\in\mathbb{N}_0}$ is a sequence of non-negative real numbers, such that $\sum_{i=0}^{\infty} d_i = 1$. A degree distribution assigns a probability $\mathbb{P}[t]$ to an ordinal tree by

$$\mathbb{P}[t] \quad = \quad \prod_{v\in t} d_{\deg(v)} \quad = \quad \prod_{i=0}^{|t|} (d_i)^{\nu_i^t}. \tag{14}$$

That is, a degree distribution d can be used to randomly construct an ordinal tree as follows: In a top-down way, starting at the root node, we determine for each node its degree i: The probability that a node is of degree i is given by d_i. If $i = 0$, then this node becomes a leaf, otherwise we attach i many children to the node and continue the process at these children. Note that this process might produce infinite trees with non-zero probability. In order to obtain finite trees with non-zero probability, we assume that $d_0 > 0$. In [52], the following notion of empirical entropy for trees was introduced:

Definition I.1 (Degree-entropy): *Let $t \in \mathfrak{T}$. The (unnormalized)* degree-entropy $H^{\deg}(t)$ *of t is the zeroth order entropy of the node degrees:*

$$H^{\deg}(t) \quad = \quad \sum_{i=0}^{|t|} \nu_i^t \lg\left(\frac{|t|}{\nu_i^t}\right).$$

We say that a degree distribution is the *empirical degree distribution* of an ordinal tree t, if $d_i = \nu_i^t/|t|$ for every index $0 \le i \le |t|$. In particular, if d is the empirical degree distribution of an ordinal tree $t \in \mathfrak{T}$, we have

$$\lg\left(\frac{1}{\mathbb{P}[t]}\right) \quad = \quad \sum_{i=0}^{|t|} \nu_i^t \lg\left(\frac{1}{d_i}\right) \quad = \quad \sum_{i=0}^{|t|} \nu_i^t \lg\left(\frac{|t|}{\nu_i^t}\right) \quad = \quad H^{\deg}(t).P$$

Example I.2 (Full m-ary trees): Probability distributions over full m-ary trees, i.e., trees where each node has either exactly m or 0 children, are obtained from degree distributions $(d_i)_{i\in\mathbb{N}_0}$ with $d_0, d_m > 0$ and $d_i = 0$ for $i \neq m, 0$. It is easy to see that a full m-ary tree t with ν_m^t many inner nodes (of degree m) always consists of $\nu_0^t = (m-1)\nu_m^t + 1$ many leaves, and is thus always of size $m\nu_m^t + 1$. The number of full m-ary trees of size $n = m\nu + 1$, for $\nu \in \mathbb{N}$, is given by [22]:

$$\frac{1}{m\nu + 1}\binom{m\nu + 1}{\nu}. \tag{15}$$

Let d be the degree distribution with $d_0 = 1/m$ and $d_m = (m-1)/m$. We have

$$\lg\left(\frac{1}{\mathbb{P}[t]}\right) \quad = \quad \nu\lg(m) + ((m-1)\nu + 1)\lg\left(\frac{m}{m-1}\right)$$

for every full m-ary tree t of size $m\nu + 1$, which is asymptotically, by (15), the minimum number of bits needed to represent a full m-ary tree of size $m\nu + 1$.

Given a degree distribution d, Equation (14) suggests a route for an encoding that encodes an ordinal tree $t \in \mathfrak{T}$ with $\mathbb{P}[t] > 0$ in $\lg(1/\mathbb{P}[t])$ (plus lower-order terms) many bits: Such an encoding may spend $\lg(1/d_i)$ many bits per node v of t of degree $\deg(v) = i$. Assuming that the degree distribution is known (and need not be stored as part of the encoding), we can use *arithmetic coding* to encode the degree of node v in that many bits: However, d can possibly consist of countably many positive coefficients, thus, we have to adapt the process of arithmetic coding slightly: In order to encode the degree $\deg(v) \in \mathbb{N}_0$ of a node v, we consider $\deg(v)$ as a unary string $s = 0^{\deg(v)}1$, which we encode using arithmetic coding as follows: In order to encode the kth symbol of s, we feed the arithmetic coder with the model that the next symbol is a number $s[k] \in \{0, 1\}$, the probability for $s[k] = 1$ being $d_{k-1}/(d_{k-1} + d_k + d_{k+1} + \dots)$. Thus, arithmetic coding uses

$$\begin{aligned}
&\sum_{k=0}^{\deg(v)-1} \lg\left(\left(1 - \frac{d_k}{\sum_{i\ge k} d_i}\right)^{-1}\right) + \lg\left(\frac{\sum_{i\ge\deg(v)} d_i}{d_{\deg(v)}}\right) \\
&= \sum_{k=0}^{\deg(v)-1} \left(\lg\left(\sum_{i\ge k} d_i\right) - \lg\left(\sum_{i\ge k+1} d_i\right)\right) + \lg\left(\sum_{i\ge\deg(v)} d_i\right) + \lg\left(\frac{1}{d_{\deg(v)}}\right) \\
&= \lg\left(\frac{1}{d_{\deg(v)}}\right)
\end{aligned}$$

many bits to encode $s = 0^{\deg(v)}1$. An encoding D_d, dependent of a given degree-distribution d, stores a tree t as follows: While traversing the tree in depth-first order, we encode the degree $\deg(v)$ of each node v, using arithmetic encoding as described above. We can reconstruct the tree t recursively from its code $D_d(t)$, as we always know the degrees of the nodes we have already visited in the depth-first order traversal of the tree. As arithmetic encoding needs $\lg(1/d_{\deg(v)})$ bits per node v, plus at most 2 bits of overhead, the total number of bits needed in order to store an ordinal tree $t \in \mathfrak{T}$ with $\mathbb{P}[t] > 0$ is thus

$$|D_d(t)| \;\le\; \sum_{v\in t} \lg\left(\frac{1}{d_{\deg(v)}}\right) + 2.$$

If a degree distribution d is the empirical degree distribution of an ordinal tree t, i.e., $d_i = \nu_i^t/|t|$ for every $i \in [t]$, we find in particular:

$$|D_d(t)| \;\le\; \sum_{i=0}^{|t|} \nu_i^t \lg\left(\frac{|t|}{\nu_i^t}\right) + 2 \;=\; H^{\deg}(t) + 2.$$

The encoding D^d yields a prefix-free code for the set of ordinal trees which satisfy $\mathbb{P}[t] > 0$ with respect to the degree distribution d. In order to show that our hypersuccinct code is universal with respect to degree-distribution sources, we start with the following lemma:

Lemma I.3 (Micro tree code bound): *Let d be a degree distribution and let $t \in \mathfrak{T}_n$ be an ordinal tree of size n with $\mathbb{P}[t] > 0$. Then*

$$\sum_{i=1}^{m} |C(\mu_i)| \le \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{n \log\log n}{\log n}\right)$$

where C is a Huffman code for the sequence of micro trees $\mu_1, \ldots, \mu_m$ from our tree covering scheme (see Section H.1).

Proof: Recall that the micro trees $\mu_1, \ldots, \mu_m$ from our tree partitioning scheme for ordinal trees are pairwise disjoint except for (potentially) sharing a common subtree root and that apart from edges leaving the subtree root, at most one other edge leads to a node outside of the subtree (Lemma H.1). Thus, there are at most two nodes in each micro tree μ_i, whose degree in μ_i might not coincide with their degree in t: The root of μ_i, which we denote with ρ_i, and a node $\pi_i \neq \rho_i$. In particular, for every node $v \neq \pi_i, \rho_i$ of μ_i, we have $\deg_{\mu_i}(v) = \deg_t(v)$. Let $\mathrm{pos}(\pi_i)$ denote the depth-first order position of π_i in μ_i. With $D_d(\mu_i \setminus \rho_i)$ (respectively, $D_d(\mu_i \setminus \rho_i, \pi_i)$), we denote the following modification of D_d: While traversing the tree μ_i in depth-first order, we encode the degree $\deg_{\mu_i}(v)$ of each node v of μ_i, using arithmetic coding as in the encoding D_d, except that we skip the root ρ_i of μ_i (respectively, we skip the root ρ_i of μ_i and the node $\pi_i \neq \rho_i$ in μ_i from which an edge to a node outside of μ_i emerges). This is well-defined: We have $d_{\deg(v)} > 0$ for every node $v \neq \rho_i, \pi_i$ of μ_i whose degree we encode, as its degree in μ_i coincides with its degree in t and as $\mathbb{P}[t] > 0$. If we know $\deg_{\mu_i}(\rho_i)$, respectively, $\deg_{\mu_i}(\rho_i)$, $\deg_{\mu_i}(\pi_i)$ and $\mathrm{pos}(\pi_i)$, we are able to recover μ_i from $D_d(\mu_i \setminus \rho_i)$, respectively, $D_d(\mu_i \setminus \rho_i, \pi_i)$. Let $\mathcal{I}_0$ denote the set of indexes $i \in [m]$ for which μ_i does not contain a node other than (possibly) the root node from which an edge to a node outside of μ_i emerges, and let $\mathcal{I}_1 = [m] \setminus \mathcal{I}_0$. We define the following modified encoding:

$$\tilde{D}_d(\mu_i) = \begin{cases} 0 \cdot \gamma(\deg_{\mu_i}(\rho_i)) \cdot D_d(\mu_i \setminus \rho_i) & \text{if } i \in \mathcal{I}_0, \\ 1 \cdot \gamma(\deg_{\mu_i}(\rho_i)) \cdot \gamma(\deg_{\mu_i}(\pi_i) + 1) \cdot \gamma(\mathrm{pos}(\pi_i)) \cdot D_d(\mu_i \setminus \rho_i, \pi_i) & \text{otherwise.} \end{cases}$$

Note that formally, $\tilde{D}_d$ is *not* a prefix-free code over Σ_μ, as there can be micro tree shapes that are assigned *several* codewords by $\tilde{D}_d$. But $\tilde{D}_d$ can again be seen as a *generalized prefix-free code*, where more than one codeword per symbol is allowed, as $\tilde{D}_d$ is uniquely decodable to local shapes of micro trees. Thus, as a Huffman code minimizes the encoding length over the class of *generalized* prefix-free codes, we find:

$$\begin{aligned} \sum_{i=1}^{m} |C(\mu_i)| &\le \sum_{i=1}^{m} |\tilde{D}_d(\mu_i)| = \sum_{i \in \mathcal{I}_0} |\tilde{D}_d(\mu_i)| + \sum_{i \in \mathcal{I}_1} |\tilde{D}_d(\mu_i)| \\ &\le \sum_{i \in \mathcal{I}_0} (|D_d(\mu_i \setminus \rho_i)| + 2\lg\mu + 2) + \sum_{i \in \mathcal{I}_1} (|D_d(\mu_i \setminus \rho_i, \pi_i)| + 6\lg\mu + 4), \end{aligned}$$

as $\deg_{\mu_i}(\rho_i), \deg_{\mu_i}(\pi_i) + 1, \mathrm{pos}(\pi_i) \le \mu$. By definition of $|D_d(\mu_i \setminus \rho_i)|$ and $|D_d(\mu_i \setminus \rho_i, \pi_i)|$, and as $|\mathcal{I}_0| + |\mathcal{I}_1| = m$, this is upper-bounded by

$$\sum_{i \in \mathcal{I}_0} \sum_{\substack{v \in \mu_i \\ v \neq \rho_i}} \lg\left(\frac{1}{d_{\deg_{\mu_i}(v)}}\right) + \sum_{i \in \mathcal{I}_1} \sum_{\substack{v \in \mu_i \\ v \neq \rho_i, \pi_i}} \lg\left(\frac{1}{d_{\deg_{\mu_i}(v)}}\right) + 6m\lg\mu + 6m.$$

As every node v of t which is not the root node of a micro tree μ_i is contained in at most one subtree μ_i and as $\deg_{\mu_i}(v) = \deg_t(v)$ for every node $v \neq \pi_i, \rho_i$, we have

$$\sum_{i=1}^{m} |C(\mu_i)| \leq \sum_{v \in t} \lg\left(\frac{1}{d_{\deg_t(v)}}\right) + 6m \lg \mu + 6m = \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{n \log\log n}{\log n}\right),$$

as $m = \Theta(n/\log n)$ and $\mu = \Theta(\log n)$ (see Section H.1). This finishes the proof. □

Theorem I.4 (Universality for degree distribution): *Let d be a degree distribution. The hypersuccinct code $\mathsf{H} : \mathfrak{T} \to \{0,1\}^\star$ satisfies*

$$|\mathsf{H}(t)| \leq \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{n \log\log n}{\log n}\right)$$

for every $t \in \mathfrak{T}_n$ with $\mathbb{P}[t] > 0$. In particular, if d coincides with the empirical degree distribution of t, we have

$$|\mathsf{H}(t)| \leq H^{\deg}(t) + O\left(\frac{n \log\log n}{\log n}\right).$$

follows from Lemma I.3 and Lemma H.2.

In particular, for full m-ary trees from Example I.2, we obtain the following corollary from Theorem I.4:

Corollary I.5: *The hypersuccinct code $\mathsf{H} : \mathfrak{T} \to \{0,1\}^\star$ optimally compresses encodes* ***full m-ary trees*** *t of size $n = m\nu + 1$, drawn uniformly at random from the set of all full m-ary trees of size n, using $|\mathsf{H}(t)| \leq \nu \lg(m) + (m-1)\nu \lg(m/(m-1)) + O(n \log\log n/\log n)$ many bits.*

J. Fixed-Size Ordinal Tree Sources

For ordinal trees, we can define fixed-size sources in a similar way as for binary trees; such a source is characterized by a function $p : \mathbb{N}^+ \to [0,1]$ with

$$\sum_{\substack{k \in \mathbb{N} \\ n_1,\ldots,n_k \in \mathbb{N} \\ n_1+\cdots+n_k = n-1}} p(n_1, \ldots, n_k) \quad = \quad 1$$

for all $n \in \mathbb{N}$. The function p assigns a probability to each possible grouping of the $n-1$ descendants of an n-node ordinal tree into subtrees of the root. Note that the choice of subtree sizes of the root is equivalent to choosing a *composition* of $n-1$ into strictly positive summands; there are 2^{n-2} of these compositions (between each consecutive pair of $n-1$ dots, we can either place a barrier or not) – a lot more than the n choices for binary trees.

J.1. Monotonic Fixed-Size Sources

Definition J.1 (Monotonic source): *A fixed-size ordinal-tree source $\mathfrak{S}_{fs}(p)$ is called monotonic if p is*

(i) *weakly decreasing in every component,*

$$p(n_1,\ldots,n_{i-1},n_i,n_{i+1},\ldots,n_k) \;\;\geq\;\; p(n_1,\ldots,n_{i-1},n_i+1,n_{i+1},\ldots,n_k),$$

(ii) *weakly decreasing upon adding new subtrees,*

$$p(n_1,\ldots,n_i,n_{i+1},\ldots,n_k) \;\;\geq\;\; p(n_1,\ldots,n_i,1,n_{i+1},\ldots,n_k),$$

(iii) *and sub-multiplicative*

$$p(n_1,\ldots,n_i,n_{i+1},\ldots,n_k) \;\;\leq\;\; p(n_1,\ldots,n_i)\cdot p(n_{i+1},\ldots,n_k).$$

The sub-multiplicativity allows us to handle shared roots in micro trees.

Example J.2 (Uniform composition trees): A simple example of a monotonic fixed-size ordinal-tree source is obtained by setting

$$p(n_1,\ldots,n_k) \;\;=\;\; \frac{1}{2^{n_1+\cdots+n_k-2}} \;\;=\;\; 2^{-(n-2)}.$$

In a sense, this is the analog of random BSTs (Example E.1) in the world of ordinal trees. The distribution is very skewed to wide and short trees.

Example J.3 (Random LRM-trees / Uniform random recursive trees): Let $p(n_1,\ldots,n_k)=\prod_{j=1}^{k}\frac{1}{n_1+\cdots+n_j}$. It is easy to check that $\mathfrak{S}_{fs}(p)$ is a monotonic ordinal-tree source. Trees with this distribution arise in several interesing ways.

- They are the shape of **LRM-trees** [4] built on a random permutation; here, the children of the root are the indices of left-to-right minima (records) in the permutation, and the subtree is constructed recursively from the subpermutation following a left-to-right-minimum up to (excluding) the next one.
- They are also the shapes of (plane/ordered) **random recursive trees** which are grown inductively: when the ith node is added, it selects its parent uniformly among the $i-1$ existing nodes and becomes that node's leftmost child. This process is also called uniform attachment.
- The distribution is also obtained by applying the FCNS mapping to random BSTs; hence Lemma J.5 below provides another proof of monotonicity.

Let $\mathcal{S}_{fs}(p)$ be a fixed-size binary-tree source. The first-child next-sibling encoding fcns : $\mathfrak{T}\to\mathcal{T}$, defined in Definition B.2, transforms an ordinal tree $t\in\mathfrak{T}_n$ into a binary tree $\mathrm{fcns}(t)\in\mathcal{T}_n$. However, this mapping is not surjective onto $\mathcal{T}_n$: As the root node of an ordinal tree $t\in\mathfrak{T}_n$ does not have a next sibling, we find that the left subtree of $\mathrm{fcns}(t)\in\mathcal{T}_n$ is always of size $n-1$, whereas the right subtree is empty. In particular, $\mathcal{S}_{fs}(p)$ is not a probability distribution on $\mathrm{fcns}(\mathfrak{T}_n)$. Thus, for a given fixed-size binary-tree source $\mathcal{S}_{fs}(p)$, we define

$$\tilde{\mathbb{P}}_{\mathcal{S}}[t] \;\;=\;\; \prod_{\substack{v\in t\\ v\neq\rho}} p(|t_\ell[v]|,|t_r[v]|),$$

for binary trees $t \in \mathrm{fcns}(\mathcal{T}_n)$, where the product ranges over all nodes v of t except for the root node ρ. We then find that $\tilde{\mathbb{P}}_{\mathcal{S}} : \mathrm{fcns}(\mathfrak{T}_n) \to [0,1]$ is a probability distribution. Moreover, we define:

Definition J.4 (FCNS source): *Let $\mathcal{S}$ be a fixed-size binary-tree source. By $\mathfrak{S}_{fcns}(\mathcal{S})$ we denote the ordinal tree source that yields $\mathbb{P}_{\mathfrak{S}}[t] = \tilde{\mathbb{P}}_{\mathcal{S}}[\mathrm{fcns}(t)]$ for every $t \in \mathfrak{T}_n$.*

That is, in order to generate a random tree in $\mathfrak{T}_n$, we can let $\mathcal{S}$ generate a binary tree $t' \in \mathcal{T}_{n-1}$ with probability $\mathbb{P}_{\mathcal{S}}[t']$, then add a new root node to t' in order to obtain a tree t'', such that t' is the left subtree of t'', and compute $t = \mathrm{fcns}^{-1}(t') \in \mathfrak{T}_n$. We find that $\mathbb{P}_{\mathcal{S}}[t'] = \tilde{\mathbb{P}}_{\mathcal{S}}[t'']$.

Lemma J.5 (FCNS preserves monotonicity): *Let $\mathcal{S}_{fs}(p)$ be a monotonic fixed-size binary tree source. Then, $\mathfrak{S}_{fcns}(\mathcal{S}_{fs}(p))$ is a monotonic fixed-size ordinal-tree source.*

Proof: We show that $\mathfrak{S}_{fcsn}(\mathcal{S}_{fs}(p))$ can be written as $\mathfrak{S}_{fs}(p')$ for a p' that fulfills the conditons of Definition J.1. By definition of fcns, we have

$$\begin{aligned} p'(n_1,\ldots,n_k) &= p(n_1-1, n_2+\cdots+n_k)\cdot p(n_2-1, n_3+\cdots+n_k)\cdot\cdots \\ &\quad \cdot p(n_{k-1}-1, n_k)\cdot p(n_k-1, 0). \end{aligned}$$

The monotonicity conditions follow by directly from monotonicity of p. $\square$

Lemma J.6 (monotonicity implies submultiplicativity): *Let $\mathfrak{S}_{fs}(p)$ be monotonic and $t \in \mathfrak{T}$ be decomposed into micro trees $\mu_1,\ldots,\mu_m$. Then $\mathbb{P}[t] \le \prod_{i=1}^{m} \mathbb{P}[\mu_i]$.*

Proof: Let v be a node of t with children $u_1,\ldots,u_k$ and let μ_i be a micro tree that v belongs to. As μ_i is a subtree of t, we find $|\mu_i[u_j]| \le |t[u_j]|$. Note that μ_i might contain only some of the nodes u_j; if a node u_j does not belong to μ_i, we define $\mu_i[u_j] = \Lambda$ and hence $|\mu_i[u_j]| = 0$. There are 3 cases for v:

1. v occurs in only one micro tree μ_i.
   Then, its contribution to $\mathbb{P}[t]$ satisfies $p(|t[u_1]|,\ldots,|t[u_k]|) \le p(|\mu_i[u_1],\ldots,|\mu_i[u_k]|)$ by monotonicity of the source.

2. v is a branching node.
   Assume $u_1,\ldots,u_k$ are spread over s micro trees $\mu_{i_1},\ldots,\mu_{i_s}$ that also contain v. Then, these micro trees each contain an interval of children (Fact B.7−(iv)), i.e., there are indices $1 \le l_1 \le r_1 \le l_2 \le r_2 \le \cdots \le l_s \le r_s \le k$ so that μ_{i_j} contains $u_{l_j},\ldots,u_{r_j}$. By monotonicity and since $p(\cdot) \le 1$, we have

$$\begin{aligned} p(|t[u_1]|,\ldots,|t[u_k]|) &\le \prod_{j=1}^{s} p(|t[u_{l_j}]|,\ldots,|t[u_{r_j}]|) \\ &\le \prod_{j=1}^{s} p(|\mu_{i_j}[u_{l_j}]|,\ldots,|\mu_{i_j}[u_{r_j}]|). \end{aligned}$$

3. v is a path node.
   As above, $u_1,\ldots,u_k$ will be spread over s micro trees $\mu_{i_1},\ldots,\mu_{i_s}$ that also contain v, but one of them, μ_{i_h} can be missing a child from its interval (Fact B.7−(v)). With indices as

above, μ_{i_j}, $j \neq h$, contains $u_{l_j}, \ldots, u_{r_j}$, and μ_{i_h} contains $u_{l_h}, \ldots, u_{q-1}, u_{q+1}, \ldots, u_{r_h}$ for a $q \in [k]$. We obtain by monotonicity

$$\begin{array}{rl} & p(|t[u_{l_h}]|, \ldots, |t[u_{q-1}]|, |t[u_q]|, |t[u_{q+1}]|, \ldots, |t[u_{r_h}]|) \\ \leq & p(|t[u_{l_h}]|, \ldots, |t[u_{q-1}]|, \quad 1 \quad, |t[u_{q+1}]|, \ldots, |t[u_{r_h}]|) \\ \leq & p(|t[u_{l_h}]|, \ldots, |t[u_{q-1}]|, \qquad\quad |t[u_{q+1}]|, \ldots, |t[u_{r_h}]|); \end{array}$$

and hence

$$\begin{array}{rcl} p(|t[u_1]|, \ldots, |t[u_k]|) & \leq & \displaystyle\prod_{j=1}^{s} p(|t[u_{l_j}]|, \ldots, |t[u_{r_j}]|) \\ & \leq & p(|t[u_{l_h}]|, \ldots, |t[u_{q-1}]|, |t[u_{q+1}]|, \ldots, |t[u_{r_h}]|) \cdot \\ & & \displaystyle\prod_{\substack{j=1,\ldots,s \\ j \neq h}} p(|t[u_{l_j}]|, \ldots, |t[u_{r_j}]|) \\ & \leq & p(|\mu_{i_h}[u_{l_h}]|, \ldots, |\mu_{i_h}[u_{q-1}]|, |\mu_{i_h}[u_{q+1}]|, \ldots, |\mu_{i_h}[u_{r_h}]|) \cdot \\ & & \displaystyle\prod_{\substack{j=1,\ldots,s \\ j \neq h}} p(|\mu_{i_j}[u_{l_j}]|, \ldots, |\mu_{i_j}[u_{r_j}]|). \end{array}$$

In all three cases we could bound the contribution of v to $\mathbb{P}[t]$ by the product of its contributions to the micro trees it belongs to. Therefore we find

$$\begin{array}{rcl} \mathbb{P}[t] & = & \displaystyle\prod_{v \in t} p(|t_1[v]|, \ldots, |t_{\deg_t(v)}[v]|) \\ & \leq & \displaystyle\prod_{i=1}^{m} \prod_{v \in \mu_i} p(|(\mu_i)_1[v]|, \ldots, |(\mu_i)_{\deg_{\mu_i}(v)}[v]|) \\ & = & \displaystyle\prod_{i=1}^{m} \mathbb{P}[\mu_i]. \end{array} \qquad \square$$

J.1.1. Universality of Monotonic Fixed-Size Ordinal Tree Sources

In order to show universality of our hypersuccinct code for ordinal trees from Section H.1 with respect to fixed-size ordinal tree sources, we start again with a source-specific encoding for ordinal trees: As for binary trees, we define a depth-first order arithmetic code D_p for ordinal trees, dependent on a given ordinal tree source $\mathfrak{S}_{fs}(p)$. Let $t \in \mathfrak{T}$ denote an ordinal tree with $\mathbb{P}[t] > 0$. Assuming that the fixed-size source p need not be stored as part of the encoding, we again make use of arithmetic coding in order to store t's subtree sizes: Recall that the function p assigns a probability to each possible grouping of the $n-1$ descendants of a tree of size n into subtrees, and that there are 2^{n-2} many choices for these groupings: the compositions of $n-1$ into positive integers. Fix an enumeration of these compositions for every n, such that if we know n, every number $\ell \in \{1, \ldots, 2^{n-2}\}$ represents one of these possible groupings.

The depth-first arithmetic code D_p now stores an ordinal tree t as follows: We initially encode the size of the tree in Elias gamma code: If the tree consists of n nodes, we store the Elias gamma code of $n+1$, $\gamma(n+1)$, in order to take the case into account that t is the empty binary tree. Additionally, while traversing the tree in depth-first order, we encode the grouping of the $|t[v]| - 1$ many descendants of v into subtrees for every node v using arithmetic coding: To encode these subtree sizes, we feed the arithmetic coder with the model that the next symbol is a number $\ell \in \{0, \ldots, 2^{|t[v]|-1}\}$, representing a composition $(|t[v_1]|, \ldots, |t[v_k]|)$ of $|t[v]| - 1$ by our fixed enumeration of all compositions of $|t[v]| - 1$, with probability $p(|t[v_1]|, \ldots, |t[v_k]|)$. We can reconstruct the tree t recursively from its code $D_p(t)$, as we always know the subtree size of the current node. This yields an encoding D_p which stores an ordinal tree t with $\mathbb{P}[t] > 0$ in

$$|D_p(t)| \quad \leq \quad \lg\left(\frac{1}{\mathbb{P}[t]}\right) + 2\lfloor \lg(|t| + 1) \rfloor + 3 \tag{16}$$

many bits.

Lemma J.7 (micro tree code): *Let $\mathfrak{S}_{fs}(p)$ be a fixed-size tree source and let $t \in \mathfrak{T}_n$ with $\mathbb{P}[t] > 0$. If $\mathfrak{S}_{fs}(p)$ is monotonic, then*

$$\sum_{i=1}^{m} |C(\mu_i)| \quad \leq \quad \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{n \log \log n}{\log n}\right),$$

where C is a Huffman code for the sequence of micro trees $\mu_1, \ldots, \mu_m$ from our tree covering scheme (see Section H.1).

Proof: As $\mathfrak{S}_{fs}(p)$ is monotonic, we have $0 < \mathbb{P}[t] \leq \mathbb{P}[\mu_i]$ by Lemma J.6 for every $i \in [m]$: Thus, $|D_p(\mu_i)|$ is well-defined for every micro tree μ_i. By optimality of Huffman codes, we find that

$$\sum_{i=1}^{m} |C(\mu_i)| \leq \sum_{i=1}^{m} |D_p(\mu_i)|,$$

where D_p is the depth-first arithmetic code for ordinal tree sources. By our estimate (16) for $|D_p|$, we find that

$$\begin{aligned} \sum_{i=1}^{m} |D_p(\mu_i)| &\leq \sum_{i=1}^{m} \left(\lg\left(\frac{1}{\mathbb{P}[\mu_i]}\right) + 3 + 2\lfloor \lg(|\mu_i| + 1) \rfloor \right) \\ &\leq \sum_{i=1}^{m} \lg\left(\frac{1}{\mathbb{P}[\mu_i]}\right) + O(m \log \mu). \end{aligned}$$

As $\mathfrak{S}_{fs}(p)$ is monotonic, we find by Lemma J.6:

$$\sum_{i=1}^{m} \lg\left(\frac{1}{\mathbb{P}[\mu_i]}\right) + O(m \log \mu) \leq \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O(m \log \mu).$$

Altogether, with $m = \Theta(n/\log n)$ and $\mu = \Theta(\log n)$ (see Section H.1), we thus obtain

$$\sum_{i=1}^{m} |C(\mu_i)| \leq \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{n \log \log n}{\log n}\right). \qquad \square$$

From Lemma J.7 and Lemma H.2, we find the following:

Theorem J.8 (Universality for monotonic sources): *Let $\mathfrak{S}_{fs}(p)$ be a monotonic fixed-size tree source. The hypersuccinct code $\mathsf{H} : \mathfrak{T} \to [0,1]$ satisfies*

$$|\mathsf{H}(t)| \le \lg\left(\frac{1}{\mathbb{P}[t]}\right) + O\left(\frac{n \log\log n}{\log n}\right)$$

for every $t \in \mathfrak{T}_n$ with $\mathbb{P}[t] > 0$.

As the ordinal tree sources from Example J.2 and Example J.3 are both monotonic, we obtain the following corollary from Theorem J.8:

Corollary J.9: *The hypersuccinct code $\mathsf{H} : \mathfrak{T} \to \{0,1\}^\star$ encodes*

(i) ***Uniform composition trees*** *of size n (see Example J.2) using*

$$|\mathsf{H}(t)| \le \lg(1/\mathbb{P}[t]) + O(n \log\log n / \log n)$$

many bits,

(ii) ***Random LRM trees*** *of size n (see Example J.3) using*

$$|\mathsf{H}(t)| \le \lg(1/\mathbb{P}[t]) + O(n \log\log n / \log n)$$

many bits.

J.2. Fringe-Dominated Fixed-Size Ordinal Tree Sources

As for binary trees, we consider a second class of fixed-size sources, fringe-dominated ordinal tree sources, for which we will be able to prove universality of the hypersuccinct code: Recall that a node v is called *heavy*, if $|t[v]| \ge B$ for the fixed parameter B, and *light*, otherwise. With $n_{\ge B}(t)$ we again denote the number of heavy nodes of t. Moreover, we call a fringe subtree *heavy*, if its root is heavy, and *light* otherwise. With $\ell_B(t)$, we denote the total number of *maximal* (non-empty) light fringe subtrees of t, i.e., of light nodes v of t, such that parent(v) is heavy. Note that for binary trees, we have $\ell_B(t) \le n_{\ge B}(t) + 1$, as the set of heavy nodes of a binary tree t induces a (binary, non-fringe) subtree t' of t, and every leaf of this subtree t' of t can have at most two children. For ordinal trees, this relation does not hold (consider, for example, an ordinal tree of size n consisting of a root node with $n-1$ children).

Definition J.10 (Average-case fringe-dominated): *We call a fixed-size ordinal tree source* average-case B-fringe-dominated, *for a function B with $B(n) = \Theta(\log n)$, if*

$$\sum_{t \in \mathfrak{T}_n} \mathbb{P}[t] \cdot \ell_B(t) = o\left(\frac{n}{\log B}\right) \quad \text{and} \quad \sum_{t \in \mathfrak{T}_n} \mathbb{P}[t] \cdot n_{\ge B}(t) = o\left(\frac{n}{\log B}\right).$$

Definition J.11 (Worst-case fringe-dominated): *We call a fixed-size ordinal tree source* worst-case B-fringe-dominated, *for a function B with $B(n) = \Theta(\log n)$, if*

$$\ell_B(t) = o(n/\log B) \quad \text{and} \quad n_{\ge B}(t) = o(n/\log B)$$

for every $t \in \mathfrak{T}_n$ with $\mathbb{P}[t] > 0$.

Note that for binary trees, these definitions accord with Definition E.9 and Definition E.10 of fringe-dominated binary tree sources, as in this case $\ell_B(t) \le n_{\ge B}(t) + 1$, by the above considerations. The parameter B will again be chosen as $B = \Theta(\log n)$.

Fringe-dominated sources can be handled similarly as binary trees using a great-branching code. We start with the following lemma:

Lemma J.12 (micro tree code): *Let $\mathfrak{S}_{fs}(p)$ be a fixed-size tree source and let $t \in \mathfrak{T}_n$ with $\mathbb{P}[t] > 0$. Then*

$$\sum_{i=1}^{m} |C(\mu_i)| \;\le\; \lg\biggl(\frac{1}{\mathbb{P}[t]}\biggr) \;+\; O(n \log\log n/\log n) \;+\; O(\ell_{B(n)}(t)\log\log n) \\ + O(n_{\ge B}(t)\log\log n),$$

where C is a Huffman code for the sequence of micro trees $\mu_1, \ldots, \mu_m$ and $B(n) \in \Theta(\log n)$ is the parameter from our tree covering scheme (see Section H.1).

Proof: As in the case of binary trees, we first observe that some of the micro trees $\mu_1, \ldots, \mu_m$ from the tree covering scheme might be fringe, but many will be internal micro trees, i.e., have child micro trees in the top tier tree Υ. Let $\mathcal{I}_0 = \{i \in [m] \mid \mu_i \text{ is fringe}\}$ and let $\mathcal{I}_1 = [m] \setminus \mathcal{I}_0$. If μ_i is a fringe micro tree, then all micro-tree local subtree sizes and node degrees coincide with the corresponding global subtree sizes and node degrees, except for (possibly) the root node's degree: The root node of μ_i might be contained in several micro trees, in that case its global degree and its micro-tree local node degree do not coincide (however, the respective subtree sizes do). Let ρ_i denote the root node of micro tree μ_i and let $f_{i,1}, \ldots, f_{i,\deg(\rho_i)}$ denote the fringe subtrees of μ_i rooted in ρ_i's children, listed in preorder. By definition of the tree covering scheme (Section B.3), we find that all the subtrees $f_{i,1}, \ldots, f_{i,\deg(\rho_i)}$ are maximal light subtrees of t, and ρ_i corresponds to a heavy node of t.

If μ_i is an internal micro tree, then its root node might be contained in several micro trees as well, resulting in different global and micro-tree local node degrees. Furthermore, the subtree sizes of the ancestors of portal nodes change. By Lemma H.1, there is at most one other edge leading to a node outside of the micro tree μ_i apart from edges leaving the subtree root: Thus, the ancestors of portals in an internal micro tree μ_i form a *unary path* from the root node to the (non-root-node) portal, if it exists. Let $bough(\mu_i)$ denote the subtree of μ_i induced by the set of nodes that are ancestors of μ_i's child micro trees (ancestors of the portals), including the root node. As observed above, $bough(\mu_i)$ is always a unary path – thus, if we know the length of $bough(\mu_i)$, we also know its shape. With $g_{i,1,1}, \ldots, g_{i,1,k_{i,1}} \cdots g_{i,|bough(\mu_i)|,1}, \ldots, g_{i,|bough(\mu_i)|,k_{i,|bough(\mu_i)|}}$ we denote the non-empty fringe subtrees of μ_i hanging off the boughs of μ_i, where $g_{i,j,1}, \ldots, g_{i,j,k_{i,j}}$ denote the fringe subtrees attached to the jth node of $bough(\mu_i)$ (listed in preorder), and $k_{i,j}$ denotes their respective number. Moreover, with $r_{i,j}$ we denote how many of them are right siblings of the $(j+1)$st node of $bough(\mu_i)$ (if $j = |bough(\mu_i)|$, we set $r_{i,|bough(\mu_i)|} = 0$). As those fringe subtrees $g_{i,j,k}$ of μ_i are fringe subtrees of t as well and pairwise-disjoint, we find that their micro-tree local subtree sizes and micro-tree local node degrees coincide with the corresponding global subtree sizes and global node degrees. Altogether, we thus have

$$\sum_{i\in\mathcal{I}_0} \sum_{k=1}^{\deg(\rho_i)} \lg\biggl(\frac{1}{\mathbb{P}[f_{i,k}]}\biggr) + \sum_{i\in\mathcal{I}_1} \sum_{j=1}^{|bough(\mu_i)|} \sum_{k=1}^{k_{i,j}} \lg\biggl(\frac{1}{\mathbb{P}[g_{i,j,k}]}\biggr) \;\le\; \lg\biggl(\frac{1}{\mathbb{P}[t]}\biggr). \tag{17}$$

Moreover, the fringe subtrees $g_{i,j,k}$ are maximal light subtrees of t, as by definition of the tree covering scheme, *bough*-nodes are heavy.

As in the proof of Lemma E.24, we now construct a new encoding, similar to the "great-branching" code, for ordinal trees: Let

$$E_{i,j} \;=\; \gamma(k_{i,j}+1)\cdot\gamma(r_{i,j}+1)\cdot D_p(g_{i,j,1})\dots D_p(g_{i,j,k_{i,j}}) \;\in\; \{0,1\}^\star,$$

where $D_p(g_{i,k})$ denotes the depth-first order arithmetic code for ordinal fixed-size tree sources from Section J.1. That is, $E_{i,j}$ stores the number of fringe subtrees attached to the jth node of $\mathit{bough}(\mu_i)$, followed by the number $r_{i,j}$ which states how many of them are right siblings of the $j+1$st node of $\mathit{bough}(\mu_i)$, followed by their depth-first order arithmetic codes, listed in preorder. We set

$$\hat{G}_B(\mu_i) = \begin{cases} \mathtt{0}\cdot\gamma(\deg(\rho_i))\cdot D_p(f_{i,1})\cdots D_p(f_{i,\deg(\rho_i)}), & \text{if } \mu_i \text{ is a fringe micro tree;}\\ \mathtt{1}\cdot\gamma(|\mathit{bough}(\mu_i)|)\cdot E_{i,1}\cdots E_{i,|\mathit{bough}(\mu_i)|}, & \text{otherwise,}\end{cases}$$

Note that this is well-defined, as the encoding D_p is only applied to fringe subtrees $f_{i,k}$ and $g_{i,j,k}$ of t, for which $\mathbb{P}[f_{i,k}], \mathbb{P}[g_{i,j,k}] > 0$ follows from $\mathbb{P}[t] > 0$. We can reconstruct μ_i from $\hat{G}_B(\mu_i)$ as follows: If μ_i is a fringe subtree, we know the degree of the root of μ_i, followed by the (uniquely decodable) encodings of the root node's subtrees, $D_p(f_{i,1}),\dots,,D_p(f_{i,\deg(\rho_i)})$. If μ_i is an internal micro tree, we first decode the size (and thus, the shape) of $\mathit{bough}(\mu_i)$. Then, for each node of $\mathit{bough}(\mu_i)$, we decode the number of fringe subtrees (which can be zero) attached to that node, followed by how many of them are right siblings of the next bough-node, followed by their depth-first order arithmetic code, which tells us their sizes and shapes, listed in preorder. The code $\hat{G}_B$ is *not* a prefix-free code over Σ_μ: there can be micro tree shapes that are assigned *several* codewords by $\hat{G}_B$, depending on which nodes are portals to other micro trees (if any). But $\hat{G}_B$ is uniquely decodable to local shapes of micro trees, and can thus be seen as a *generalized prefix-free code*, where more than one codeword per symbol is allowed: Thus, the Huffman code C for micro trees used in the hypersuccinct code achieves no worse encoding length than the great-branching code $\hat{G}_B$:

$$\sum_{i=1}^{m}|C(\mu_i)| \le \sum_{i=1}^{m}|\hat{G}_B(\mu_i)| = \sum_{i\in\mathcal{I}_0}|\hat{G}_B(\mu_i)| + \sum_{i\in\mathcal{I}_1}|\hat{G}_B(\mu_i)|.$$

By definition of $\hat{G}_B$ and $E_{i,j}$, we find

$$\begin{aligned}\sum_{i\in\mathcal{I}_0}|\hat{G}_B(\mu_i)| + \sum_{i\in\mathcal{I}_1}|\hat{G}_B(\mu_i)| &= \sum_{i\in\mathcal{I}_0}\left(1+|\gamma(\deg(\rho_i))| + \sum_{k=1}^{\deg(\rho_i)}|D_p(f_{i,k})|\right)\\ &+ \sum_{i\in\mathcal{I}_1}\left(1+|\gamma(|\mathit{bough}(\mu_i)|)| + \sum_{j=1}^{|\mathit{bough}(\mu_i)|}\left(|\gamma(k_{i,j}+1)| + |\gamma(r_{i,j}+1)| + \sum_{k=1}^{k_{i,j}}|D_p(g_{i,j,k})|\right)\right).\end{aligned}$$

With $\deg(\rho_i), k_{i,j}, r_{i,j} \le \mu - 1$, this is upper-bounded by

$$\begin{aligned}&\sum_{i\in\mathcal{I}_0}\left(2+2\lg(\mu) + \sum_{k=1}^{\deg(\rho_i)}|D_p(f_{i,k})|\right)\\ &+ \sum_{i\in\mathcal{I}_1}\left(2+2\lg(|\mathit{bough}(\mu_i)|) + |\mathit{bough}(\mu_i)|(4\lg(\mu)+2) + \sum_{j=1}^{|\mathit{bough}(\mu_i)|}\sum_{k=1}^{k_{i,j}}|D_p(g_{i,j,k})|\right).\end{aligned}$$

Using the estimate (16) for $|D_p|$, we can upper-bound this by

$$\sum_{i\in\mathcal{I}_0}\left(2+2\lg(\mu)+\sum_{k=1}^{\deg(\rho_i)}\left(\lg\left(\frac{1}{\mathbb{P}[f_{i,k}]}\right)+2\lg(|f_{i,k}|+1)+3\right)\right)$$
$$+\sum_{i\in\mathcal{I}_1}(2+2\lg(|\mathit{bough}(\mu_i)|)+|\mathit{bough}(\mu_i)|(4\lg(\mu)+2))$$
$$+\sum_{i\in\mathcal{I}_1}\sum_{j=1}^{|\mathit{bough}(\mu_i)|}\sum_{k=1}^{k_{i,j}}\left(\lg\left(\frac{1}{\mathbb{P}[g_{i,j,k}]}\right)+2\lg(|g_{i,j,k}|+1)+3\right).$$

With $|\mathcal{I}_0|\le m$ and by inequality (17), this is smaller than

$$\lg\left(\frac{1}{\mathbb{P}[t]}\right)+4m\lg(\mu)+5\sum_{i\in\mathcal{I}_0}\sum_{k=1}^{\deg(\rho_i)}\lg(|f_{i,k}|+1)+10\sum_{i\in\mathcal{I}_1}|\mathit{bough}(\mu_i)|\lg(\mu)$$
$$+5\sum_{i\in\mathcal{I}_1}\sum_{j=1}^{|\mathit{bough}(\mu_i)|}\sum_{k=1}^{k_{i,j}}\lg(|g_{i,j,k}|+1).$$

Recall that all fringe subtrees $f_{i,k}$ and $g_{i,j,k}$ are distinct maximal light subtrees of t. Thus, their total number is upper-bounded by the number $\ell_B(t)$ of maximal light fringe subtrees:

$$\sum_{i\in\mathcal{I}_0}\deg(\rho_i)+\sum_{i\in\mathcal{I}_1}\sum_{j=1}^{|\mathit{bough}(\mu_i)|}k_{i,j}\le\ell_B(t). \tag{18}$$

Furthermore, every *bough*-node is heavy. However, the paths $\mathit{bough}(\mu_i)$ are not necessarily disjoint subtrees of t, as possibly many micro tree root nodes correspond to the same node of t: Thus, at most one node per micro tree is counted multiple times if we add up the sizes of the boughs. We thus have

$$\sum_{i\in\mathcal{I}_1}|\mathit{bough}(\mu_i)|\le n_{\ge B}(t)+m. \tag{19}$$

With the bounds (18) and (19), and as $|f_{i,j}|,|g_{i,j,k}|\le\mu$, we find altogether:

$$\sum_{i=1}^{m}|C(\mu_i)|\le\lg\left(\frac{1}{\mathbb{P}[t]}\right)+O(m\log(\mu))+O(\ell_B(t)\log(\mu))+O(n_{\ge B}(t)\log(\mu)).$$

As $m=\Theta(n/\log n)$ and $\mu=\Theta(\log n)$ (see Section H.1), we have

$$\sum_{i=1}^{m}|C(\mu_i)|\le\lg\left(\frac{1}{\mathbb{P}[t]}\right)+O\left(\frac{n\log\log n}{\log n}\right)+O(\ell_B(t)\log\log n)+O(n_{\ge B}(t)\log\log n). \qquad\square$$

From Lemma J.12 and Lemma H.2, we find the following:

Theorem J.13 (Universality from fringe dominance): *Let $\mathfrak{S}_{fs}(p)$ be an average-case fringe-dominated fixed-size ordinal tree source. Then the hypersuccinct code $\mathsf{H}:\mathfrak{T}\to\{0,1\}^\star$ satisfies*

$$\sum_{t\in\mathfrak{T}_n}\mathbb{P}[t]|\mathsf{H}(t)|\quad\le\quad\sum_{t\in\mathfrak{T}_n}\mathbb{P}[t]\lg\left(\frac{1}{\mathbb{P}[t]}\right)+o(n).$$

Let $\mathfrak{S}_{fs}(p)$ be a worst-case fringe-dominated fixed-size ordinal tree source. Then the hypersuccinct code $\mathsf{H} : \mathfrak{T} \to \{0,1\}^\star$ satisfies

$$|\mathsf{H}(t)| \quad \le \quad \lg\left(\frac{1}{\mathbb{P}[t]}\right) + o(n)$$

for every ordinal tree $t \in \mathfrak{T}_n$ with $\mathbb{P}[t] > 0$.

Remark J.14 (Fixed-height sources?): Fixed-height sources for ordinal trees could in principle be handled similar to the fixed-size ones below; but unless node degrees are bounded, there are infinitely many ordinal trees of a given height, which makes the utility of such sources questionable (and would not satisfy to the filter definitions in [83]). We will therefore not explore this route.

K. Label-Shape Entropy

In [46] (see also [47]), another measure of empirical entropy for (node-labeled) ordinal trees was introduced that we denote with $\mathcal{H}^s_k$: In [47], this measure is referred to as *label-shape-entropy*, as it considers both labels and structure of the tree, but this notion of empirical entropy is also a suitable entropy measure for *unlabeled* ordinal trees. Since we do not consider labeled trees in this work, we refer to this notion of empirical entropy for trees as *shape-entropy* for short. In this section, we show that the length of our hypersuccinct code H for binary trees (see Section C.1) can be upper-bounded in terms of the kth-order shape entropy $\mathcal{H}^s_k$ of an *ordinal* tree (for suitable k), plus lower-order terms.

Remark K.1 (Relation to degree entropy): In [47], it is shown that kth order shape entropy $\mathcal{H}^s_k$ can be exponentially smaller than the degree entropy $H^{\deg}$ (see Definition I.1), but that a reverse statement cannot hold, that is, the following two statements are shown:

Lemma K.2 (Lemmas 4 and 5, [47]): *There exists a family of trees $(t_n)_{n\in\mathbb{N}}$, such that $|t_n| = \Theta(n)$, $H^{\deg}(t_n) = (2 - o(1))n$ and $\mathcal{H}^s_k(t_n) \le \lg(en)$.*

Theorem K.3 (Theorem 4, [47]): *For every ordinal tree $t \in \mathfrak{T}$ of size $|t| \ge 2$ and integer $k \ge 1$, we have $\mathcal{H}^s_k(t) \le 2H^{\deg}(t) + 2\lg(|t|) + 4$.*

We need some additional notation. We introduce two additional types of tree processes to apply our proof template for universality. We call them *shape-processes* (as considered before in [46]) and *childtype-processes*. The childtype-processes will allow us to write $\mathcal{H}^s_k(t)$ as $\lg(1/\mathbb{P}[t])$, where $\mathbb{P}[t]$ is the probability that a certain tree process (a childtype process) generates t, which then can be written as a product of contributions of the nodes of the tree.

Let $\mathcal{T}^\diamond$ denote the set of full binary trees, and let $\mathcal{T}^\diamond_n$ likewise denote the set of full binary trees of size n. Let v be a node of a full binary tree $t \in \mathcal{T}^\diamond$. We define the *shape-history* $h^s(v)$ of v inductively as follows: If v is the root node of t, we set $h^s(v) = \varepsilon$ (the empty string). If v is the left child of a node w of t, we set $h^s(v) = h^s(w)0$ and if v is a right child of a node w of t, we set $h^s(v) = h^s(w)1$. In other words, in order to obtain $h^s(v)$, we walk downwards in the tree from the root node to node v, and concatenate bits 0 and 1 for each edge we traverse, where a number 0 (resp., 1) states that we move on to a left (resp. right) child node. Morever, we define the *k-th order shape history* $h^s_k(v) \in \{0,1\}^k$ of a node v of a full binary tree $t \in \mathcal{T}^\diamond$ as the length-k-suffix of the string $0^k h^s(v)$, that is, if $|h^s(v)| \ge k$, we take the last k directions 0 and 1 on the path from the root to the node v, and if $|h^s(v)| < k$, we pad this too short history with 0's, in order to obtain a string of length k. (This accords with the definition in [46]: Several alternatives of how to define k-shape histories of nodes v for which $|h^s(v)| < k$ are discussed in the long version of [46]). Recall the definition of type(v) for a node v of a binary tree t from Section D: In particular, we find that type$(v) \in \{0, 2\}$ if v is a node of a full binary tree. For a string $z \in \{0,1\}^k$ and an integer $i \in \{0, 2\}$, we define m^t_z as the number of nodes v of t, for which $h^s_k(v) = z$, and $m^t_{z,i}$ as the number of nodes v of t, for which $h^s_k(v) = z$ and type$(v) = i$. A *kth order shape process* $\vartheta = (\vartheta_z)_{z\in\{0,1\}^k}$ is a tuple of probability distributions $\vartheta_z : \{0, 2\} \to [0, 1]$ (see [46]). A kth order shape process ϑ assigns a probability $\mathbb{P}_\vartheta(t)$ to a full binary tree $t \in \mathcal{T}^\diamond$ by

$$\mathbb{P}_\vartheta[t] = \prod_{v\in t} \vartheta_{h^s_k(v)}(\mathrm{type}(v)) = \prod_{z\in\{0,1\}^k} \prod_{i\in\{0,2\}} (\vartheta_z(i))^{m^t_{z,i}}. \tag{20}$$

A kth order shape process randomly generates a full binary tree as follows: In a top-down way, starting at the root node, we determine for each node v its type type$(v) \in \{0, 2\}$, where this decision depends on the k-shape-history $h^s_k(v)$: The probability that a node v is of type i is given by $\vartheta_{h^s_k(v)}(i)$. If $i = 0$, this node becomes a leaf and the process stops at this node. Otherwise, i.e., if $i = 2$, we attach a left and a right child to the node and continue the process at these child nodes. Note that this process might generate infinite trees with non-zero probability. In [46], the *kth order empirical shape entropy* of a full binary tree t is defined as follows:

Definition K.4 (Shape entropy for full binary trees, [46]): *Let $k \geq 0$ be an integer and let $t \in \mathcal{T}^\diamond$ be a full binary tree. The (unnormalized) kth-order shape entropy of t is defined as*

$$\mathcal{H}^s_k(t) \;=\; \sum_{z\in\{0,1\}^k} \sum_{i\in\{0,2\}} m^t_{z,i} \lg\left(\frac{m^t_z}{m^t_{z,i}}\right).$$

The corresponding normalized tree entropy is obtained by dividing by the tree size. Note that shape entropy for full binary trees was already considered in Remark D.11. For a full binary tree $t \in \mathcal{T}^\diamond$, we define the corresponding *empirical kth order shape process* as the shape process $(\vartheta^t_z)_{z\in\{0,1\}^k}$ with $\vartheta^t_z(i) = m^t_{z,i}/m^t_z$ for every $z \in \{0,1\}^k$ and $i \in \{0, 2\}$ (if $m^t_z = 0$, we simply set $\vartheta^t_z(0) = 1$). In particular, for the kth order empirical shape process $(\vartheta^t_z)_{z\in\{0,1\}^k}$ of a full binary tree $t \in \mathcal{T}^\diamond$, we find

$$\lg\left(\frac{1}{\mathbb{P}_{\vartheta^t}[t]}\right) = \sum_{z\in\{0,1\}^k} \sum_{i\in\{0,2\}} m^t_{z,i} \lg\left(\frac{1}{\vartheta^t_z(i)}\right) = \sum_{z\in\{0,1\}^k} \sum_{i\in\{0,2\}} m^t_{z,i} \lg\left(\frac{m^t_z}{m^t_{z,i}}\right) = \mathcal{H}^s_k(t). \tag{21}$$

Next, we define a *modified first-child next-sibling encoding* $\mathrm{fcns}^\diamond : \mathfrak{F} \to \mathcal{T}^\diamond$, which maps a forest to a *full* binary tree, as follows:

Definition K.5 (Modified fcns): *The* modified first-child next-sibling encoding $\mathrm{fcns}^\diamond : \mathfrak{F} \to \mathcal{T}^\diamond$ *is recursively defined by* $\mathrm{fcns}^\diamond(\varepsilon) = \bullet$ *for the empty forest ε, and*

$$\mathrm{fcns}^\diamond(\bullet(f)g) = \bullet(\mathrm{fcns}^\diamond(f), \mathrm{fcns}^\diamond(g))$$

for forests $f, g \in \mathfrak{F}$.

That is, the left child (resp. right child) of a node in $\mathrm{fcns}^{\diamond}(f)$ is its first child (resp. next sibling) in f or a newly-added leaf, if it does not exist. In particular, we find that $\mathrm{fcns}^{\diamond}(f)$ is always a full binary tree, and that $\mathrm{fcns}^{\diamond} : \mathfrak{F} \to \mathcal{T}^{\diamond}$ is a bijection. Moreover, we find that we obtain the modified first-child next-sibling encoding $\mathrm{fcns}^{\diamond}(f)$ from $\mathrm{fcns}(f)$ (as defined in Definition B.2) by adding a leaf to each null-pointer of $\mathrm{fcns}(f)$. Furthermore, we find that each node v of a forest $f \in \mathfrak{F}$ uniquely corresponds to an inner node of $\mathrm{fcns}^{\diamond}(f)$, which we denote with $\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v)$. The shape entropy of an ordinal tree is defined as the shape entropy of its corresponding modified first-child next-sibling encoding in [46]:

Definition K.6 (Shape entropy for ordinal trees, [46]): *Let $k \geq 0$ be an integer and let $t \in \mathfrak{T}$ be an ordinal tree. The (unnormalized) kth order shape entropy of t is defined as*

$$\mathcal{H}^s_k(t) = \mathcal{H}^s_k(\mathrm{fcns}^{\diamond}(t)).$$

For an inner node v of a full binary tree $t \in \mathcal{T}^{\diamond}$, we define its *childtype* as follows:

$$\mathrm{childtype}(v) = \begin{cases} 0 & \text{if } v\text{'s children are both leaves,} \\ 1 & \text{if only } v\text{'s left child is a leaf,} \\ 2 & \text{if only } v\text{'s right child is a leaf,} \\ 3 & \text{if } v\text{'s children are both inner nodes.} \end{cases}$$

Moreover, for a node v of a forest $f \in \mathfrak{F}$, we set $\mathrm{childtype}(v) = \mathrm{childtype}(\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v))$. In particular, we find:

Lemma K.7: *Let v be a node of a forest $f \in \mathfrak{F}$, then*

$$\mathrm{childtype}(v) = \begin{cases} 0 & \textit{if } v \textit{ is a leaf and does not have a next sibling,} \\ 1 & \textit{if } v \textit{ is a leaf and has a next sibling,} \\ 2 & \textit{if } v \textit{ is not a leaf and does not have a next sibling,} \\ 3 & \textit{if } v \textit{ is not a leaf and has a next sibling.} \end{cases}$$

The proof of Lemma K.7 follows immediately from Definition K.5 and the definition of the childtype-mapping. Furthermore, for a node v of a forest $f \in \mathfrak{F}$, we define the shape-history $h^s(v)$ as $h^s(\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v))$, i.e., as the shape-history of its corresponding node in $\mathrm{fcns}^{\diamond}(f)$. We find that if v is the root node of the first tree in (the sequence of trees) f, then $h^s(v) = \varepsilon$ (the empty string). Otherwise, if v is the first child of a node w of f, then $h^s(v) = h^s(w)0$ and if v is the next sibling of a node w of f, then $h^s(v) = h^s(w)1$. Note that basically, for a node v of a forest f, $h^s(v)$ represents the numbers of v's left siblings and of v's ancestors' left siblings in unary. Similarly, we define $h^s_k(v)$ as $h^s_k(\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v))$.

A *kth order childtype process* $\zeta = (n_\zeta, (\zeta_z)_{z \in \{0,1\}^k})$ is a tuple of probability distributions $\zeta_z : \{0,1,2,3\} \to [0,1]$ together with a number $n_\zeta \in [0,1]$. A kth order childtype process ζ assigns a probability $\mathbb{P}_\zeta$ to a full binary tree $t \in \mathcal{T}^{\diamond}$ by

$$\mathbb{P}_\zeta[t] = \begin{cases} 1 - n_\zeta & \text{if } |t| = 1, \\ n_\zeta \cdot \displaystyle\prod_{\substack{v \in t \\ v \text{ inner node of } t}} \zeta_{h^s_k(v)}(\mathrm{childtype}(v)) & \text{otherwise.} \end{cases} \tag{22}$$

A kth order childtype process randomly generates a full binary tree t as follows: With probability $1 - n_\zeta$, t consists of just one node. Otherwise, in a top-down way, starting at the root node, we determine for each node v its $\mathrm{childtype}(v) \in \{0, 1, 2, 3\}$, where this decision depends on the k-shape-history $h_k^s(v)$: The probability that a node v is of childtype i is given by $\zeta_{h_k^s(v)}(i)$. We add a left child and a right child to the node and if $i = 0$, we (implicitly) mark both of them as leaves, if $i = 1$, we mark the left child as a leaf, if $i = 2$, we mark the right child as a leaf and if $i = 3$, we do not mark the children as leaves. The process then continues at child nodes which are not marked as leaves. For a forest $f \in \mathfrak{F}$, we set

$$\mathbb{P}_\zeta[f] = \mathbb{P}_\zeta[\mathrm{fcns}^\diamond(f)]. \tag{23}$$

Thus, via the $\mathrm{fcns}^\diamond$-encoding, a kth order childtype process can be seen as a process randomly generating a forest f as follows: With probability $1 - n_\zeta$, the forest is empty. Otherwise, in a top-down left-to-right way, starting at the root node of the first tree in the forest, we determine for each node v its $\mathrm{childtype}(v) \in \{0, 1, 2, 3\}$ (i.e., whether this node has a first child and whether this node has a next sibling), where this decision depends on the k-shape-history $h_k^s(v)$: Note that as we generate f in a top-down left-to-right way, we always know $h_k^s(v)$ at every node we visit. If $\mathrm{childtype}(v) = 0$, the process stops at this node. If $\mathrm{childtype}(v) = 1$, then we add a new child node to v's parent (respectively, if v is a root node itself, we add a new tree of size one to the forest), if $\mathrm{childtype}(v) = 2$, we add a new child to v, and if $\mathrm{childtype}(v) = 3$, we add a new child to v and a new child to v's parent node. The process then continues at these newly added nodes. In particular, we find

Lemma K.8: *Let $t \in \mathfrak{T}$ be a non-empty ordinal tree, then*

$$\mathbb{P}_\zeta[t] = n_\zeta \cdot \prod_{v \in t} \zeta_{h_k^s(v)}(\mathrm{childtype}(v)).$$

Proof: We find by the definition of $\mathbb{P}_\zeta$ (see (22)), the definition of the k-shape-history and the definition of the mapping childtype:

$$\begin{aligned}
\mathbb{P}_\zeta[t] = \mathbb{P}_\zeta[\mathrm{fcns}^\diamond(t)] &= n_\zeta \cdot \prod_{\substack{v \in \mathrm{fcns}^\diamond(t) \\ v \text{ inner node}}} \zeta_{h_k^s(v)}(\mathrm{childtype}(v)) \\
&= n_\zeta \cdot \prod_{v \in t} \zeta_{h_k^s(\mathrm{id}^\diamond_{\mathrm{fcns}}(v))}(\mathrm{childtype}(\mathrm{id}^\diamond_{\mathrm{fcns}}(v))) = n_\zeta \cdot \prod_{v \in t} \zeta_{h_k^s(v)}(\mathrm{childtype}(v)). \qquad \square
\end{aligned}$$

Finally, we make the following definition:

Definition K.9: *Let $\vartheta = (\vartheta_z)_{z \in \{0,1\}^k}$ be a kth order shape process. We define the corresponding $k-1$st-order childtype process $\zeta^\vartheta = (n_\zeta^\vartheta, (\zeta_z^\vartheta)_{z \in \{0,1\}^{k-1}})$ by setting $n_{\zeta^\vartheta} = \vartheta_{0^k}(2)$ and*

$$\zeta_z^\vartheta(0) = \vartheta_{z0}(0) \cdot \vartheta_{z1}(0), \quad \zeta_z^\vartheta(1) = \vartheta_{z0}(0) \cdot \vartheta_{z1}(2),$$
$$\zeta_z^\vartheta(2) = \vartheta_{z0}(2) \cdot \vartheta_{z1}(0), \quad \zeta_z^\vartheta(3) = \vartheta_{z0}(2) \cdot \vartheta_{z1}(2),$$

for every $z \in \{0,1\}^{k-1}$.

It is easy to see that ζ_z^ϑ is well-defined. In particular, we find

Lemma K.10: *Let $t \in \mathcal{T}^\diamond$ be a full binary tree. Then*

$$\mathbb{P}_\vartheta[t] = \mathbb{P}_{\zeta^\vartheta}[t].$$

Proof: First, let $|t| = 1$: Then t consists of only one leaf node v of k-history 0^k, and thus, we have

$$\mathbb{P}_\vartheta[t] = \vartheta_{0^k}(0) = 1 - n_{\zeta^\vartheta} = \mathbb{P}_{\zeta^\vartheta}[t].$$

In the next part of the proof, assume that $|t| > 1$. Let $\tilde{m}^t_{z,i}$ denote the number of inner nodes of t with k-shape-history $z \in \{0,1\}^\star$ and of childtype $i \in \{0,1,2,3\}$, and recall that $m^t_{z,i}$ denotes the number of nodes of t with k-shape-history z and with $\mathrm{type}(v) = i \in \{0,2\}$. Let v be a node of t. First, we assume that $h^s_k(v) = z0$ for some $z \in \{0,1\}^{k-1}$ with $z \neq 0^{k-1}$ (thus, v is not the root node of t), and that v is a leaf: Then v's parent w is of $k-1$-shape-history z, and w's childtype is either 0 or 1. In particular, the correspondence between leaves v of t with k-shape-history $z0$ and inner nodes $w = \mathrm{parent}(v)$ of t with $k-1$-shape-history z and childtype 0 or 1 is bijective, as every node v with k-shape-history $z0$ is a left child of its parent node. We thus have

$$m^t_{z0,0} = \tilde{m}^t_{z,0} + \tilde{m}^t_{z,1}.$$

In a similar way, we find that inner nodes v of t with k-shape-history $z0$ for $z \neq 0^{k-1}$ correspond to inner nodes $w = \mathrm{parent}(v)$ of t with $k-1$-shape-history z and childtype $i \in \{2,3\}$: We find

$$m^t_{z0,2} = \tilde{m}^t_{z,2} + \tilde{m}^t_{z,3}.$$

Furthermore, we obtain the following relations in the same way:

$$m^t_{z1,0} = \tilde{m}^t_{z,0} + \tilde{m}^t_{z,2},$$
$$m^t_{z1,2} = \tilde{m}^t_{z,1} + \tilde{m}^t_{z,3},$$

for every $z \in \{0,1\}^k$. It remains to deal with nodes of k-shape-history $z = 0^k$: We find that every inner node v of t of k-shape-history 0^k uniquely corresponds to an inner node $w = \mathrm{parent}(v)$ of t of $k-1$-shape-history 0^{k-1} and childtype $i \in \{2,3\}$, except for the root node: We thus have

$$m^t_{0^k,2} - 1 = \tilde{m}^t_{0^{k-1},2} + \tilde{m}^t_{0^{k-1},3}.$$

Finally, every leaf v of t of k-shape-history 0^k uniquely corresponds to an inner node $w = \mathrm{parent}(v)$ of t of $k-1$-shape-history 0^{k-1} and childtype $i \in \{1,2\}$, as the root node is an inner node by assumption:

$$m^t_{0^k,0} = \tilde{m}^t_{0^{k-1},0} + \tilde{m}^t_{0^{k-1},1}.$$

Altogether, we thus have for trees t with $|t| > 1$:

$$\begin{aligned}
\mathbb{P}_\vartheta[t] &= \prod_{z\in\{0,1\}^k} \prod_{i\in\{0,2\}} (\vartheta_z(i))^{m^t_{z,i}} = (\vartheta_{0^k}(0))^{\tilde{m}^t_{0^{k-1},0}+\tilde{m}^t_{0^{k-1},1}} \cdot (\vartheta_{0^k}(2))^{\tilde{m}^t_{0^{k-1},2}+\tilde{m}^t_{0^{k-1},3}+1} \\
&\cdot \prod_{\substack{z\in\{0,1\}^{k-1}\\ z\neq 0^{k-1}}} (\vartheta_{z0}(0))^{\tilde{m}^t_{z,0}+\tilde{m}^t_{z,1}} \cdot (\vartheta_{z0}(2))^{\tilde{m}^t_{z,2}+\tilde{m}^t_{z,3}} \\
&\cdot \prod_{z\in\{0,1\}^{k-1}} (\vartheta_{z1}(0))^{\tilde{m}^t_{z,0}+\tilde{m}^t_{z,2}} \cdot (\vartheta_{z1}(2))^{\tilde{m}^t_{z,1}+\tilde{m}^t_{z,3}} \\
&= \vartheta_{0^k}(2) \cdot \prod_{z\in\{0,1\}^{k-1}} (\vartheta_{z0}(0)\cdot\vartheta_{z1}(0))^{\tilde{m}^t_{z,0}} \cdot \prod_{z\in\{0,1\}^{k-1}} (\vartheta_{z0}(0)\cdot\vartheta_{z1}(2))^{\tilde{m}^t_{z,1}} \\
&\cdot \prod_{z\in\{0,1\}^{k-1}} (\vartheta_{z0}(2)\cdot\vartheta_{z1}(0))^{\tilde{m}^t_{z,2}} \cdot \prod_{z\in\{0,1\}^{k-1}} (\vartheta_{z0}(2)\cdot\vartheta_{z1}(2))^{\tilde{m}^t_{z,3}} \\
&= n_\zeta \cdot \prod_{z\in\{0,1\}^{k-1}} \left(\zeta^\vartheta_z(0)\right)^{\tilde{m}^t_{z,0}} \cdot \prod_{z\in\{0,1\}^{k-1}} \left(\zeta^\vartheta_z(1)\right)^{\tilde{m}^t_{z,1}} \\
&\cdot \prod_{z\in\{0,1\}^{k-1}} \left(\zeta^\vartheta_z(2)\right)^{\tilde{m}^t_{z,2}} \cdot \prod_{z\in\{0,1\}^{k-1}} \left(\zeta^\vartheta_z(3)\right)^{\tilde{m}^t_{z,3}} = \mathbb{P}_{\zeta^\vartheta}[t].
\end{aligned}$$

This finishes the proof. □

Corollary K.11: *Let $t \in \mathfrak{T}$ be an ordinal tree, and let $\vartheta^t := \vartheta^{\mathrm{fcns}^\diamond(t)}$ denote the empirical shape process of its corresponding first-child next-sibling encoding. Then*

$$\mathcal{H}^s_k(t) = \lg\left(\frac{1}{\mathbb{P}_{\zeta^{\vartheta^t}}[t]}\right).$$

Proof: We have

$$\mathcal{H}^s_k(t) = \mathcal{H}^s_k(\mathrm{fcns}^\diamond(t)) = \lg\left(\frac{1}{\mathbb{P}_{\vartheta^t}[\mathrm{fcns}^\diamond(t)]}\right) = \lg\left(\frac{1}{\mathbb{P}_{\zeta^{\vartheta^t}}[\mathrm{fcns}^\diamond(t)]}\right) = \lg\left(\frac{1}{\mathbb{P}_{\zeta^{\vartheta^t}}[t]}\right),$$

where the first equality follows from Definition K.6, the second equality follows from the fact that ϑ^t is the empirical kth order shape-process of $\mathrm{fcns}^\diamond(t)$ (see (21)), the third equality follows from Lemma K.10 and the last equality follows from (23). □

In order to show that our hypersuccinct encoding from Section C.1 achieves the shape-entropy defined in [46] for ordinal trees, we start with defining a *source-specific* encoding (called depth-first order arithmetic code) with respect to a given kth order childtype process ζ, against which we will compare the hypersuccinct code: The formula for $\mathbb{P}_\zeta[t]$ from Lemma K.8 suggests a route for an (essentially) optimal source-specific encoding of any ordinal tree $t \in \mathfrak{T}$ with $\mathbb{P}_\zeta[t] > 0$, that, given a kth order childtype process ζ, spends $\lg(1/\mathbb{P}[t])$ (plus lower-order terms) many bits in order to encode an ordinal tree $t \in \mathfrak{T}$ with $\mathbb{P}_\zeta[t] > 0$: Such an encoding may spend $\lg\Big(1/\zeta_{h^s_k(v)}(\mathrm{childtype}(v))\Big)$ many bits per node v of t, plus $\lg(1/n_\zeta)$ many bits, if t is non-empty, respectively, $\lg(1/(1-n_\zeta))$ many bits, if t is the empty tree. (Note that as $\mathbb{P}_\zeta[t] > 0$ by assumption, we have $\zeta_{h^s_k(v)}(\mathrm{childtype}(v)) > 0$ for every node v of t.) Assuming that we know the childtype process $\zeta = (\zeta_z)_{z\in\{0,1\}^k}$, i.e., that we need not store it as part of the encoding, we can make use of *arithmetic coding* in order to devise a simple (source-dependent) encoding $\mathcal{D}_\zeta$, dependent on ζ, that stores an ordinal tree t as follows: First, we store a number $i \in \{1, 2\}$ which tells us whether t is empty ($i = 1$) or non-empty ($i = 2$) using arithmetic encoding, i.e., we feed the arithmetic coder with the model that the next symbol is a number $i \in \{1, 2\}$ with probability $1-n_\zeta$, respectively, n_ζ. Next, while traversing the tree in depth-first order, we encode $\mathrm{childtype}(v) \in \{0, 1, 2, 3\}$ for each node v of t that we pass, using arithmetic coding: To encode $\mathrm{childtype}(v)$ (i.e., whether v is a leaf or not and whether v has a next sibling or not, see Lemma K.7), we feed the arithmetic coder with the model that the next symbol is a number $i \in \{0, 1, 2, 3\}$ with probability $\zeta_{h_k^s(v)}(i)$. Note that we always know $h_k^s(v)$ at each node v we traverse: By definition, we have $h_k^s(v) = h_k^s(\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v))$. If v is the root node of t, then $\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v)$ is the root node of $\mathrm{fcns}^{\diamond}(t)$ and thus $h_k^s(v) = h_k^s(\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v)) = 0^k$. Otherwise, $h^s(\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v)) = h^s(\mathrm{id}^{\diamond}_{\mathrm{fcns}}(w))0$ or $h^s(\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v)) = h^s(\mathrm{id}^{\diamond}_{\mathrm{fcns}}(w))1$ for a node w of t, which is either v's left sibling or, if v is the first child of its parent node in t, v's parent in t, as $\mathrm{id}^{\diamond}_{\mathrm{fcns}}(w)$ is $\mathrm{id}^{\diamond}_{\mathrm{fcns}}(v)$'s parent. Thus, as we visit the nodes of t in depth-first order, we have already visited w and know $h_k^s(w)$, from which we can compute $h_k^s(v)$. Altogether, this yields a source dependent code $\mathcal{D}_\zeta(t)$, which we refer to as the *depth-first arithmetic code* with respect to the childtype-process ζ. Note that an ordinal tree t is always uniquely decodable from $\mathcal{D}_\zeta(t)$. As arithmetic coding uses at most $\lg\Big(1/\zeta_{h_k^s(v)}(\mathrm{childtype}(v))\Big)$ many bits per node v, plus $\lg(1/n_\zeta)$ many bits if t is non-empty, plus at most 2 bits of overhead, we find

$$|\mathcal{D}_\zeta(t)| \le \begin{cases} \displaystyle\sum_{v\in t} \lg\Big(1/\zeta_{h_k^s(v)}(\mathrm{childtype}(v))\Big) + \lg(1/n_\zeta) + 2 & \text{if } t \text{ is non-empty,} \\ \lg(1/(1-n_\zeta)) + 2 & \text{otherwise.} \end{cases}$$

We now start with the following lemma:

Lemma K.12: *Let $(\zeta_z)_{z\in\{0,1\}^k}$ be a kth order childtype process and let $t \in \mathfrak{T}$ be an ordinal tree of size n with $\mathbb{P}_\zeta[t] > 0$. Then*

$$\sum_{i=1}^{m} |C(\mu_i)| \le \lg\left(\frac{1}{\mathbb{P}_\zeta[t]}\right) + O\left(\frac{n\log\log n + kn}{\log n}\right),$$

where C is a Huffman code for the sequence of micro trees $\mu_1, \ldots, \mu_m$ obtained from the tree-covering scheme.

Proof: Recall that the micro trees $\mu_1, \ldots, \mu_m$ from our tree partitioning scheme for ordinal trees are pairwise disjoint except for (potentially) sharing a common subtree root and that apart from edges leaving the subtree root, at most one other edge leads to a node outside of the subtree (see Fact B.7). The probability $\mathbb{P}_\zeta[t]$ consists of the contributions $\zeta_{h_k^s(v)}(\mathrm{childtype}(v))$ for every node v of t. However, $\zeta_{h_k^s(v)}(\mathrm{childtype}(v))$ depends on the childtype and k-shape-history of each node v, and there might be nodes, for which childtype and k-shape-history differ in t and μ_i. For the sake of clarity, let $h_k^s(v, t)$ denote the k-shape history of a node v in t (and likewise $h_k^s(v, \mu_i)$ the k-shape history of a node v in a micro tree μ_i), and let $\mathrm{childtype}(v, t)$ (resp. $\mathrm{childtype}(v, \mu_i)$) denote the childtype of a node v in t (resp. μ_i). First, we investigate under which conditions it might occur that a node v of micro tree μ_i satisfies $h_k^s(v, t) \ne h_k^s(v, \mu_i)$ or $\mathrm{childtype}(v, t) \ne \mathrm{childtype}(v, \mu_i)$. We find:

(i) If v is the root node of a micro tree μ_i, then it might have left, respectively, right siblings in t, which it does not have in μ_i: Thus, its childtype and its k-shape-history might change.

(ii) If v is the first child of the root of μ_i, then it might have left siblings in t, which it does not have in μ_i. Thus, its k-shape-history changes. Furthermore, the k-shape-history of its close descendants and right siblings thus changes as well, i.e., the k-shape-history of the descendants of order less than k of $\mathrm{id}_{\mathrm{fcns}}^{\diamond}(v)$: However, if we know $h_k^s(v,t)$, we are able to recover $h_k^s(w,t)$ for all nodes w which are descendants, right siblings, or right siblings of descendants of v.

(iii) If v is the last child of the root of μ_i, then it might have right siblings in t, which it does not have in μ_i: Thus, its childtype might change.

(iv) The root node's children in μ_i are consecutive children of this node in t, except for possibly one child node x, which might be missing in μ_i (see Fact B.7). Thus, if v is the right sibling of x in t, its k-shape-history in μ_i might differ from its k-shape-history in t. Furthermore, the k-shape-histories of nodes corresponding to the descendants of order at most k of $\mathrm{id}_{\mathrm{fcns}}^{\diamond}(v)$ in $\mathrm{fcns}^{\diamond}(t)$ might change as well. Again, if we know $h_k^s(v,t)$, we are able to recover $h_k^s(w,t)$ of nodes w which correspond to descendants of $\mathrm{id}_{\mathrm{fcns}}^{\diamond}(v)$ in $\mathrm{fcns}^{\diamond}(t)$.

(v) There is at most one other edge which leads to a node outside of the micro tree μ_i, besides edges emanating from the root of μ_i (see Fact B.7). Let v be the node in μ_i, from which this other edge emanates: If v has only one child in t, then it does not have a child node in μ_i, and thus, its childtypes in t and μ_i do not coincide. Otherwise, the degree of v in t is greater than one and in particular, there might be a child node w of v, whose left sibling in t does not belong to μ_i. Thus, w's k-shape-history might change, as well as the k-shape-history of the nodes corresponding to the descendants of order less than k of $\mathrm{id}_{\mathrm{fcns}}^{\diamond}(w)$: Again, if we know $h_k^s(v,t)$, we are able to recover $h_k^s(w,t)$ of nodes w which correspond to descendants of $\mathrm{id}_{\mathrm{fcns}}^{\diamond}(v)$ in $\mathrm{fcns}^{\diamond}(t)$. Finally, there might be a child node u of v, which has a right sibling in t and which does not have a right sibling in μ_i; thus, its childtype changes.

By the above considerations, there can be several nodes v in μ_i for which $h_k^s(v,t) \neq h_k^s(v,\mu_i)$, however, we only need to know $h_k^s(v,t)$ for at most four of these nodes (see items (i), (ii), (iv) and (v)) in order to be able to determine the k-shape-history in t of all nodes of μ_i. Let ℓ_i denote the number of k-shape-histories we need to know in order to be able to determine $h_k^s(v,t)$ for all nodes v of μ_i. Furthermore, let j_i denote the number of nodes v of μ_i, for which $\mathrm{childtype}(v,t) \neq \mathrm{childtype}(v,\mu_i)$, where we always include the root node π_i of μ_i in this j_i many nodes (even if its childtypes in t and μ_i are identical). By the above considerations, we find that j_i is upper-bounded by four (see items (i), (iii) and (v)). Let $S_i \in \{0,1\}^{\star}$ denote the following binary string, obtained as the concatenation of

- an encoding of the number j_i using two bits,
- the preorder positions in μ_i of the j_i many nodes for which $\mathrm{childtype}(v,t) \neq \mathrm{childtype}(v,\mu_i)$ (plus the root node π_i of μ_i), encoded in Elias gamma code and listed in preorder,
- the encodings of the childtypes in t of these j_i nodes using two bits each, listed in preorder,
- the encodings of the childtypes in μ_i of these j_i nodes using two bits each, listed in preorder,
- an encoding of the number ℓ_i using two bits,

- the Elias gamma encodings of the preorder positions in μ_i of the ℓ_i many nodes from whose k-shape histories in t we are able to determine the k-shape history in t of all nodes of μ_i, listed in preorder,
- the k-shape-histories of these ℓ_i nodes, listed in preorder, using k bits each.

We find that $|S_i| \leq O(\log(\mu) + k)$. We define the following modification of the depth-first order arithmetic code $\mathcal{D}_\zeta$, which we denote with $\bar{\mathcal{D}}_\zeta$: The encoding $\bar{\mathcal{D}}_\zeta(\mu_i)$ consists of the string S_i followed by an encoding of childtype(v, t) for every node v of μ_i in depth-first order (preorder) of μ_i except for the root node π_i of μ_i, using arithmetic coding: The childtype of the root node π_i is already stored in S_i. We traverse the tree μ_i in depth-first order; to encode childtype(v, t), we feed the arithmetic coder with the model that the next symbol is a number $i \in \{0, 1, 2, 3\}$ with probability $\zeta_{h_k^s(v,t)}(i)$. Note that at each node v that we pass, we know $h_k^s(v, t)$ (either from S_i or as we are able to determine $h_k^s(v, t)$ from the k-shape-history of the node v's left sibling or parent) and we know both childtype(v, t) and childtype(v, μ_i) (either because childtype$(v, t) =$ childtype(v, μ_i) or because we have stored both childtype(v, t) and childtype(v, μ_i) explicitly in S_i). Altogether, this yields the encoding $\bar{\mathcal{D}}(\mu_i)$. Note that we leave out the $\lg(1/n_\zeta)$ many bits (used in the encoding $\mathcal{D}(\mu_i)$) which encode the number $i \in \{1, 2\}$ which tells us whether μ_i is empty or not (by definition, every micro tree μ_i of a non-empty tree t is non-empty). As we have $\zeta_{h_k^s(v,t)}(\text{childtype}(v, t)) > 0$ for every node v the encoding $\bar{\mathcal{D}}_\zeta(\mu_i)$ is well-defined. We find that

$$|\bar{\mathcal{D}}_\zeta(\mu_i)| \leq |S_i| + \sum_{\substack{v \in \mu_i \\ v \neq \pi_i}} \lg(1/\zeta_{h_k^s(v,t)}(\text{childtype}(v, t))) + 2.$$

Furthermore, note that we can uniquely recover a micro tree shape μ_i from the encoding $\bar{\mathcal{D}}_\zeta(\mu_i)$ and that formally, $\bar{\mathcal{D}}_\zeta$ is not a *prefix-free* code over Σ_μ, as as there can be micro tree shapes that are assigned several codewords by $\bar{\mathcal{D}}_\zeta$. But $\bar{\mathcal{D}}_\zeta$ can again be seen as a generalized prefix-free code, where more than one codeword per symbol is allowed, as $\bar{\mathcal{D}}_\zeta$ is uniquely decodable to local shapes of micro trees. Thus, as a Huffman code minimizes the encoding length over the class of generalized prefix-free codes, we find:

$$\sum_{i=1}^{m} |C(\mu_i)| \leq \sum_{i=1}^{m} |\bar{\mathcal{D}}_\zeta(\mu_i)| \leq \sum_{i=1}^{m} \left(|\mathcal{S}_i| + \sum_{\substack{v \in \mu_i \\ v \neq \pi_i}} \lg(1/\zeta_{h_k^s(v,t)}(\text{childtype}(v, t))) + 2 \right).$$

Recall that the micro trees μ_i are disjoint except for possibly sharing a common root node and that $|S_i| \leq O(\log \mu + k)$. Thus, we have

$$\sum_{i=1}^{m} |C(\mu_i)| \leq \sum_{v \in t} \lg(1/\zeta_{h_k^s(v,t)}(\text{childtype}(v, t))) + O(m \log \mu + mk).$$

With $m = \Theta(n/\log n)$ and $\mu = \Theta(\log n)$ (see Section H.1), we have

$$\sum_{i=1}^{m} |C(\mu_i)| \leq \log\left(\frac{1}{\mathbb{P}_\zeta[t]}\right) + O\left(\frac{n \log \log n + kn}{\log n}\right).$$

This finishes the proof. $\square$

From Lemma H.2, Lemma K.12 and Corollary K.11, we now find the following:

Corollary K.13: *The hypersuccinct code* $\mathsf{H} : \mathfrak{T} \to \{0,1\}^\star$ *satisfies*

$$|\mathsf{H}(t)| \quad \leq \quad \mathcal{H}_k^s(t) + O\left(\frac{n \log\log n + kn}{\log n}\right)$$

for every ordinal tree $t \in \mathfrak{T}$ *of size* n.

It remains to remark that the above result from Corollary K.13 requires $k \in o(\log n)$ in order to be non-trivial: This bound on k also occurs in [46].

L. Notation Index

We collect used notation here for reference.

L.1. Elementary Notation

$\mathbb{N}$, $\mathbb{N}_0$ . . . . natural numbers without 0 (resp., with 0), $\mathbb{N} = \{1, 2, \ldots\}$, $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$

$\ln(n)$, $\lg(n)$ . . . . natural and binary logarithm; $\ln(n) = \log_e(n)$, $\lg(n) = \log_2(n)$.

$[m..n]$, $[n]$ . . . . integer intervals, $[k..n] = \{k, k+1, \ldots, n\}$; $[n] = [1..n]$.

$O(f(n))$, Ω, Θ, $\sim$ . . . . asymptotic notation as defined, e.g., in [22, § A.2]; in particular, $f \sim g$ means $f = g(1 + o(1))$; $f = g \pm O(h)$ is equivalent to $|f - g| \in O(|h|)$.

$x \pm y$ . . . . x with absolute error $|y|$; formally the interval $x \pm y = [x - |y|, x + |y|]$; as with O-terms, we use "one-way equalities": $z = x \pm y$ instead of $z \in x \pm y$.

L.2. Tree Notation

$\mathcal{T}_n$, $\mathcal{T}$ . . . . set of binary tree over n nodes, $\mathcal{T} = \bigcup_{n \geq 0} \mathcal{T}_n$

$\mathcal{T}^h$ . . . . set of binary tree of height h

$\mathfrak{T}_n$, $\mathfrak{T}$ . . . . set of ordinal tree over n nodes, $\mathfrak{T} = \bigcup_{n \geq 0} \mathfrak{T}_n$

$\mathfrak{T}^h$ . . . . set of ordinal trees of height h

$\mathfrak{F}$ . . . . the set of all forests, i.e., (possibly empty) sequences of trees from $\mathfrak{T}$

Λ . . . . the empty tree "null"

$v \in t$ . . . . v is a node in tree t; unless indicated otherwise, we identify nodes with their preorder rank

$|t|$ . . . . number of nodes in t, i.e., $t \in \mathcal{T}_n$ or $t \in \mathfrak{T}_n$ implies $|t| = n$

$h(t)$ . . . . height of the tree t

$\mathrm{type}(v)$ . . . . type of a node of a binary tree (leaf, left-unary, right-unary or binary)

$\deg(v)$ . . . . degree of v, i.e., the number of children of v

$t[v]$ . . . . subtree of t rooted at v; if v does not occur in t, $t[v] = \Lambda$

$t_\ell[v]$, $t_r[v]$ . . . . left resp. right subtree of $v \in t \in \mathcal{T}$

t_ℓ, t_r . . . . left resp. right subtree of the root of tree $t \in \mathcal{T}$

$t_k[v]$ . . . . kth subtree of $v \in t \in \mathfrak{T}$, for $k \in [\deg(v)]$

$BP(t)$ . . . . balanced parenthesis encoding of the binary tree $t \in \mathcal{T}$, see Definition B.1

$BP_o(t)$ . . . . balanced parenthesis encoding of the ordinal tree $t \in \mathfrak{T}$, see Definition B.1

$\mathrm{fcns}(t)$ . . . . the first-child next-sibling encoding of the binary tree t, see Definition B.2

$h(v)$, $h_k(v)$ . . . . (k-) history of a node v of a binary tree: string consisting of the node types of v's (k closest) ancestors

n_z^t . . . . number of nodes of t with k-history z

$n_{z,i}^t$ . . . . number of nodes of t with k-history z and type i

ν_i^t . . . . number of nodes of degree i of t

$n_b(t), n_{\geq b}(t)$ . . . . number of nodes of t with $|t[v]| = b$, resp. $|t[v]| \geq b$

`height` etc . . . . operations on trees; see Table 1 and Table 6

L.3. Tree Covering

B . . . . parameter of micro tree size, $B = \lceil \frac{1}{8} \lg n \rceil$

μ . . . . $\mu = \frac{1}{4} \lg n$ maximal micro tree size

m . . . . number of micro trees, $m = \Theta(n/B)$

$\mu_1, \ldots, \mu_m$ . . . . micro trees in preorder of their roots, with ties broken by next node in micro tree

Υ . . . . top tier tree, obtained by contracting each micro tree into a single node

Σ_μ . . . . set consisting of (the different shapes of) micro trees $\mu_1, \ldots, \mu_m$

L.4. Tree Sources

T_n . . . . a random tree of size n, i.e., a random variable taking values in $\mathcal{T}_n$ or $\mathfrak{T}_n$ with respect to some probability distribution

$\tau = (\tau_z)_{z \in \{1,2,3\}^k}$ . . . . . a kth-order type process, see Section D

$H_k^{\text{type}}(t)$ . . . . kth-order empirical type entropy of a binary tree t, see Definition D.1

$d = (d_i)_{i \in \mathbb{N}_0}$ . . . . a degree distribution, see Section I

$H^{\deg}(t)$ . . . . the degree entropy of an ordinal tree t, see Definition I.1

$\mathcal{S}_{fs}(p)$ . . . . fixed-size binary tree source induced by p, see Section E.1

$\mathcal{S}_{fh}(p)$ . . . . fixed-height binary tree source induced by p, see Section E.2

$\mathfrak{S}_{fs}(p)$ . . . . fixed-size ordinal tree source induced by p, see Section J

$\mathfrak{S}_{fcns}(\mathcal{S})$ . . . . FCNS-source of the fixed-size binary tree source $\mathcal{S}$, see Definition J.4

$H_n(\mathcal{S}_{fs}(p))$ . . . . entropy induced by the fixed-size source $\mathcal{S}_{fs}(p)$ over the set $\mathcal{T}_n$, Section E.3

$H_h(\mathcal{S}_{fh}(p))$ . . . . entropy induced by the fixed-height source $\mathcal{S}_{fh}(p)$ over the set $\mathcal{T}^h$, see Section E.3

$\mathcal{T}_n(\mathcal{P})$, $\mathcal{T}(\mathcal{P})$ . . . . set of binary trees of size n which satisfy property $\mathcal{P}$, $\mathcal{T}(\mathcal{P}) = \bigcup_{n \geq 0} \mathcal{T}_n(\mathcal{P})$

$\mathcal{U}_\mathcal{P}$ . . . . uniform subclass source with respect to property $\mathcal{P}$, see Section F

$\mathcal{T}(\mathcal{A})$ . . . . set of AVL trees, see Example E.6 and Example F.2

$\mathcal{T}(\mathcal{R})$ . . . . set of red-black trees, see Example F.3

$\mathcal{T}(\mathcal{W}_\alpha)$ . . . . set of α-weight-balanced trees, see Example F.4

References

[1] *http://github.com/ot/succinct.*

[2] Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Transactions on Algorithms*, 1(2):243–264, October 2005. *doi:10.1145/1103963.1103966.*

[3] Diego Arroyuelo and Rajeev Raman. Adaptive succinctness. In *String Processing and Information Retrieval (SPIRE)*, pages 467–481. Springer International Publishing, 2019. *doi:10.1007/978-3-030-32686-9_33.*

[4] Jérémy Barbay, Johannes Fischer, and Gonzalo Navarro. LRM-trees: Compressed indices, adaptive sorting, and compressed permutations. *Theoretical Computer Science*, 459:26–41, November 2012. *doi:10.1016/j.tcs.2012.08.010.*

[5] Niklas Baumstark, Simon Gog, Tobias Heuer, and Julian Labeit. Practical range minimum queries revisited. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *International Symposium on Experimental Algorithms (SEA)*, volume 75 of *LIPIcs*, pages 12:1–12:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. *doi:10.4230/LIPIcs.SEA.2017.12.*

[6] Djamal Belazzougui, Veli Mäkinen, and Daniel Valenzuela. Compressed suffix array. In *Encyclopedia of Algorithms*, pages 1–6. Springer US, 2014. *doi:10.1007/978-3-642-27848-8_82-2.*

[7] Philip Bille, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Tree compression with top trees. *Information and Computation*, 243:166–177, August 2015. *doi:10.1016/j.ic.2014.12.012.*

[8] Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM Journal on Computing*, 44(3):513–539, January 2015. *doi:10.1137/130936889.*

[9] Yongwook Choi and Wojciech Szpankowski. Compression of graphical structures: Fundamental limits, algorithms, and experiments. *IEEE Transactions on Information Theory*, 58(2):620–638, February 2012. *doi:10.1109/tit.2011.2173710.*

[10] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley Interscience, 2nd edition, 2006.

[11] Pooya Davoodi, Gonzalo Navarro, Rajeev Raman, and Srinivasa Rao Satti. Encoding range minima and range top-2 queries. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2016):20130131–20130131, apr 2014. *doi:10.1098/rsta.2013.0131.*

[12] Pooya Davoodi, Rajeev Raman, and Srinivasa Rao Satti. On succinct representations of binary trees. *Mathematics in Computer Science*, 11(2):177–189, March 2017. *doi:10.1007/s11786-017-0294-4.*

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

[14] Bartlomiej Dudek and Pawel Gawrychowski. Slowing down top trees for better worst-case compression. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPIcs*, pages 16:1–16:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CPM.2018.16.

[15] M. Effros, K. Visweswariah, S. R. Kulkarni, and S. Verdu. Universal lossless source coding with the burrows wheeler transform. *IEEE Transactions on Information Theory*, 48(5):1061–1081, May 2002. doi:10.1109/18.995542.

[16] Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, June 2014. doi:10.1007/s00453-012-9664-0.

[17] Arash Farzan, Rajeev Raman, and S. Srinivasa Rao. Universal succinct representations of trees? In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 451–462. Springer, 2009. doi:10.1007/978-3-642-02927-1_38.

[18] Héctor Ferrada and Gonzalo Navarro. Improved range minimum queries. *Journal of Discrete Algorithms*, 43:72–80, mar 2017. doi:10.1016/j.jda.2016.09.002.

[19] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Comput. Soc, 2000. doi:10.1109/sfcs.2000.892127.

[20] Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science*, 372(1):115–121, March 2007. doi:10.1016/j.tcs.2006.12.012.

[21] Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, January 2011. doi:10.1137/090779759.

[22] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. (available on author's website: http://algo.inria.fr/flajolet/Publications/book.pdf).

[23] Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *STOC 1984*. ACM Press, 1984. doi:10.1145/800057.808675.

[24] Moses Ganardi, Danny Hucke, Artur Jez, Markus Lohrey, and Eric Noeth. Constructing small tree grammars and small circuits for formulas. *Journal of Computer and System Sciences*, 86:136–158, June 2017. doi:10.1016/j.jcss.2016.12.007.

[25] 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, October 2019. doi:10.1109/tit.2019.2919829.

[26] Moses Ganardi, Danny Hucke, Markus Lohrey, and Eric Noeth. Tree compression using string grammars. *Algorithmica*, 80(3):885–917, February 2017. doi:10.1007/s00453-017-0279-3.

[27] Moses Ganardi, Artur Jez, and Markus Lohrey. Balancing straight-line programs. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1169–1183. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00073.

[28] Michal Ganczorz. Entropy lower bounds for dictionary compression. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 11:1–11:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.11.

[29] Michał Gańczorz. Using statistical encoding to achieve tree succinctness never seen before. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, LIPIcs, pages 22:1–22:29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.STACS.2020.22.

[30] Adrià Gascón, Markus Lohrey, Sebastian Maneth, Carl Philipp Reh, and Kurt Sieber. Grammar-based compression of unranked trees. *Theory of Computing Systems*, 64(1):141–176, 2020. doi:10.1007/s00224-019-09942-y.

[31] Pawel Gawrychowski and Artur Jez. LZ77 Factorisation of Trees. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*, volume 65 of *LIPIcs*, pages 35:1–35:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2016.35.

[32] Paweł Gawrychowski, Seungbum Jo, Shay Mozes, and Oren Weimann. Compressed range minimum queries. *Theoretical Computer Science*, 812:39–48, April 2020. doi:10.1016/j.tcs.2019.07.002.

[33] Paweł Gawrychowski and Patrick K. Nicholson. Optimal encodings for range top-k, selection, and min-max. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 593–604, 2015. doi:10.1007/978-3-662-47672-7_48.

[34] Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2(4):510–534, October 2006. doi:10.1145/1198513.1198516.

[35] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *International Symposium on Experimental Algorithms (SEA)*, pages 326–337, 2014. doi:10.1007/978-3-319-07959-2_28.

[36] Zbigniew Gołębiewski, Abram Magner, and Wojciech Szpankowski. Entropy and optimal compression of some general plane trees. *ACM Transactions on Algorithms*, 15(1):1–23, January 2019. doi:10.1145/3275444.

[37] Mordecai Golin, John Iacono, Danny Krizanc, Rajeev Raman, Srinivasa Rao Satti, and Sunil Shende. Encoding 2d range maximum queries. *Theoretical Computer Science*, 609:316–327, January 2016. doi:10.1016/j.tcs.2015.10.012.

[38] Rodrigo González and Gonzalo Navarro. Statistical encoding of succinct data structures. In *Combinatorial Pattern Matching*, pages 294–305. Springer Berlin Heidelberg, 2006. doi:10.1007/11780441_27.

[39] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation For Computer Science*. Addison-Wesley, 1994.

[40] Roberto Grossi. Random access to high-order entropy compressed text. In *Lecture Notes in Computer Science*, pages 199–215. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40273-9_14.

[41] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*. ACM Press, 2000. doi:10.1145/335305.335351.

[42] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, January 2005. doi:10.1137/s0097539702402354.

[43] Dan Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.

[44] Meng He, J. Ian Munro, and Srinivasa Satti Rao. Succinct ordinal trees based on tree covering. *ACM Transactions on Algorithms*, 8(4):1–32, September 2012. doi:10.1145/2344422.2344432.

[45] Lorenz Hübschle-Schneider and Rajeev Raman. Tree compression with top trees revisited. In Evripidis Bampis, editor, *Experimental Algorithms - 14th International Symposium, SEA 2015, Paris, France, June 29 - July 1, 2015, Proceedings*, volume 9125 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2015. doi:10.1007/978-3-319-20086-6_2.

[46] Danny Hucke, Markus Lohrey, and Louisa Seelbach Benkner. Entropy bounds for grammar-based tree compressors. In *IEEE International Symposium on Information Theory (ISIT)*. IEEE, July 2019. doi:10.1109/isit.2019.8849372.

[47] Danny Hucke, Markus Lohrey, and Louisa Seelbach Benkner. A comparison of empirical tree entropies. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2020. doi:10.1007/978-3-030-59212-7_17.

[48] Hsien-Kuei Hwang and Ralph Neininger. Phase change of limit laws in the quicksort recurrence under varying toll functions. *SIAM Journal on Computing*, 31(6):1687–1722, jan 2002. doi:10.1137/s009753970138390x.

[49] OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, A001263, 2021. URL: https://oeis.org/A001263.

[50] G. Jacobson. Space-efficient static trees and graphs. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1989. doi:10.1109/sfcs.1989.63533.

[51] Philippe Jacquet and Wojciech Szpankowski. *Analytic Pattern Matching*. Cambridge University Press, 2015.

[52] 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, March 2012. doi:10.1016/j.jcss.2011.09.002.

[53] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. ACM Press, 2018. doi:10.1145/3188745.3188814.

[54] J.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, May 2000. doi:10.1109/18.841160.

[55] John C. Kieffer, En-Hui Yang, and Wojciech Szpankowski. Structural complexity of random binary trees. In *2009 IEEE International Symposium on Information Theory*. IEEE, jun 2009. doi:10.1109/isit.2009.5205704.

[56] Markus Lohrey. Grammar-based tree compression. In *International Conference on Developments in Language Theory*, pages 46–57. Springer, 2015.

[57] Markus Lohrey, Sebastian Maneth, and Manfred Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *Journal of Computer and System Sciences*, 78(5):1651–1669, September 2012. doi:10.1016/j.jcss.2012.03.003.

[58] Markus Lohrey, Carl Philipp Reh, and Kurt Sieber. Optimal top dag compression, 2017. arXiv:1712.05822.

[59] Tomasz Luczak, Abram Magner, and Wojciech Szpankowski. Compression of preferential attachment graphs. In *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, July 2019. doi:10.1109/isit.2019.8849739.

[60] Abram Magner, Krzysztof Turowski, and Wojciech Szpankowski. Lossless compression of binary trees with correlated vertex names. *IEEE Transactions on Information Theory*, 64(9):6070–6080, sep 2018. doi:10.1109/tit.2018.2851224.

[61] Conrado Martínez. *Statistics under the BST model*. PhD thesis, University Barcelona, 1992.

[62] Raymond E. Miller, Nicholas Pippenger, Arnold L. Rosenberg, and Lawrence Snyder. Optimal 2,3-trees. *SIAM Journal on Computing*, 8(1):42–59, February 1979. doi:10.1137/0208004.

[63] J. Ian Munro and Sebastian Wild. Entropy trees and range-minimum queries in optimal average-case space, 2019. arXiv:1903.02533.

[64] Gonzalo Navarro. *Compact Data Structures – A practical approach*. Cambridge University Press, 2016.

[65] Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Computing Surveys*, 54(2):29:1–29:36, February 2021.

[66] Gonzalo Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Computing Surveys*, 54(2):26:1–26:38, February 2021. doi:10.1145/3432999.

[67] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2, April 2007. doi:10.1145/1216370.1216372.

[68] Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 10(3):1–39, may 2014. doi:10.1145/2601073.

[69] Jürg Nievergelt and Edward M. Reingold. Binary search trees of bounded balance. *SIAM J. Comput.*, 2(1):33–43, 1973. doi:10.1137/0202005.

[70] Andrew M. Odlyzko. Some new methods and results in tree enumeration. *Congressus Numerantium*, 42:27–52, 1984. URL: http://www.dtc.umn.edu/~odlyzko/doc/arch/enumer.methods.pdf.

[71] Nicola Prezza. Optimal Rank and Select Queries on Dictionary-Compressed Text. In Nadia Pisanti and Solon P. Pissis, editors, *Symposium on Combinatorial Pattern Matching (CPM)*, volume 128 of *LIPIcs*, pages 4:1–4:12, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CPM.2019.4.

[72] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4):43–es, nov 2007. doi:10.1145/1290672.1290680.

[73] Rajeev Raman and S. Srinivasa Rao. Succinct representations of ordinal trees. In Brodnik A., López-Ortiz A., Raman V., and Viola A., editors, *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *LNCS*, pages 319–332. Springer, 2013. doi:10.1007/978-3-642-40273-9_20.

[74] Robert Sedgewick. Left-leaning red-black trees, 2008. URL: http://www.cs.princeton.edu/~rs/talks/LLRB/LLRB.pdf.

[75] Robert Sedgewick and Philippe Flajolet. *An introduction to the analysis of algorithms*. Addison-Wesley-Longman, 1996.

[76] Louisa Seelbach Benkner and Markus Lohrey. Average Case Analysis of Leaf-Centric Binary Tree Sources. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 117 of *LIPIcs*, pages 16:1–16:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl. doi:10.4230/LIPIcs.MFCS.2018.16.

[77] Dekel Tsur. Representation of ordered trees with a given degree distribution, 2018. arXiv:1807.00371.

[78] Elad Verbin and Wei Yu. Data structure lower bounds on random access to grammar-compressed strings. In *Combinatorial Pattern Matching*, pages 247–258. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-38905-4_24.

[79] Sebastian Wild. *An Earley-style Parser for Solving the RNA-RNA Interaction Problem*. Bachelor's thesis, TU Kaiserslautern, 2010. URL: https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-22827.

[80] Sebastian Wild. *Dual-Pivot Quicksort and Beyond: Analysis of Multiway Partitioning and Its Practical Potential*. Dissertation (Ph. D. thesis), 2016. URL: https://www.wild-inter.net/publications/wild-2016.

[81] Sebastian Wild. Quicksort is optimal for many equal keys. In *Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 8–22. SIAM, 2018. arXiv:1608.04906, doi:10.1137/1.9781611975062.2.

[82] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, jun 1987. doi:10.1145/214762.214771.

[83] 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, March 2014. doi:10.1109/tit.2013.2295392.