Concatenation trees: A framework for efficient universal cycle and de Bruijn sequence constructions

Joe Sawada
University of Guelph, Canada

Jackson Sears
University of Guelph, Canada

Andrew Trautrim
University of Guelph, Canada

Aaron Williams
Williams College, USA

Abstract

Classic cycle-joining techniques have found widespread application in creating universal cycles for a diverse range of combinatorial objects, such as shorthand permutations, weak orders, orientable sequences, and various subsets of k-ary strings, including de Bruijn sequences. In the most favorable scenarios, these algorithms operate with a space complexity of $O(n)$ and require $O(n)$ time to generate each symbol in the sequences. In contrast, concatenation-based methods have been developed for a limited selection of universal cycles. In each of these instances, the universal cycles can be generated far more efficiently, with an amortized time complexity of $O(1)$ per symbol, while still using $O(n)$ space. This paper introduces *concatenation trees*, which serve as the fundamental structures needed to bridge the gap between cycle-joining constructions based on the pure cycle register and corresponding concatenation-based approaches. They immediately demystify the relationship between the classic Lyndon word concatenation construction of de Bruijn sequences and a corresponding cycle-joining based construction. To underscore their significance, concatenation trees are applied to construct universal cycles for shorthand permutations and weak orders in $O(1)$-amortized time per symbol. Moreover, we provide insights as to how similar results can be obtained for other universal cycles including cut-down de Bruijn sequences and orientable sequences.

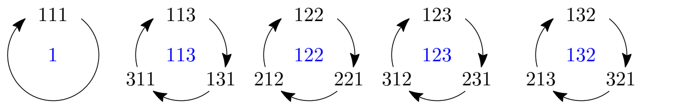
1 Introduction

Readers are likely familiar with the concept of a *de Bruijn sequence*, which is a circular string of length k^n in which every n-bit k-ary string appears once as a substring. For example, a binary de Bruijn sequence for $n = 4$ is 0000100110101111. The study of these sequences dates back to Pingala's *Chandahśāstra* छन्दःशास्त्र ('A Treatise on Prosody') over two thousand years ago (see [32, 48, 49, 50]). They have a wide variety of well-known modern-day applications [1] and their theory is even being applied to de novo assembly of read sequences into a genome [4, 9, 34, 47, 52]. More broadly, when the underlying objects are not k-ary strings, the analogous concept is often called a *universal cycle* [8], and they have been studied for many fundamental objects including permutations [31, 26, 40, 51], combinations [10, 28, 29], set partitions [25], and graphs [6].

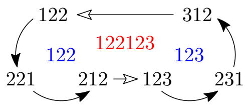
In this paper we wish to establish a new high-water mark for the efficient generation of de Bruijn sequences and universal cycles. To illustrate our approach, it is helpful to consider a slightly more complex object. A *weak order* is a way competitors can rank in an event, where ties are allowed. For example, in a horse race with five horses labeled h_1, h_2, h_3, h_4, h_5, the weak order (using a rank representation) 22451 indicates h_5 finished first, the horses h_1 and h_2 tied for second, horse h_3 finished fourth, and horse h_4 finished fifth. No horse finished third as a result of the tie for second. Let $\mathbf{W}(n)$ denote the set of weak orders of order n. For example, the thirteen weak orders for $n = 3$ are given below:

$$\mathbf{W}(3) = \{111, 113, 131, 311, 122, 212, 221, 123, 132, 213, 231, 312, 321\}.$$

Note that $\mathbf{W}(n)$ is closed under rotation. For this reason, we can construct a universal cycle by applying the pure cycling register (PCR) to induce small cycles, and then repeatedly join them together. In this approach, we partition $\mathbf{W}(n)$ into equivalence classes under rotation. These classes are called *necklaces* and we use the lexicographically smallest member of each class as its representative. So $\{113, 131, 311\}$ is one class with representative 113, and $\{111\}$ is another class. Each class of size t has a simple universal cycle of length t, namely the representative's aperiodic prefix (i.e., the shortest prefix of a string that can be concatenated some number of times to create the entire string). So 113 is a universal cycle for $\{113, 131, 311\}$, and 1 is a universal cycle for $\{111\}$ (since 1 is viewed cyclically). Each necklace class can be viewed as a directed cycle induced by the PCR, where each edge corresponds to a rotation (i.e., the leftmost symbol is shifted out and then shifted back in as the new rightmost symbol), as seen in Figure 1a for $n = 3$. Two cycles can be joined together via a conjugate pair (formally defined in Section 2.1) to create a larger cycle as illustrated in Figure 1b. This is done by replacing a pair of rotation edges with a pair of edges that shift in a new symbol. Repeating this process yields a universal cycle 1113213122123 for $\mathbf{W}(3)$.



(a) Necklace cycles for $\mathbf{W}(3)$, where the representative of each class is at the top of its cycle, and the universal cycle is in the middle.

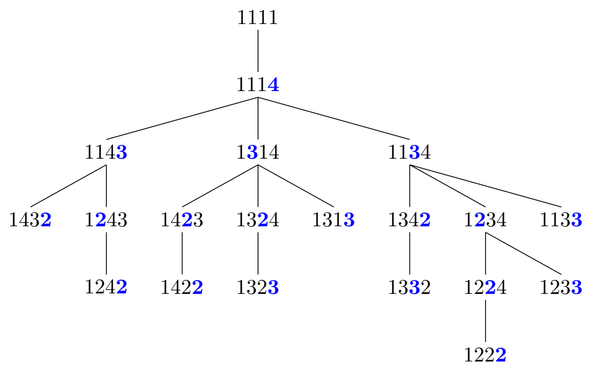


(b) Necklace cycles 122 and 123 are joined into a single cycle. The universal cycle for these strings is 122123.

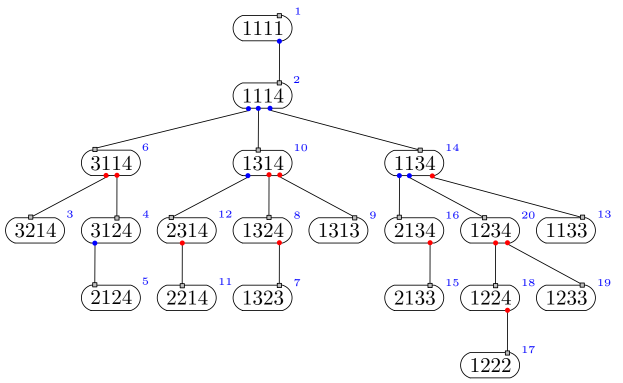
Figure 1 Initial steps to building a universal cycle for $\mathbf{W}_3$.

In many cases, pairs of cycles can be joined together to form a cycle-joining tree. For example, Figure 2a illustrates a cycle-joining tree for $\mathbf{W}(4)$ based on an explicit parent rule stated in Section 5.2. Given a cycle-joining tree, existing results in the literature [22, 23] allow us to generate a corresponding universal cycle *one symbol at a time*. But what if we want to generate the universal cycle faster? For instance, suppose that instead of generating one symbol at a time, we can generate necklaces one at a time.[1] How can we do this? This goal of generating one necklace at a time has been achieved in only a handful of cases [12, 18, 21, 39, 42]. Most notably, the de Bruijn sequence known as the Ford sequence, or the *granddaddy* (see Knuth [33]), can be created by concatenating the associated representatives in lexicographic order [19], matching the de Bruijn sequence given earlier: 0 0001 0011 01 0111 1. But these concatenation constructions have been the exception rather

[1] In practice, a de Bruijn sequence (or universal cycle) does not need to be returned to an application one symbol at a time, but rather a word of length t can be shared between the generation algorithm and the application. The algorithm repeatedly informs the application that the next batch of t symbols in the sequence is ready. This allows the generation algorithm to slightly modify the shared word and provide t symbols to the application as efficiently as $O(1)$-amortized time [12].



(a) A cycle-joining tree for weak orders when $n = 4$. The precise parent rule appears in Section 5.2.



(b) A concatenation tree $\mathcal{T}_{weak}$ for weak orders when $n = 4$ illustrating the RCL order.

Figure 2 Two tree structures for creating a universal cycle for $\mathbf{W}_4$.

than the rule, and there has been no theoretical framework for understanding why they work. Here, we provide the missing link. For example, the unordered cycle joining tree in Figure 2a is redrawn in Figure 2b. The new diagram is a bifurcated ordered tree, meaning that children are ordered and partitioned into left and right classes, and importantly some representatives have changed. If the tree is explored using an *RCL traversal* (i.e., right children, then current, then left children), then — presto! — a concatenation construction of a universal cycle for $\mathbf{W}(4)$ is created:

1 1114 3214 3124 2124 3114 1323 1324 13 1314 2214 2314 1133 1134 2133 2134 1222 1224 1233 1234.

This paper introduces *concatenation trees* and *RCL traversals*, which bridge the gap between PCR-based cycle-joining trees and concatenation constructions for the corresponding universal cycle. The resulting concatenations can often be generated in $O(1)$-amortized time per symbol using polynomial space. The framework applies to k-ary alphabets allowing for the broadest possible application.

While our focus here is on PCR-based cycle-joining trees, preliminary evidence indicates that it can be generalized (though non-trivially) to other underlying feedback functions. This will unify a large body of independent results enabling new and interesting results. In particular, the recently introduced pure run-length register (PRR) [41] is conjectured to be the underlying feedback function used in a lexicographic composition construction [18]. Furthermore, the PRR is proved to be the underlying function used in the greedy prefer-same [13] and prefer-opposite [3] constructions; however, no concatenation construction is known. The first shift-rule based on the complementing cycling register (CCR) is noted to have a very good local 0-1 balance [27]; however, no corresponding concatenation construction is known. There are two known CCR-based concatenation constructions [20, 21], but there is no clear correlation to an underlying cycle-joining approach, even though one appears to be equivalent to a shift rule from [22]. The cool-lex concatenation constructions [39] have equivalent underlying shift rules based on the pure summing register (PSR) and the complementing summing register (CSR). This correspondence was not observed until considering larger alphabets [42], though little insight to the correspondence is provided in the proof. Cycle-joining constructions based on the PSR/CSR are also considered in [15, 16].

Outline. In Section 2, we present the necessary background definitions and notation along with a detailed discussion of cycle-joining trees and their corresponding successor rules. In Section 3, we introduce bifurcated ordered trees, which are the structure underlying concatenation trees. In Section 4, we introduce concatenation trees along with a statement of our main result. In Section 5, we apply our framework to a wide variety of interesting combinatorial objects, including de Bruijn sequences. In Section 6, we prove our main result.

2 Preliminaries

Let $\Sigma = \{0, 1, 2, \ldots, k-1\}$ denote an alphabet with k symbols. Let Σ^n denote the set of all length n strings over Σ. Let $\alpha = \mathtt{a}_1\mathtt{a}_2 \cdots \mathtt{a}_n$ denote a string in Σ^n. The notation α^t denotes t copies of α concatenated together. The *aperiodic prefix* of α, denoted $\mathrm{ap}(\alpha)$, is the shortest string β such that $\alpha = \beta^t$ for some $t \geq 1$; the *period* of α is $|\beta|$. For example, if $\alpha = 01010101$ then $\mathrm{ap}(\alpha) = 01$ and α has period equal to 2. If the period of α is n, then α is said to be *aperiodic*; otherwise, it is said to be *periodic*. When $k = 2$, let $\bar{\mathtt{a}}_i$ denote the complement of a bit $\mathtt{a}_i$.

A *necklace class* is an equivalence class of strings under rotation. A *necklace* is the lexicographically smallest representative of a necklace class. A *Lyndon word* is an aperiodic necklace. Let $\mathbf{N}_k(n)$ denote the set of all k-ary necklaces of order n. As an example, the six binary necklaces for $n = 4$ are: $\mathbf{N}_2(4) = \{0000, 0001, 0011, 0101, 0111, 1111\}$. Let $[\alpha]$ denote the set of all strings in α's necklace class. For example, $[0001] = [1000] = \{0001, 0010, 0100, 1000\}$ and $[0101] = \{0101, 1010\}$. The *pure cycling register* (PCR) is a shift register with feedback function $f(\mathtt{a}_1\mathtt{a}_2 \cdots \mathtt{a}_n) = \mathtt{a}_1$. Starting with α, it induces a cycle containing the strings in α's necklace class. For example,

$$001101 \rightarrow 011010 \rightarrow 110100 \rightarrow 101001 \rightarrow 010011 \rightarrow 100110 \rightarrow \mathit{001101}$$

is a cycle induced by the PCR that can be represented by any string in the cycle. Given a tree T with nodes (cycles induced by the PCR) labeled by necklace representatives $\{\alpha_1, \alpha_2, \ldots, \alpha_t\}$, let $\mathbf{S}_T = [\alpha_1] \cup [\alpha_2] \cup \cdots \cup [\alpha_t]$.

Given $\mathbf{S} \subseteq \Sigma^n$, a *universal cycle* U for $\mathbf{S}$ is a cyclic sequence of length $|\mathbf{S}|$ that contains each string in $\mathbf{S}$ as a substring (exactly once). Given a universal cycle U for a set $\mathbf{S} \subseteq \Sigma^n$, a *UC-successor* for U is a function $f : \mathbf{S} \rightarrow \Sigma$ such that $f(\alpha)$ is the symbol following α in U. If $\mathbf{S} = \Sigma^n$ then U is a *de Bruijn sequence* (DB sequence), and we call a UC-successor a *DB-successor*. A UC-successor can be thought of as a shift rule where the underlying object is a universal cycle.

2.1 Cycle joining trees

In this section we review how two universal cycles can be joined to obtain a larger universal cycle. Let $\mathtt{x}, \mathtt{y}$ be distinct symbols in Σ. If $\alpha = \mathtt{x}\mathtt{a}_2 \cdots \mathtt{a}_n$ and $\hat{\alpha} = \mathtt{y}\mathtt{a}_2 \cdots \mathtt{a}_n$, then α and $\hat{\alpha}$ are said to be *conjugates* of each other, and $(\alpha, \hat{\alpha})$ is called a *conjugate pair*. The following well-known result (see for instance Lemma 3 in [43]) based on conjugate pairs is the crux of the cycle-joining approach.[2]

▶ **Theorem 1.** *Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be disjoint subsets of Σ^n such that $\alpha = \mathtt{x}\mathtt{a}_2 \cdots \mathtt{a}_n \in \mathbf{S}_1$ and $\hat{\alpha} = \mathtt{y}\mathtt{a}_2 \cdots \mathtt{a}_n \in \mathbf{S}_2$; $(\alpha, \hat{\alpha})$ is a conjugate pair. If U_1 is a universal cycle for $\mathbf{S}_1$ with suffix α and U_2 is a universal cycle for $\mathbf{S}_2$ with suffix $\hat{\alpha}$ then $U = U_1U_2$ is a universal cycle for $\mathbf{S}_1 \cup \mathbf{S}_2$.*

Let U_i denote a universal cycle for $\mathbf{S}_i \subseteq \Sigma^n$. Two universal cycles U_1 and U_2 are said to be *disjoint* if $\mathbf{S}_1 \cap \mathbf{S}_2 = \emptyset$. A *cycle-joining tree* $\mathbb{T}$ is an unordered tree where the nodes correspond to a disjoint set of universal cycles $U_1, U_2, \ldots, U_t$; an edge between U_i and U_j is defined by a conjugate pair $(\alpha, \hat{\alpha})$ such that $\alpha \in \mathbf{S}_i$ and $\hat{\alpha} \in \mathbf{S}_j$. For our purposes, we consider cycle-joining trees to be rooted. If the cycles are induced by the PCR, i.e., the cycles correspond to necklace classes, then $\mathbb{T}$ is said to be a *PCR-based cycle-joining tree*. As examples, four PCR-based cycle-joining trees are illustrated in Figure 3; their nodes are labeled by the necklaces $\mathbf{N}_2(6)$. They are defined by the following *parent-rules*, which determines the parent of given non-root node.

Four "simplest" parent rules defining binary PCR-based cycle-joining trees

- $\mathbb{T}_1$: rooted at 1^n and the parent of every other node $\alpha \in \mathbf{N}_2(n)$ is obtained by flipping the last 0.
- $\mathbb{T}_2$: rooted at 0^n and the parent of every other node $\alpha \in \mathbf{N}_2(n)$ is obtained by flipping the first 1.
- $\mathbb{T}_3$: rooted at 0^n and the parent of every other node $\alpha \in \mathbf{N}_2(n)$ is obtained by flipping the last 1.
- $\mathbb{T}_4$: rooted at 1^n and the parent of every other node $\alpha \in \mathbf{N}_2(n)$ is obtained by flipping the first 0.

[2] The cycle-joining approach has graph theoretic underpinnings related to Hierholzer's algorithm for constructing Euler cycles [24].

Note that for $\mathbb{T}_3$ and $\mathbb{T}_4$, the parent of a node α is obtained by first flipping the named bit and then rotating the string to its lexicographically least rotation to obtain a necklace. Each node α and its parent β are joined by a conjugate pair where the highlighted bit in α is the first bit in one of the conjugates. For example, the nodes $\alpha = 011011$ and $\beta = 001011$ in $\mathbb{T}_2$ from Figure 3 are joined by the conjugate pair $(1\underline{10110}, 0\underline{10110})$.

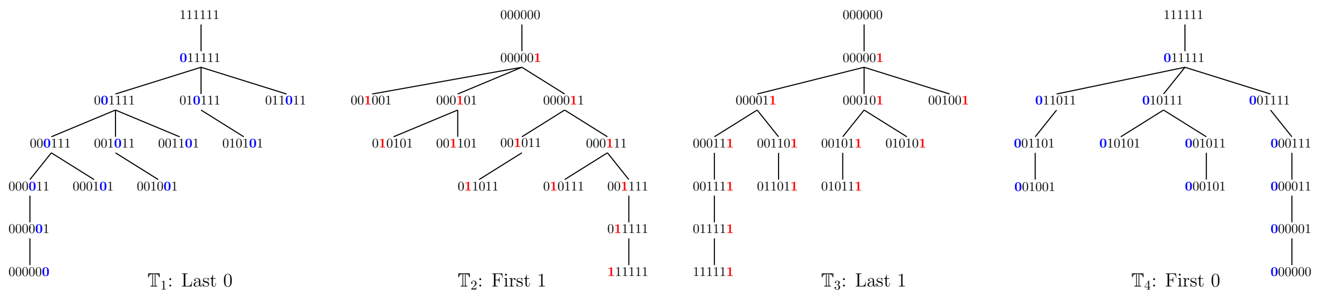
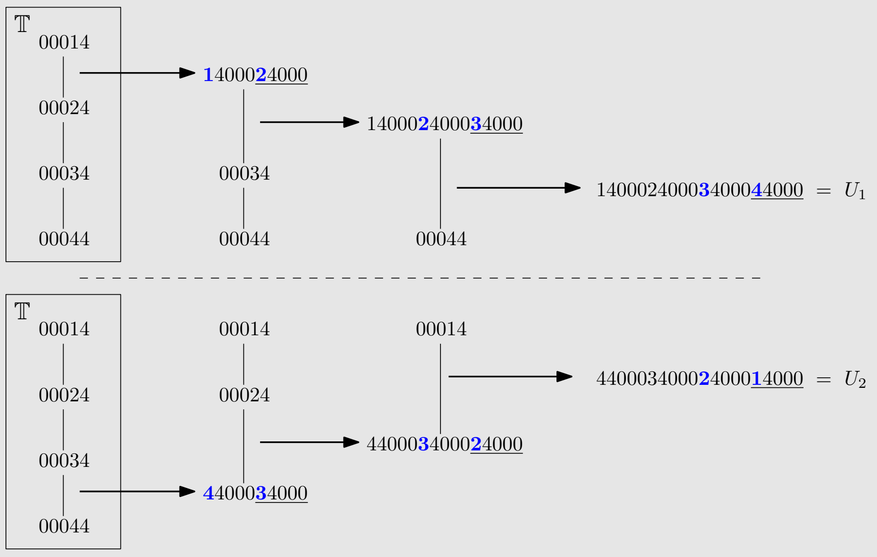


Figure 3 Cycle-joining trees for $n = 6$ derived from simple parent rules. The node 001101 joined to a different parent cycle in each tree. In particular, the edge 001101–001111 in $\mathbb{T}_1$ is obtained by flipping its last 0.

When two adjacent nodes U_i and U_j in a cycle-joining tree $\mathbb{T}$ are joined to obtain U via Theorem 1 (rotating the cycles as appropriate), the nodes are unified and replaced with U (the edge between U_i and U_j is contracted). Repeating this process until only one node remains produces a universal cycle for $\mathbf{S}_1 \cup \mathbf{S}_2 \cup \cdots \cup \mathbf{S}_t$. In the binary case, the same universal cycle is produced, no matter the order in which the cycles are joined. This is because no string can belong to more than one conjugate pair in the underlying definition of $\mathbb{T}$. However, when $k > 2$, the order can be important.

Example 1 The following illustrates two (of several) different ways to join the cycles in a PCR-based cycle-joining tree $\mathbb{T}$ for $n = 5$ and $k = 5$ with four nodes represented by $00014, 00024, 00034, 00044$ joined via conjugate pairs $(14000, 24000)$, $(24000, 34000)$, $(34000, 44000)$.



The resulting universal cycle for $\mathbf{S}_{\mathbb{T}} = [00014] \cup [00024] \cup [00034] \cup [00044]$ is different in each case.

In upcoming discussion regarding both successor rules and concatenation trees, we require the following assumption about underlying cycle-joining trees when $k > 2$.

▶ **Assumption 2.** *If a node in a cycle-joining tree has two children joined via conjugate pairs* $(\mathtt{x}\mathtt{a}_2 \cdots \mathtt{a}_n, \mathtt{y}\mathtt{a}_2 \cdots \mathtt{a}_n)$ *and* $(\mathtt{x}'\mathtt{b}_2 \cdots \mathtt{b}_n, \mathtt{y}'\mathtt{b}_2 \cdots \mathtt{b}_n)$ *then* $\mathtt{a}_2 \cdots \mathtt{a}_n \neq \mathtt{b}_2 \cdots \mathtt{b}_n$.

Observe that this assumption is satisfied in our previous example, and is always satisfied when $k = 2$.

2.2 Successor-rule constructions

Let $\mathbb{T}$ be a PCR-based cycle-joining tree where the nodes are joined by a set $\mathbf{C}$ of conjugate pairs. We say γ *belongs to* a conjugate pair $(\alpha, \hat{\alpha})$ if either $\gamma = \alpha$ or $\gamma = \hat{\alpha}$. If $k = 2$, then from [22], the following function f_0 is a UC-successor for $\mathbf{S}_{\mathbb{T}}$, where $\alpha = \mathtt{a}_1\mathtt{a}_2 \cdots \mathtt{a}_n$:

$$f_0(\alpha) = \begin{cases} \overline{\mathtt{a}}_1 & \text{if } \alpha \text{ belongs to some conjugate pair in } \mathbf{C}; \\ \mathtt{a}_1 & \text{otherwise.} \end{cases}$$

More generally, let $k \geq 2$. Let $\mathbb{T}$ satisfy Assumption 2, which implies a string can belong to at most two conjugate pairs (joining a node to its parent and/or a child). Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ denote a maximal length path of nodes in $\mathbb{T}$ such that for each $1 \leq i < m$, the node α_i is the parent of α_{i+1} and they are joined via a conjugate pair of the form $(\mathtt{x}_i\beta, \mathtt{x}_{i+1}\beta)$; β is the same in each conjugate pair. We call such a path a *chain* of length m. Suppose $\alpha = \mathtt{x}_i\beta$ belongs to a conjugate pair that joins two nodes in a chain of length m for some $1 \leq i \leq m$. Let $\text{first}(\alpha) = \mathtt{x}_1$ and let $\text{last}(\alpha) = \mathtt{x}_m$. Define $g_1(\alpha) = \mathtt{x}_{i+1}$, where $\mathtt{x}_{m+1} = \text{first}(\alpha)$, and let $g_2(\alpha) = \mathtt{x}_{i-1}$, where $\mathtt{x}_0 = \text{last}(\alpha)$. Then the following functions f_1 and f_2 are both UC-successors for $\mathbf{S}_{\mathbb{T}}$ (see Theorem 2.8 and Theorem 2.9 from [23]):

$$f_1(\alpha) = \begin{cases} g_1(\alpha) & \text{if } \alpha \text{ belongs to a conjugate pair in } \mathbf{C}; \\ \mathtt{a}_1 & \text{otherwise,} \end{cases}$$

$$f_2(\alpha) = \begin{cases} g_2(\alpha) & \text{if } \alpha \text{ belongs to a conjugate pair in } \mathbf{C}; \\ \mathtt{a}_1 & \text{otherwise.} \end{cases}$$

If $k = 2$ then $f_0 = f_1 = f_2$.

Example 2 Continuing Example 1, let $\alpha = 44000$; it belongs to a conjugate pair. Note that $\alpha_1 = 00014$, $\alpha_2 = 00024$, $\alpha_3 = 00034$, $\alpha_4 = 00044$ form a chain of length $m = 4$ where $\text{first}(\alpha) = 1$ and $\text{last}(\alpha) = 4$. Then $g_1(\alpha) = 1$ and $g_2(\alpha) = 3$. Observe that f_1 is a UC-successor for U_1; it joins the cycles top-down and the symbol following $\alpha = 44000$ is $f_1(\alpha) = g_1(\alpha) = 1$. Similarly, f_2 is a UC-successor for U_2; it joins the cycles bottom-up and the symbol following $\alpha = 44000$ is $f_2(\alpha) = g_2(\alpha) = 3$.

Applying a UC-successor f_0, f_1, or f_2 directly requires an exponential amount of memory to store the conjugate pairs. However, a cycle-joining tree defined by a simple parent rule may allow for a much more efficient implementation, using as little as $O(n)$ space and running in $O(n)$ time. Recall the four parent rules stated for the trees $\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3, \mathbb{T}_4$. The upcoming four shift rules $\text{pcr}_1, \text{pcr}_2, \text{pcr}_3, \text{pcr}_4$, which correspond to f_0, are stated generally for any subtree T of the corresponding cycle-joining tree. Previously, these shift rules were stated for the entire trees in [22], and then for subtrees that included all nodes up to a given level [23] putting a restriction on the minimum or maximum weight (number of 1s) of any length-n substring.

$\mathbb{T}_1$ (Last 0) Let j be the smallest index of $\alpha = \mathtt{a}_1\mathtt{a}_2 \cdots \mathtt{a}_n$ such that $\mathtt{a}_j = 0$ and $j > 1$, or $j = n{+}1$ if no such index exists. Let $\gamma = \mathtt{a}_j\mathtt{a}_{j+1} \cdots \mathtt{a}_n 0 \mathtt{a}_2 \cdots \mathtt{a}_{j-1} = \mathtt{a}_j\mathtt{a}_{j+1} \cdots \mathtt{a}_n 0 1^{j-2}$.

$$\text{pcr}_1(\alpha) = \begin{cases} \overline{\mathtt{a}}_1 & \text{if } \gamma \text{ is a necklace and } \mathtt{a}_2 \cdots \mathtt{a}_n \overline{\mathtt{a}}_1 \in \mathbf{S}_T; \\ \mathtt{a}_1 & \text{otherwise.} \end{cases}$$

$\mathbb{T}_2$ (First 1) Let j be the largest index of $\alpha = \mathtt{a}_1\mathtt{a}_2 \cdots \mathtt{a}_n$ such that $\mathtt{a}_j = 1$, or $j = 0$ if no such index exists. Let $\gamma = \mathtt{a}_{j+1}\mathtt{a}_{j+2} \cdots \mathtt{a}_n 1 \mathtt{a}_2 \cdots \mathtt{a}_j = 0^{n-j} 1 \mathtt{a}_2 \cdots \mathtt{a}_j$.

$$\text{pcr}_2(\alpha) = \begin{cases} \overline{\mathtt{a}}_1 & \text{if } \gamma \text{ is a necklace and } \mathtt{a}_2 \cdots \mathtt{a}_n \overline{\mathtt{a}}_1 \in \mathbf{S}_T; \\ \mathtt{a}_1 & \text{otherwise.} \end{cases}$$

$\mathbb{T}_3$ (Last 1) Let $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n$ and let $\gamma = \mathtt{a}_2\mathtt{a}_3\cdots\mathtt{a}_n1$.

$$\mathrm{pcr}_3(\alpha) = \begin{cases} \overline{\mathtt{a}}_1 & \text{if } \gamma \text{ is a necklace and } \mathtt{a}_2\cdots\mathtt{a}_n\overline{\mathtt{a}}_1 \in \mathbf{S}_T; \\ \mathtt{a}_1 & \text{otherwise.} \end{cases}$$

$\mathbb{T}_4$ (First 0) Let $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n$ and let $\gamma = 0\mathtt{a}_2\mathtt{a}_3\cdots\mathtt{a}_n$.

$$\mathrm{pcr}_4(\alpha) = \begin{cases} \overline{\mathtt{a}}_1 & \text{if } \gamma \text{ is a necklace and } \mathtt{a}_2\cdots\mathtt{a}_n\overline{\mathtt{a}}_1 \in \mathbf{S}_T; \\ \mathtt{a}_1 & \text{otherwise.} \end{cases}$$

The DB sequences obtained by applying the four shift rules for $n = 6$ are provided in Table 1. The spacing between some symbols are used to illustrate the correspondence to upcoming concatenation constructions.

Shift rule	**DB sequence for $n = 6$**
pcr_1	0 000001 000011 000101 000111 001 001011 001101 001111 01 010111 011 011111 1
pcr_2	0 000001 001 000101 01 001101 000011 001011 011 000111 010111 001111 011111 1
pcr_3	1 111110 111100 111000 110 110100 110000 101110 101100 10 101000 100 100000 0
pcr_4	1 111110 110 100 100110 111010 10 110010 100010 111100 111000 110000 100000 0

Table 1 DB sequences resulting from the shift rules corresponding to the cycle-joining trees $\mathbb{T}_1$, $\mathbb{T}_2$, $\mathbb{T}_3$, $\mathbb{T}_4$ from Figure 3.

The DB sequence generated by pcr_1 is the well-known Ford sequence [17], and is called the *Granddaddy* by Knuth [33]. It is the lexicographically smallest DB sequence, and it can also be generated by a prefer-0 greedy approach attributed to Martin [35]. Furthermore, Fredricksen and Maiorana [19] demonstrate an equivalent necklace (or Lyndon word) concatenation construction that can generate the sequence in $O(1)$-amortized time per bit. The DB sequence generated by pcr_2 can also be generated in $O(1)$-amortized time per bit by concatenating necklaces in co-lexicographic order. The DB sequence generated by pcr_3, was first discovered by Jansen [30] for $k = 2$, then generalized in [45]. It is conjectured to have a simple concatenation construction by Gabric and Sawada [21], a fact we prove in this paper. The DB sequence generated by pcr_4, was first discovered by Gabric, Sawada, Williams, and Wong [22]. No concatenation construction for this sequence was previously known which served as the initial motivation for this work.

2.3 Insights into concatenation trees

The sequence in Table 1 generated by pcr_1 starting with 0^n has an interesting property: It corresponds to concatenating the *aperiodic prefixes* of each necklace in the corresponding cycle-joining tree $\mathbb{T}_1$ (see Figure 3) as they are visited in post-order. Notice also, that a post-order traversal visits the necklaces as they appear in lexicographic order; this corresponds to the well-known necklace concatenation construction [19]. Similarly, the sequence generated by the shift rule pcr_2 starting with 0^n corresponds to concatenating the aperiodic prefixes of each node in the corresponding cycle-joining tree $\mathbb{T}_2$ as they are visited in pre-order. This traversal visits the necklaces as they appear in colex order, which is another known concatenation construction [12]. Unfortunately, this *magic* does not carry over to the trees $\mathbb{T}_3$ and $\mathbb{T}_4$, no matter how we order the children; the existing proofs for $\mathbb{T}_1$ and $\mathbb{T}_2$ offer no higher-level insights or pathways towards generalization.

Our discovery to finding a concatenation construction for a given UC-successor is to tweak the corresponding cycle-joining tree by: (i) determining the appropriate representative of each cycle, (ii) determining an ordering of the children, and (iii) determining how the tree is traversed. The resulting concatenation trees for $\mathbb{T}_1$, $\mathbb{T}_2$, $\mathbb{T}_3$, and $\mathbb{T}_4$, which are formally defined in Section 4, are illustrated in Figure 7 for $n = 6$. The concatenation trees derived from $\mathbb{T}_1$ and $\mathbb{T}_2$ look very similar to the original cycle-joining trees. For the concatenation tree derived from $\mathbb{T}_3$, the representatives are obtained by rotating the initial prefix of 0s of a necklace to the suffix; a post-order traversal produces the corresponding DB sequence in Table 1. This traversal corresponds to visiting these representatives in reverse lexicographic order that is equivalent to a construction defined in [21]. The concatenation tree derived from $\mathbb{T}_4$ is non-trivial and proved to be the basis for discovering our more general result. Each representative is determined from its parent, and the tree differentiates "left-children" (blue dots) from "right-children" (red dots). A concatenation construction corresponding to pcr_4 is obtained by a somewhat unconventional traversal that recursively visits right-children, followed by the current node, followed by the left-children.

3 Bifurcated ordered trees

Our new "concatenation-tree" approach to generating DB sequences relies on tree structures that mix together ordered trees and binary trees. First we review basic tree concepts. Then we introduce our notion of a bifurcated ordered tree together with a traversal called an RCL traversal.

An *ordered tree* is a rooted tree in which the children of each node are given a total order. For example, a node in an ordered tree with three children has a first child, a second child, and a third (last) child. In contrast, a *cardinal tree* is a rooted tree in which the children of each node occupy specific positions. In particular, a *k-ary tree* has k positions for the children of each node. For example, each child of a node in a 3-ary tree is either a left-child, a middle child, or a right-child.

We consider a new type of tree that is both ordinal and cardinal; while ordered trees have one "type" of child, our trees will have two types of children. We refer to such a tree as a *bifurcated ordered tree* (*BOT*), with the two types of children being *left-children* and *right-children*. To illustrate bifurcated ordered trees, Figure 4 provides all BOTs with $n = 3$ nodes. This type

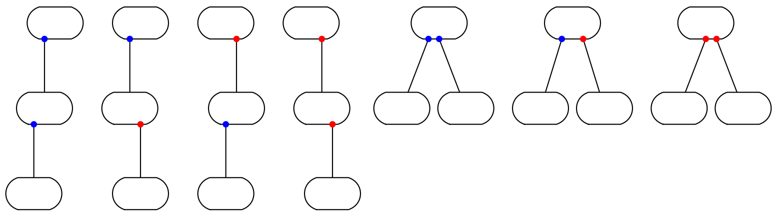


Figure 4 All eight bifurcated ordered trees (BOTs) with n=3 nodes. Each left-child descends from a blue ●, while each right-child descends from a red ●.

of "ordinal-cardinal" tree seems quite natural[3], and it is very likely to have been used in previous academic investigations. Nevertheless, the authors have not been able to find an exact match in the literature. In particular, 2-tuplet trees use a different notion of a root, and correspond more closely to ordered forests of BOTs. A computer program[4] to enumerate all BOTs demonstrates that the total number for n from 1 to 12 are:

$$1, 2, 7, 30, 143, 728, 3876, 21318, 120175, 690690, 4032015, 23841480.$$

When extended for larger n, the sequence corresponds to all 23 entries for sequence A006013 in the Online Encyclopedia of Integer Sequence [2]; however, no obvious relationship to such a tree is noted.

3.1 Right-Current-Left (RCL) traversals

The distinction between left-children and right-children in a BOT allows for a very natural notion of an *in-order traversal*: visit the left-children from first to last, then the current node, then the right-children from first to last. During our work with concatenation trees (see Section 4) it will be more natural to use a modified traversal, in which the right-children are visited before the left-children. Formally, we recursively define a *Right-Current-Left (RCL) traversal* of a bifurcated ordered tree starting from the root as follows:

- visit all right-children of the current node from first to last;
- visit the current node;
- visit all left-children of the current node from first to last.

Note that the resulting RCL order is not the same as a *reverse in-order traversal* (i.e., an in-order traversal written in reverse), since the children of each type are visited in the usual order (i.e., first to last) rather than in reverse order (i.e., last to first). An example of an RCL traversal is shown in Figure 5.

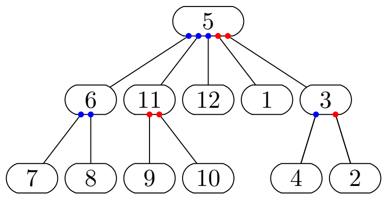


Figure 5 A BOT with its n=12 nodes labeled as they appear in RCL order.

[3] More broadly, one can consider ordered trees with $k \geq 2$ types of children. Experimentally, we have found that the number of these trees are counted by the Fuss-Catalan sequence $a(n, p, r) = r \cdot \binom{np+r}{n}/(np + r)$ with $p = k$ and $r = k - 1$, including unifurcated (OEIS A000108), bifurcated, trifurcated (OEIS A006632), quadfurcated (OEIS A118971), and even decemfurcated (OEIS A234573).

[4] This program was surprisingly non-trivial to implement, and is the topic of future work. A dynamic programming implementation in C used to construct the listing is given in Appendix A.

4 Concatenation trees

Let $\mathbb{T}$ be a PCR-based cycle-joining tree rooted at r satisfying Assumption 2. In this section we describe how $\mathbb{T}$ can be converted into a labeled BOT $\mathcal{T}$ we call a *concatenation tree*. The nodes and the parent-child relationship in $\mathcal{T}$ are the same as in $\mathbb{T}$; however, the labels (representatives) of the nodes may change. The definitions of these labels are defined recursively along with a corresponding *change index*, the unique index where a node's label differs from that of its parent. The root of $\mathcal{T}$ is r, and it is assigned an arbitrary change index c.[5] The label of a non-root node γ depends on the label of its parent $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n$, which can be written as $\beta_1\mathtt{x}\beta_2$ where $(\mathtt{x}\beta_2\beta_1, \mathtt{y}\beta_2\beta_1)$ is the conjugate pair joining α and γ in $\mathbb{T}$. If α is aperiodic, there is only one possible index i for $\mathtt{x}$; however, if it is periodic, there will be multiple such indices possible. If $\alpha = (\mathtt{a}_1\cdots\mathtt{a}_p)^q$ has period p with change index c where $jp < c \le jp+p$ for some integer $0 \le j < q$, then we say the *acceptable range* of α is $\{jp+1, \ldots, jp+p\}$. Note, if α is aperiodic, its acceptable range is $\{1, 2, \ldots, n\}$. Now, $\alpha = \beta_1\mathtt{x}\beta_2$ can be written uniquely such that $\mathtt{x}$ is found at an index i in α's acceptable range. The label of γ is defined to be $\beta_1\mathtt{y}\beta_2$ with change index i.

Example 3 Let $x = 001001001$ be the parent of $y = 001002001$ in a PCR-based cycle-joining tree $\mathbb{T}$ joined via the conjugate pair $(100100100, 200100100)$. Let α and γ denote the corresponding nodes in the concatenation tree $\mathcal{T}$. Suppose $\alpha = 1001001\underline{0}0$ (a rotation of x) with change index 8. Since α has period $p = 3$, its acceptable range is $\{7, 8, 9\}$. Thus, $\beta_1 = 100100$, $\mathtt{x} = 1$, $\beta_2 = 00$, $\alpha = \beta_1\mathtt{x}\beta_2$, and $\gamma = 100100\mathbf{2}00$ (a rotation of y) with change index 7.

To complete the definition of $\mathcal{T}$, we must specify how the children of a node with change index c are partitioned into ordered left-children and right-children: The left-children are those with change index less than c, and the right-children are those with change index greater than c. Both are ordered by increasing change index. A child with change index c can be considered to be either a left-child or right-child. We say $\mathcal{T}$ is a *left concatenation tree* if every node that has the same change index as its parent is considered to be a left-child; $\mathcal{T}$ is a *right concatenation tree* if every node that has the same change index as its parent is considered to be a right-child. Let concat($\mathbb{T}$, c, *left*) denote the left concatenation tree derived from $\mathbb{T}$ and let concat($\mathbb{T}$, c, *right*) denote the corresponding right concatenation tree derived from $\mathbb{T}$, where in each case the root is assigned change index c. See Figure 6 for example concatenation trees, where the small gray box on top of each node indicates the node's change index.

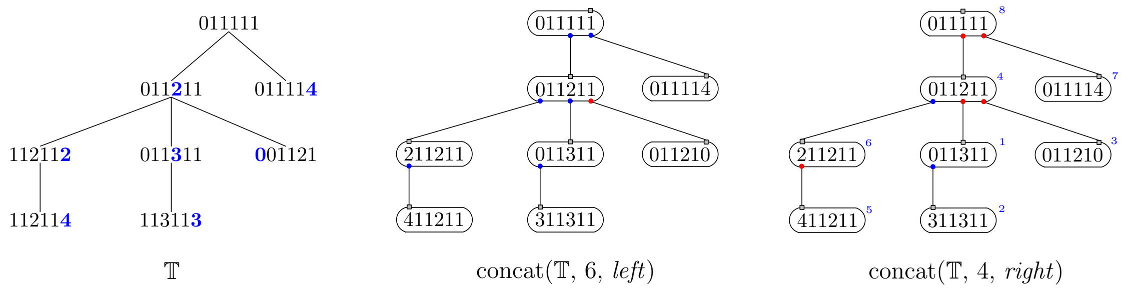
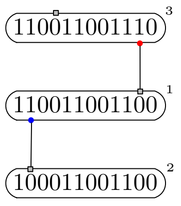


Figure 6 Left and right concatenation trees for a given cycle-joining tree $\mathbb{T}$. The right concatenation tree indicates the RCL order.

Let RCL($\mathcal{T}$) denote the *RCL sequence* produced by traversing the concatenation tree $\mathcal{T}$ in RCL order, outputting the aperiodic prefix of the node label when it is visited. For example, the RCL sequence for concat($\mathbb{T}$, 4, *right*) in Figure 6 is:

011311 $\underline{311}$ 011210 011211 411211 $\underline{211}$ 011114 011111.



It is critical how we defined the acceptable range, since our goal is to demonstrate that RCL($\mathcal{T}$) produces a universal cycle. For example, consider three necklace class representatives (a) 11001100110, (b) 110011001100, and (c) 100011001100 where $n = 12$. They can be joined by flipping the second last 0 in (b) and flipping the first 0 in (c); (a) is the parent of (b) and (b) is the parent of (c). A BOT for this cycle-joining tree is shown on the right. It is *not* a concatenation tree since the change index for

[5] Though the change index of the root is arbitrary, its choice may impact the "niceness" of the upcoming RCL sequence.

the bottom node is outside the acceptable range of its periodic parent. Outputting the aperiodic prefixes of the nodes when visited in RCL order produces 1100 100110011001100 11001110. Since the substring 110011001100 appears twice, it is not a universal cycle.

The concatenation trees for the four cycle-joining trees in Figure 3 are given in Figure 7; the RCL sequence for each tree matches the sequences in Table 1. The only concatenation tree with both left-children and right-children is the one corresponding to $\text{concat}(\mathbb{T}_4, 6, \textit{left})$. In fact, it was the discovery of this tree that lead us to the introduction of BOTs and our definition of concatenation trees.

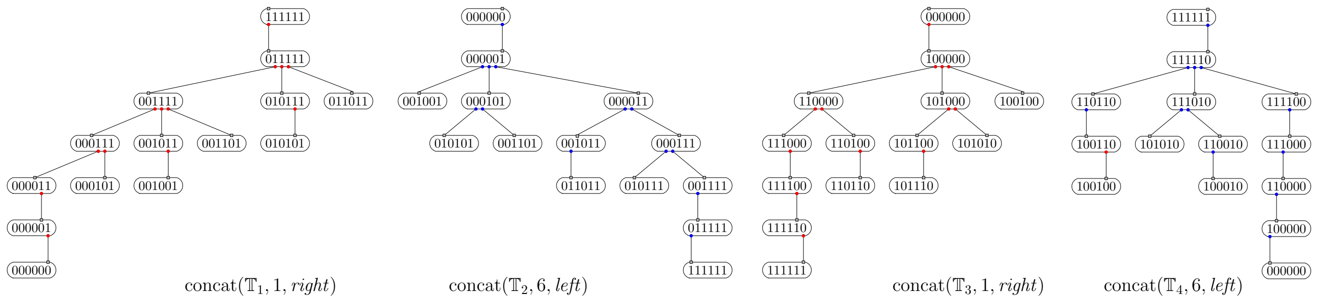

concat($\mathbb{T}_1$, 1, *right*)
concat($\mathbb{T}_2$, 6, *left*)
concat($\mathbb{T}_3$, 1, *right*)
concat($\mathbb{T}_4$, 6, *left*)

Figure 7 Concatenation trees for $n = 6$ based on $\mathbb{T}_1$, $\mathbb{T}_2$, $\mathbb{T}_3$, $\mathbb{T}_4$. These bifurcated ordered trees (BOTs) provide additional structure to the unordered cycle-joining trees from Figure 3. This structure provides the missing information for fully understanding the corresponding concatenation constructions. The gray box above each node indicates its change index.

We are ready to present our main result which we prove in Section 6.

▶ **Theorem 3.** *Let $\mathbb{T}$ be a PCR-based cycle-joining tree satisfying Assumption 2. Let $\mathcal{T}_1 = \text{concat}(\mathbb{T}, c, \textit{left})$ and let $\mathcal{T}_2 = \text{concat}(\mathbb{T}, c, \textit{right})$. Then*

- $\text{RCL}(\mathcal{T}_1)$ *is a universal cycle for $\mathbf{S}_{\mathbb{T}}$ with shift rule f_1, and*
- $\text{RCL}(\mathcal{T}_2)$ *is a universal cycle for $\mathbf{S}_{\mathbb{T}}$ with shift rule f_2.*

Recall that pcr_1, pcr_2, pcr_3, and pcr_4 were stated generally for subtrees of their corresponding cycle-joining tree.

▶ **Corollary 4.** *Let T_1, T_2, T_3, T_4 be subtrees of $\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3, \mathbb{T}_4$, respectively. For any $1 \le c \le n$ and $\ell \in \{\textit{left}, \textit{right}\}$:*

- $\text{RCL}(\text{concat}(T_1, c, \ell))$ *is a UC-successor for $\mathbf{S}_{T_1}$ with shift rule* pcr_1.
- $\text{RCL}(\text{concat}(T_2, c, \ell))$ *is a UC-successor for $\mathbf{S}_{T_2}$ with shift rule* pcr_2.
- $\text{RCL}(\text{concat}(T_3, c, \ell))$ *is a UC-successor for $\mathbf{S}_{T_3}$ with shift rule* pcr_3.
- $\text{RCL}(\text{concat}(T_4, c, \ell))$ *is a UC-successor for $\mathbf{S}_{T_4}$ with shift rule* pcr_4.

Efficient implementations of these concatenation constructions are presented in Section 5.4.

4.1 Algorithmic details and analysis

A concatenation tree can be traversed to produce a universal cycle in $O(1)$-amortized time per symbol; but, it requires exponential space to store the tree. However, if the children of a given node α can be computed without knowledge of the entire tree, then we can apply Algorithm 1 to traverse a concatenation tree $\mathcal{T}$ in a space-efficient manner. The initial call is RCL(α,c, ℓ) where $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n$ is the root node with change index c. The variable ℓ is set to 1 for left concatenation trees; ℓ is set to 0 for right concatenation trees. The crux of the algorithm is the function CHILD(α, i) which returns $\mathtt{x}$ if there exists $\mathtt{x} \in \Sigma$ such that $\mathtt{a}_1\cdots\mathtt{a}_{i-1}\mathtt{x}\mathtt{a}_{i+1}\cdots\mathtt{a}_n$ is a child of α, or -1 otherwise. Since the underlying cycle-joining tree satisfies Assumption 2, if such an $\mathtt{x}$ exists then it is unique. In practice, the function will concern itself with the acceptable range of α as per the construction of concatenation trees.

Algorithm 1 Traversing a concatenation tree $\mathcal{T}$ in RCL order rooted at α with change index c

```
1: procedure RCL($\alpha = \mathtt{a}_1 \cdots \mathtt{a}_n$, $c$, $\ell$)
2:     for $i \leftarrow c + \ell$ to $n$ do          ▷ Visit right-children
3:         $x \leftarrow$ CHILD($\alpha, i$)
4:         if $\mathtt{x} \neq -1$ then  RCL($\mathtt{a}_1 \cdots \mathtt{a}_{i-1}\mathtt{x}\mathtt{a}_{i+1} \cdots \mathtt{a}_n$, $i$, $\ell$)
5:     $p \leftarrow$ period of $\alpha$
6:     PRINT($\mathtt{a}_1 \cdots \mathtt{a}_p$)
7:     for $i \leftarrow 1$ to $c - 1 + \ell$ do      ▷ Visit left-children
8:         $\mathtt{x} \leftarrow$ CHILD($\alpha, i$)
9:         if $\mathtt{x} \neq -1$ then  RCL($\mathtt{a}_1 \cdots \mathtt{a}_{i-1}\mathtt{x}\mathtt{a}_{i+1} \cdots \mathtt{a}_n$, $i$, $\ell$)
```

The running time of the Algorithm 1 depends on how efficiently the function CHILD(α, i) can be computed for each index i. Provided each call to CHILD(α, i) uses at most $O(n)$ space, the overall algorithm will also require $O(n)$ space assuming α is passed by reference (or stored globally) and restored appropriately after each recursive call.

▶ **Theorem 5.** *Let $\mathcal{T}$ be a concatenation rooted at α with change index c. The sequence resulting from a call to* RCL(α, c, ℓ) *is generated in $O(1)$-amortized time per symbol if (i) at each recursive step the work required by all calls to* CHILD(α, i) *is $O((t+1)n)$, where t is the number of α's children, and (ii) the number of nodes in $\mathcal{T}$ that are periodic is less than some constant times the number of nodes that are aperiodic.*

Proof. The work done at each recursive step is $O(n)$ plus the cost associated to all calls to CHILD(α, i). If condition (i) is satisfied, then the work can be amortized over the t children if $t \geq 1$, or onto the node itself if there are no children. Thus, each recursive node is the result of $O(n)$ work. By condition (ii), the total number of symbols output will be proportional to n times the number of nodes. Thus, each symbol is output in $O(1)$-amortized time. ◀

5 Applications

In this section we highlight how our concatenation tree framework can be applied to a variety of interesting objects including de Bruijn sequences.

5.1 Universal cycles for shorthand permutations

A *shorthand* permutation is a length $n-1$ prefix of some permutation $\mathtt{p}_1\mathtt{p}_2 \cdots \mathtt{p}_n$. Let $\mathbf{SP}(n)$ denote the set of shorthand permutations of order n. For example, $\mathbf{SP}(3) = \{12, 13, 21, 23, 31, 32\}$. An *inversion* of a shorthand permutation $\mathtt{p}_1\mathtt{p}_2 \cdots \mathtt{p}_{n-1}$ is an ordered pair $(\mathtt{p}_i, \mathtt{p}_j)$ such that $i < j$ and $\mathtt{p}_i > \mathtt{p}_j$. Note that $\mathbf{SP}(n)$ is closed under rotation. The necklace classes of $\mathbf{SP}(n)$ can be joined into a PCR-based cycle-joining tree via the following parent rule [23], where each cycle representative is a necklace. If σ contains 1, its representative is the rotation that begins with 1; otherwise, it is the rotation beginning with 2.

Parent rule for cycle-joining: Let r denote the root $12 \cdots (n-1)$. Let σ denote a non-root node where $\mathtt{z}$ is the missing symbol. If $\mathtt{z} = n$, let j denote the smallest index such that there exists an inversion $(\mathtt{p}_i, \mathtt{p}_j)$ for some $i < j$; otherwise let j denote the index of $\mathtt{z} + 1$. Then

$$\mathrm{par}(\sigma) = \mathtt{p}_1 \cdots \mathtt{p}_{j-1}\mathtt{z}\mathtt{p}_{j+1} \cdots \mathtt{p}_{n-1}.$$

Let $\mathbb{T}$ be the cycle-joining tree derived from the above parent rule and let $\mathcal{T}_{perm} = \mathrm{concat}(\mathbb{T}, n, \mathit{left})$. Figure 8 illustrates $\mathcal{T}_{perm}$ for $n = 5$; an RCL traversal of this tree produces the following universal cycle RCL$(\mathcal{T}_{perm})$ of length $5! = 120$:

$$\begin{gathered}1234\ 1235\ 1243\ 1254\ 1325\ 1435\ 2435\ 1425\ 1324\ 1354\ 2354\ 1253\ 1245\ 1342\ 1352\\ 1432\ 1543\ 2543\ 1542\ 1532\ 1453\ 2453\ 1423\ 1524\ 1534\ 2534\ 1523\ 1452\ 1345\ 2345.\end{gathered}$$

Let $\sigma' = \mathtt{q}_1 \cdots \mathtt{q}_{n-1}$ denote the cycle representative of σ. Let j be the smallest index such that there exists an inversion $(\mathtt{q}_i, \mathtt{q}_j)$ in σ' for some $i < j$; let $\mathrm{inv}(\sigma)$ denote $\mathtt{q}_j$, or 0 if no such j exists. For example, if $\sigma = 23146$ then $\sigma' = 14\underline{62}3$, $j = 4$, and $\mathrm{inv}(\sigma) = 2$. If $\sigma = 23461$, then $\mathrm{inv}(\sigma) = 0$.

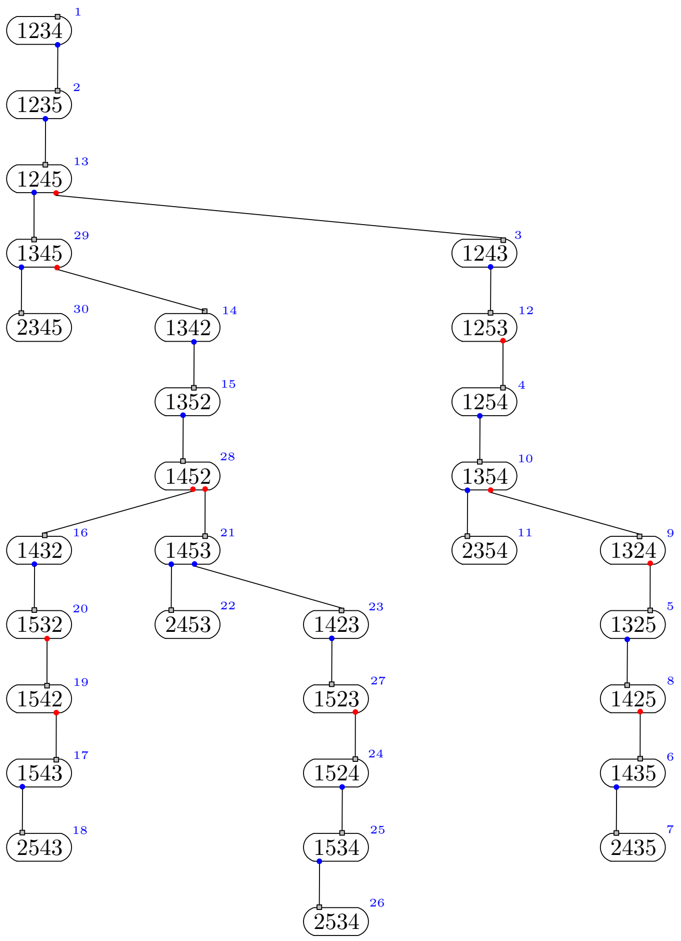


Figure 8 Concatenation tree $\mathcal{T}_{perm}$ for shorthand permutations when $n = 5$ illustrating the RCL order.

The following UC-successor is derived from the above parent rule for cycle-joining the cycle representatives of shorthand permutations. Naïvely, it can be computed in $O(n)$ time using $O(n)$ space; however with some optimizations, it can be applied to generate successive symbols in $O(1)$-amortized time [23].

UC-successor: Let $\sigma = \mathtt{p}_1\mathtt{p}_2\cdots\mathtt{p}_{n-1}$. Then:

$$f_{perm}(\sigma) = \begin{cases} \mathtt{z} & \text{if } (\mathtt{z} = n \text{ and } \mathtt{p}_1 = \text{inv}(\sigma)) \text{ or } (\mathtt{p}_1 = n \text{ and } \mathtt{z} = \text{inv}(\mathtt{z}\mathtt{p}_2\cdots\mathtt{p}_{n-1}));\\ \mathtt{z} & \text{if } \mathtt{z} \in \{\mathtt{p}_1 - 1, \mathtt{p}_1 + 1\};\\ \mathtt{p}_1 & \text{otherwise.} \end{cases}$$

The children of a cycle representative σ can easily be determined in a single scan of σ in $O(n)$ time. Thus, Theorem 3 and Theorem 5 imply the following result.

▶ **Theorem 6.** $\text{RCL}(\mathcal{T}_{perm})$ *is a universal cycle for* $\mathbf{SP}(n)$ *with shift rule* f_{perm}*. Moreover,* $\text{RCL}(\mathcal{T}_{perm})$ *can be constructed in* $O(1)$*-amortized time per symbol using* $O(n)$ *space.*

Efficient concatenation constructions of universal cycles for shorthand permutations are known [40]; however (i) there is no clear connection between their construction and corresponding UC-successor and (ii) they do not have underlying PCR-based cycle-joining trees. A next step for the framework presented in this paper is to extend it to other underlying feedback functions that will ultimately demystify the relationship between the concatenation constructions and their corresponding UC-successors in [40, 26].

5.2 Universal cycles for weak orders

Recall that $\mathbf{W}(n)$ denotes the set of weak orders of order n; it is closed under rotation.[6] The first construction of a universal cycle for $\mathbf{W}(n)$ defined the upcoming PCR-based cycle-joining tree, where the cycle-representatives (nodes) are the lex-smallest representatives [46]. Let $\omega = \mathtt{w}_1\mathtt{w}_2\cdots\mathtt{w}_n$ denote a string in $\mathbf{W}(n)$. Let $n_\omega(i)$ denote the number of occurrences of the symbol i in ω. Let $\mathbf{W_1}(n)$ denote the set of all weak orders of order n with no repeating symbols other than perhaps 1.

Parent rule for cycle-joining: Let r denote the root 1^n. Let $\omega = \mathtt{w}_1\mathtt{w}_2\cdots\mathtt{w}_n$ denote a non-root node. If $\omega \in \mathbf{W_1}(n)$, let j denote the index of the symbol $n_\omega(1)+1$ and let $\mathtt{x}=1$; otherwise let j be the largest index containing a repeated (non-1) symbol and let $\mathtt{x} = \mathtt{w}_j + n_\omega(\mathtt{w}_j) - 1$. Then

$$\text{par}(\omega) = \mathtt{w}_1\cdots\mathtt{w}_{j-1}\mathtt{x}\mathtt{w}_{j+1}\cdots\mathtt{w}_n.$$

Let $\mathbb{T}$ be the cycle-joining tree derived from the above parent rule and let $\mathcal{T}_{weak} = \text{concat}(\mathbb{T}, n, \textit{left})$. Figure 2 illustrates both the cycle-joining tree and $\mathcal{T}_{weak}$ for $n=4$. The following UC-successor is derived $\mathbb{T}$; It can be computed in $O(n)$ time [46].

UC-successor: Let $\omega = \mathtt{w}_1\mathtt{w}_2\cdots\mathtt{w}_n$ and let $\mathtt{p}$ be the largest symbol in ω less than $\mathtt{w}_1$. Let j be the smallest index such that $\mathtt{w}_j\cdots\mathtt{w}_n\mathtt{w}_1\cdots\mathtt{w}_{j-1}$ is a cycle representative and let j' be the smallest index such that $\mathtt{w}_{j'}\cdots\mathtt{w}_n\mathtt{p}\mathtt{w}_2\cdots\mathtt{w}_{j'-1}$ is a cycle representative. Then:

$$f_{weak}(\omega) = \begin{cases} 1 & \text{if } \omega \in \mathbf{W_1}(n) \text{ and } \mathtt{w}_1 = n_\omega(1)+1, \\ n_\omega(1) & \text{if } \omega \in \mathbf{W_1}(n) \text{ and } \mathtt{w}_1 = 1, \\ \mathtt{w}_1 + n_\omega(\mathtt{w}_1) - 1 & \text{if } \omega \notin \mathbf{W_1}(n),\ n_\omega(\mathtt{w}_1) > 1,\ \mathtt{w}_1 > 1,\ (n_\omega(\mathtt{w}_i) = 1 \text{ or } \mathtt{w}_i = 1) \text{ for all } 2 \le i \le j-1, \\ \mathtt{p} & \text{if } n_\omega(\mathtt{w}_1) = 1,\ \mathtt{w}_1 > 1,\ \mathtt{p} \neq 1,\ (\mathtt{w}_i \neq \mathtt{p} \text{ and } (n_\omega(\mathtt{w}_i) = 1 \text{ or } \mathtt{w}_i = 1)) \text{ for all } 2 \le i \le j'-1; \\ \mathtt{w}_1 & \text{otherwise.} \end{cases}$$

Though not as trivial as with shorthand permutations, the t children of a given cycle representative has been implemented in $O((t+1)n)$ time. Once proved, then Theorem 5 implies that $\text{RCL}(\mathcal{T}_{weak})$ can be constructed in $O(1)$-amortized time per symbol using $O(n)$ space.

▶ **Theorem 7.** *$\text{RCL}(\mathcal{T}_{weak})$ is a universal cycle for $\mathbf{W}(n)$ with shift rule f_{weak}.*

5.3 Orientable sequences

An *orientable sequence* is a universal cycle for a set $\mathbf{S} \subseteq \{0,1\}^n$ such that if $\mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n \in \mathbf{S}$, then its reverse $\mathtt{a}_n\cdots\mathtt{a}_2\mathtt{a}_1 \notin \mathbf{S}$. Thus, $\mathbf{S}$ does not contain palindromes. Orientable sequences do not exist for $n < 5$, and a maximal length orientable sequence for $n=5$ is 001011. Somewhat surprisingly, the maximal length of binary orientable sequences are known only for $n = 5, 6, 7$. Orientable sequences were introduced in [11] with applications related to robotic position sensing. They established upper and lower bounds for their maximal length; the lower bound is based on the *existence* of a PCR-based cycle-joining tree, though no construction of such a tree was provided. Subsequently, a recursive construction based on Lempel's lift constructs orientable sequences with length that are asymptotic to a trivial upper bound, though less than the established lower bound [36].

Motivated by the concatenation framework, a cycle-joining tree $\mathbb{T}$ for necklaces that are lexicographically smaller than any of their reversed rotations was recently discovered. It applies three of the four "simplest" parent rules defined earlier for PCR-based cycle joining trees. As an example, the concatenation tree in Figure 9 can be traversed to produce the following orientable sequence for $n=9$ of length 126:

111001011 011001011 110011011 110001011 100101011 100011011 101011011
100001001 110001001 010001001 100001011 001001011 000101011 000001011.

[6] Weak orders of order n are counted by the ordered Bell or Fubini numbers (OEIS A000670).

Preliminary findings indicate that the RCL sequence produced by traversing the corresponding concatenation tree can be computed in $O(1)$-amortized time per bit using polynomial space; it generates the longest known orientable sequences efficiently. This result will be presented in a follow up to this work. It will be the first known efficient construction to attain the lower bound derived in [11].

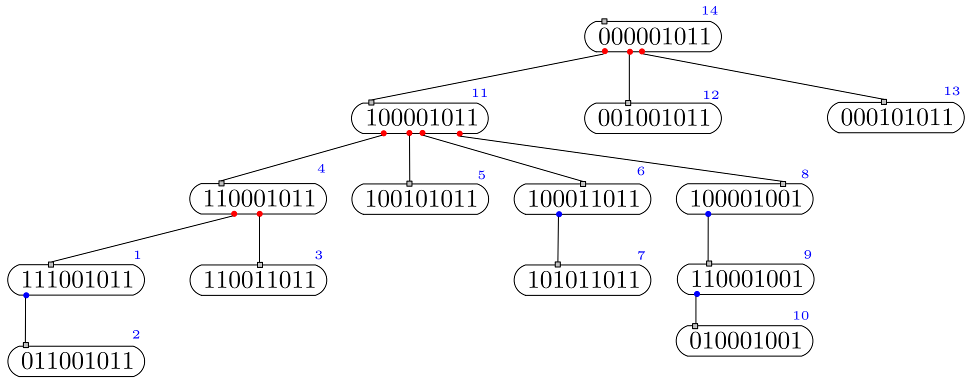


Figure 9 A right concatenation tree that yields an orientable sequence for $n = 9$. The nodes correspond to all necklace classes, where the necklace of each class is strictly less than all reversed rotations. The small blue numbers indicate the order a node is visited in RCL-order.

5.4 De Bruijn sequences

Recall the cycle-joining trees $\mathbb{T}_1$, $\mathbb{T}_2$, $\mathbb{T}_3$, and $\mathbb{T}_4$ for binary stings of length n. These trees can be generalized to larger alphabets following the theory in [23]. For instance, the parent rule used to create $\mathbb{T}_1$ can be generalized to "increment the last non-$(k{-}1)$" where the alphabet is $\Sigma = \{0, 1, \dots, k-1\}$. Though we do not provide details here, most of the following discussion can also be applied to arbitrary sized alphabets.

Recall Corollary 4 and the concatenation constructions producing DB sequences. To obtain the ultimate goal of constructions that run in $O(1)$-amortized time, we apply Algorithm 1 and Theorem 5 to demonstrate that the children of a node can be efficiently computed, in an amortized sense. To simplify our analysis, we define an array *child*[ ] initialized to 0's such that the function IsChild(α, j) simply returns *child*$[j]$. Since the depth of each tree is at most n, this results in $O(n^2)$ space when recursion is applied. Without using the array, the algorithm could be implemented using $O(n)$ space. In our running time analysis, we implicitly apply the fact that number of periodic necklaces is not more than the number of aperiodic necklaces (of order n) [38, Lemma 4.4]. We will apply a boolean function IsNecklace(α) that returns whether or not α is a necklace; it can be implemented to run in $O(n)$ time [5]. For each of the four upcoming concatenation trees, computing the lone child of the root is trivial. Thus, we focus on a non-root node $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n$ with change index c. In each case, the indices being updated are clearly within the *acceptable range* of α.

Let $\mathcal{T}_1 = \text{concat}(\mathbb{T}_1, 1, \textit{right})$. The following remark is easily verified by applying the definition of a necklace.

▶ Remark 8. If j is the largest index such that $\mathtt{a}_j = 0$ (the last 0), then $\mathtt{a}_1\cdots\mathtt{a}_{j-1}\mathbf{1}\mathtt{a}_{j+1}\cdots\mathtt{a}_n$ is a necklace.

Recall the root of $\mathcal{T}_1$ is 1^n. Since the change index of the root in $\mathcal{T}_1$ is set to $c = 1$, Remark 8 implies that all the representatives in $\mathcal{T}_1$ will be necklaces. Thus, c is the largest index in α such that $\mathtt{a}_c = 0$ and any children of α must come from flipping a 1 at an index greater than c and less than n (in order for the index to hold the "last 0" in the resulting necklace). Thus, applying the following pseudocode will accurately determine the children of α.

```
for $j \leftarrow c+1$ to $n-1$ do   $child[j] \leftarrow$ IsNecklace($\mathtt{a}_1\cdots\mathtt{a}_{j-1}\mathbf{0}\mathtt{a}_{j+1}\cdots\mathtt{a}_n$)
```

▶ **Theorem 9.** $\text{RCL}(\mathcal{T}_1)$ *is a DB sequence with shift rule* pcr_1 *that can be constructed in $O(1)$-amortized time using $O(n^2)$ space.*

Proof. The fact that $\mathrm{RCL}(\mathcal{T}_1)$ is a DB sequence with shift rule pcr_1 follows immediately from Theorem 3. Interestingly, there can be many failed tests to IsNecklace. We ignore the single case when $j = c+1$ as it uses only $O(n)$ time. Suppose $\beta = \mathtt{a}_1 \cdots \mathtt{a}_{j-1}\mathbf{0}\mathtt{a}_{j+1} \cdots \mathtt{a}_n$ is not a necklace where $j > c+1$, and let i denote the first index in α such that $\mathtt{i} = 1$. Since α is a necklace there does not exist a substring 0^i in β since $j > c+1$. Thus, clearly $\gamma = 0\mathtt{a}_1 \cdots \mathtt{a}_{j-1}\mathbf{0}\mathtt{a}_{j+1} \cdots \mathtt{a}_{n-1}$ is a necklace; it begins with a prefix 0^i. The mapping of each failed β to its corresponding γ is clearly 1-1. Thus, the total number of failed calls to IsNecklace is bounded by the number of nodes in the tree. Amortizing this work over these nodes and applying Theorem 5 implies that $\mathrm{RCL}(\mathcal{T}_1)$ can be generated in $O(1)$-amortized time. ◀

Recall pcr_1 is a DB-successor of the "Granddaddy" sequence presented in [19]. Thus, an immediate consequence of the above theorem (and straightforward observation) is that an RCL traversal of $\mathcal{T}_1$ corresponds to visiting the necklaces in $\mathbf{N}_2(n)$ in lexicographic order.

Let $\mathcal{T}_2 = \mathrm{concat}(\mathbb{T}_2, n, \mathit{left})$. The following remark is easily verified by applying the definition of a necklace.

▶ Remark 10. If j is the smallest index such that $\mathtt{a}_j = 1$ (the first 1), then $\mathtt{a}_1 \cdots \mathtt{a}_{j-1}\mathbf{0}\mathtt{a}_{j+1} \cdots \mathtt{a}_n$ is a necklace.

Recall the root of $\mathcal{T}_2$ is 0^n. Since the change index of the root in $\mathcal{T}_1$ is set to $c = n$, Remark 10 implies that all the representatives in $\mathcal{T}_2$ are be necklaces. Thus, c is the smallest index in α such that $\mathtt{a}_c = 1$ and any the children of α must come from flipping a 1 at an index less than c (so the resulting index becomes the "first 1" in the resulting necklace). Clearly a longest run of 0's appears as a prefix in a necklace. Thus, the "first 1" in a child of α cannot come until an index after the maximum of $\lfloor c/2 \rfloor$ and the longest run of 0's. Similarly, the resulting prefix of 0's must be at least z, the maximal length run of 0's in α after the initial run of 0's. Thus, applying the following pseudocode will accurately determine the children of α.

```
$z \leftarrow$ max length run of 0's in $\mathtt{a}_{c+1} \cdots \mathtt{a}_n$
**for** $j \leftarrow \mathrm{MAX}(z, \lfloor c/2 \rfloor)) + 1$ **to** $c - 1$ **do**   $\mathit{child}[j] \leftarrow \mathrm{IsNecklace}(\mathtt{a}_1 \cdots \mathtt{a}_{j-1}\mathbf{1}\mathtt{a}_{j+1} \cdots \mathtt{a}_n)$
```

▶ **Theorem 11.** $\mathrm{RCL}(\mathcal{T}_2)$ *is a DB sequence with shift rule* pcr_2 *that can be constructed in $O(1)$-amortized time using $O(n^2)$ space.*

Proof. The fact that $\mathrm{RCL}(\mathcal{T}_2)$ is a DB sequence with shift rule pcr_2 follows immediately from Theorem 3. To analyze the running time, observe that there will be at most one failed call to IsNecklace – the first one. After that, flipping $\mathtt{a}_i$ from a 0 to a 1 will leave the maximal run of 0's in the resulting string at $\mathtt{a}_1 \cdots \mathtt{a}_{j-1}$. Thus, Theorem 5 implies $\mathrm{RCL}(\mathcal{T}_2)$ can be generated in $O(1)$-amortized time. ◀

Recall pcr_2 is a DB-successor of the "Grandmama" sequence presented in [12]. Thus, an immediate consequence of the above theorem (and straightforward observation) is that an RCL traversal of $\mathcal{T}_2$ corresponds to visiting the necklaces in $\mathbf{N}_2(n)$ in co-lexicographic order.

Let $\mathcal{T}_3 = \mathrm{concat}(\mathbb{T}_3, 1, \mathit{right})$. The following remark follows by applying the fact that a longest run of 0's must appear as a prefix in a necklace.

▶ Remark 12. If j is the smallest index such that $\mathtt{a}_j = 1$ (the first 1), then $\mathtt{a}_{i+1} \cdots \mathtt{a}_n\mathtt{a}_1 \cdots \mathtt{a}_{i-1}\mathbf{1}$ is a necklace where $\mathtt{a}_i = 0$ only if $i < j$.

Recall the root of $\mathcal{T}_3$ is 0^n. From the parent rule, $\mathtt{a}_{c+1} \cdots \mathtt{a}_n\mathtt{a}_1 \cdots \mathtt{a}_c$ is a necklace representative of α; $\mathtt{a}_c$ corresponds to the "last 1". Since the change index of the root is set to $c = 1$, α will start with a 1 and $\mathtt{a}_{c+1} \cdots \mathtt{a}_n = 0^{n-c}$ because any children of α must come from flipping a 0 at an index greater than c from Remark 12. Since $\mathtt{a}_{c+1} \cdots \mathtt{a}_n$ represents a longest run of 0's in α, flipping any 0 outside this range will not result in a necklace. Since flipping a 0 at index j after c must be the "last 1" in a necklace for it to be a child, it must that $\mathtt{a}_{j+1} \cdots \mathtt{a}_n$ is a longest run of 0's in the resulting string. Thus, applying a similar logic as with $\mathcal{T}_2$, the following pseudocode will accurately determine the children of α.

```
$z \leftarrow$ max length run of 0's in $\mathtt{a}_1 \cdots \mathtt{a}_{c-1}$
**for** $j \leftarrow c + 1$ **to** $\mathrm{MIN}(n - \lfloor (n-c)/2 \rfloor, n - z))$ **do**   $\mathit{child}[j] \leftarrow \mathrm{IsNecklace}(\mathtt{a}_{j+1} \cdots \mathtt{a}_n\mathtt{a}_1 \cdots \mathtt{a}_{j-1}\mathbf{1})$
```

▶ **Theorem 13.** $\mathrm{RCL}(\mathcal{T}_3)$ *is a DB sequence with shift rule* pcr_3 *that can be constructed in* $O(1)$*-amortized time using* $O(n^2)$ *space.*

Proof. The fact that $\mathrm{RCL}(\mathcal{T}_3)$ is a DB sequence with shift rule pcr_3 follows immediately from Theorem 3. The analysis follows the same logic as the proof of Theorem 11: there will be at most one failed call to ISNECKLACE (the last one). ◀

It is relatively straightforward to see that $\mathrm{RCL}(\mathcal{T}_3)$ is the same concatenation sequence as the one presented in [21], thus answering an unproved claim regarding its equivalence to the UC-successor pcr_3. It is the first $O(1)$-amortized time algorithm for this sequence which we can immediately apply to cutdown de Bruijn sequences discussed in Section 5.5.

Let $\mathcal{T}_4 = \mathrm{concat}(\mathbb{T}_4, n, \mathit{left})$. It remains an open question as to whether or not $\mathrm{RCL}(\mathcal{T}_4)$ can be generated in $O(1)$-amortized time; experimental evidence indicate that it is possible with some more intricate arguments as to exactly which indices need to be tested for children. The following theorem follows immediately from Theorem 3.

▶ **Theorem 14.** $\mathrm{RCL}(\mathcal{T}_4)$ *is a DB sequence with shift rule* pcr_4.

Let the *weight* of a string denote the sum of its symbols. To conclude this section, we consider specific subtrees of $\mathcal{T}_1, \mathcal{T}_2$, and $\mathcal{T}_3$ whose nodes have bounded weight. The running time analysis of the above concatenation trees also applies to these cases. Let T_1 be the subtree of $\mathcal{T}_1$ where the nodes have weight less than or equal to some constant w. Let T_2, T_3 be the subtrees of $\mathcal{T}_2$ and $\mathcal{T}_3$, respectively, where the nodes have weight greater than or equal to some constant w.

▶ **Corollary 15.** $\mathrm{RCL}(T_1)$, $\mathrm{RCL}(T_2)$, *and* $\mathrm{RCL}(T_3)$ *are universal cycle for* $\mathbf{S}_{T_1}$, $\mathbf{S}_{T_3}$, *and* $\mathbf{S}_{T_3}$, *respectively, that can be generated in* $O(1)$*-amortized time using* $O(n^2)$ *space, with corresponding UC-successors* pcr_1, pcr_2, *and* pcr_3.

Universal cycles for strings with bounded weight are considered in [44].

5.5 Cut-down de Bruijn sequences

A *cut-down de Bruijn sequence* of length m is a universal cycle for a set $\mathbf{S} \subseteq \Sigma^n$, where $|\mathbf{S}| = m$. They are known to exist for all m, and their most efficient construction is based on a PCR-based cycle joining tree [14, 7]; it requires $O(n)$ time per symbol using $O(n)$ space. In the binary case, the underlying tree described in [7] is a subtree of $\mathbb{T}_3$ with bounded weight. Thus, Corollary 4 can be applied along with added theory to *cut-out* small cycles, to construct the same cut-down de Bruijn sequence in $O(1)$-amortized time per bit using $O(n)$-space. This result will be presented in a follow up to this work. A recent result attains the same time complexity, but requires exponential space [37].

6 Proof of Theorem 3

Our proof relies on properties exhibited between successive nodes in an RCL traversal. To aid the discussion, we define the following relationships given a node x in a BOT.

- A *right-descendant* of x is a node obtained by traversing down zero or more right-children.
- A *left-descendant* of x is a node obtained by traversing down zero or more left-children.
- The *rightmost left-descendant* of x is the node obtained by repeatedly traversing down the last left-child as long as one exists.
- The *leftmost right-descendant* of x is the node obtained by repeatedly traversing down the first right-child as long as one exists.

Note that a node is its own leftmost right-descendent if it has no right-children. Similarly, a node is its own rightmost left-descendent if it has no left-children. The following remark details the three cases for when two nodes from a BOT appear consecutively in RCL order; they are illustrated in Figure 10.

▶ Remark 16. If a bifurcated ordered tree has RCL traversal $\ldots, x, y, \ldots$, then one of the following three cases holds:

(a) x is an ancestor of y: y is the leftmost right-descendant of x's first left-child;
(b) x is a descendant of y: x is the rightmost left-descendent of y's last right-child;
(c) x and y are descendants of a common ancestor a (other than x and y): x is the rightmost left-descendant and y is the leftmost right-descendant of consecutive left-children or right-children of a.

Moreover, if the traversal sequence is cyclic (i.e., x is last in the ordering and y is first), there are three additional cases:

(d) x is an ancestor of y: x is the root and y is its leftmost right-descendant;
(e) x is a descendant of y: y is the root and x is its rightmost left-descendant;
(f) x and y are descendants of a common ancestor a (other than x and y): x is the rightmost left-descendant of the root, and y is the leftmost right-descendant of the root.

The three cases provided for cyclic sequences are stated in a way to convince the reader that all options are considered; however, they can be collapsed to the single case (f) if we allow the common ancestor a to be x or y.

Theorem 3 restated: Let $\mathbb{T}$ be a PCR-based cycle-joining tree satisfying Assumption 2. Let $\mathcal{T}_1 = \text{concat}(\mathbb{T}, c, \textit{left})$ and let $\mathcal{T}_2 = \text{concat}(\mathbb{T}, c, \textit{right})$. Then

- $\text{RCL}(\mathcal{T}_1)$ is a universal cycle for $\mathbf{S}_{\mathbb{T}}$ with shift rule f_1, and
- $\text{RCL}(\mathcal{T}_2)$ is a universal cycle for $\mathbf{S}_{\mathbb{T}}$ with shift rule f_2.

Let $\mathcal{T}$ represent either $\mathcal{T}_1$ or $\mathcal{T}_2$, and let $\alpha_1, \alpha_2, \ldots, \alpha_t$ denote the nodes of $\mathcal{T}$ as they are visited in RCL order; $U = \text{RCL}(\mathcal{T}) = \text{ap}(\alpha_1)\,\text{ap}(\alpha_2)\cdots\text{ap}(\alpha_t)$. The proof of Theorem 3 is by induction on t. We specify whether $\mathcal{T}$ is a left-concatenation tree $\mathcal{T}_1$ or a right-concatenation tree $\mathcal{T}_2$ only when necessary. In the base case case when $t = 1$, the result is immediate; $\mathcal{T}$ contains a single cycle and in each case the UC-successor simplifies to $f(\mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n) = \mathtt{a}_1$. Suppose $t > 1$. Let $\alpha_j = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n$ denote an arbitrary leaf of $\mathcal{T}$ with change index c. Let $\beta_1 = \mathtt{a}_1\cdots\mathtt{a}_{c-1}$, $\mathtt{y} = \mathtt{a}_c$, and $\beta_2 = \mathtt{a}_{c+1}\cdots\mathtt{a}_n$. Then $\alpha_j = \beta_1\mathtt{y}\beta_2$ and its parent is $\beta_1\mathtt{y}'\beta_2$ for some $\mathtt{y}' \in \Sigma$; the corresponding nodes in $\mathbb{T}$ are joined via the conjugate pair $(\mathtt{y}\beta_1\beta_2, \mathtt{y}'\beta_1\beta_2)$. If $\mathcal{T} = \mathcal{T}_1$, let $\mathtt{x} = \mathtt{y}'$; if $\mathcal{T} = \mathcal{T}_2$, let $\mathtt{x} = \text{first}(\mathtt{y}'\beta_1\beta_2)$ with respect to $\mathbb{T}$. Let $\mathcal{T}'$ denote the concatenation tree obtained by removing α_j from $\mathcal{T}$. Similarly, let $\mathbb{T}'$ denote the cycle-joining tree $\mathbb{T}$ with the leaf corresponding to α_j removed. Let $U_1 = \text{ap}(\alpha_{j+1})\cdots\text{ap}(\alpha_t)\,\text{ap}(\alpha_1)\cdots\text{ap}(\alpha_{j-1})$ denote a rotation of $\text{RCL}(\mathcal{T}')$. By induction, U_1 is a universal cycle for $\mathbf{S}' = \mathbf{S} - [\alpha_j]$. Let $U_2 = \text{ap}(\alpha_j)$; it is a universal cycle for $[\alpha_j]$. Note that U_1 contains $\mathtt{x}\beta_2\beta_1$ and U_2 contains $\mathtt{y}\beta_2\beta_1$. The following claim will be proved later.

▷ Claim 17. U_1 (considered cyclically) has prefix β_1 and suffix $\mathtt{x}\beta_2$.

Let $U_1' = \cdots\mathtt{x}\beta_2\beta_1$ and let $U_2' = \cdots\mathtt{y}\beta_2\beta_1$ be rotations of U_1 and U_2, respectively. Then by Theorem 1 and Claim 17, U_1 and U_2 can be joined via the conjugate pair $(\mathtt{x}\beta_2\beta_1, \mathtt{y}\beta_2\beta_1)$ to produce universal cycle $U_1'U_2'$, which is a rotation of U_1U_2, for $\mathbf{S}$. Since U_1U_2 is a rotation of U, the latter is also a universal cycle for $\mathbf{S}$.

Clearly $f_1 = f_2$, with respect to the single PCR cycle $[\alpha_j]$; both functions are UC-successors for U_2. Suppose $\mathcal{T} = \mathcal{T}_1$. From the induction hypothesis, f_1 (with respect to $\mathbb{T}'$) is a UC-successor for U_1. Since the two cycles U_1 and U_2 were joined via the conjugate pair $(\mathtt{x}\beta_2\beta_1, \mathtt{y}\beta_2\beta_1)$ to obtain U; the successors of only these two strings are altered. By the joining, the successor of $\mathtt{y}\beta_2\beta_1$ becomes the successor of $\mathtt{x}\beta_2\beta_1$ in U_1 which is precisely $g_1(\mathtt{y}\beta_2\beta_1)$ with respect to $\mathbb{T}$. The successor of $\mathtt{x}\beta_2\beta_1$ is $\mathtt{y}$, which is the same as $g_1(\mathtt{x}\beta_2\beta_1)$ with respect to $\mathbb{T}$. Thus, f_1 (with respect to $\mathbb{T}$) is a UC-successor for U. A similar argument applies for $\mathcal{T} = \mathcal{T}_2$.

It remains to prove Claim 17.

6.0.1 Proof of Claim 17

Recall, $U_1 = \text{ap}(\alpha_{j+1})\cdots\text{ap}(\alpha_t)\,\text{ap}(\alpha_1)\cdots\text{ap}(\alpha_{j-1})$, $\alpha_j = \beta_1\mathtt{y}\beta_2$, and $\text{ancestor}(\alpha_j) = \beta_1\mathtt{x}\beta_2$. We first demonstrate α_{j+1} has prefix β_1 and α_{j-1} has suffix $\mathtt{x}\beta_2$ by tracing the six cases from Remark 16 as illustrated in Figure 10. Assume operations of the indices are taken modulo t, i.e., $\alpha_0 = \alpha_t$ and $\alpha_{t+1} = \alpha_1$. For each $1 \le i \le t$, let c_i denote the change index of α_i.

▷ Claim 18. α_{j+1} has prefix β_1.

Proof. Since α_j is a leaf we need only consider the four cases (b)(c)(e)(f) from Remark 16 following the notation from Figure 10. Let $x = \alpha_j$ and $y = \alpha_{j+1}$.

(b) r_m clearly has prefix β_1. Since the change index of r_m is less than or equal to c_j, and r_m only differs from its parent y at its change index, y must also have the prefix β_1.

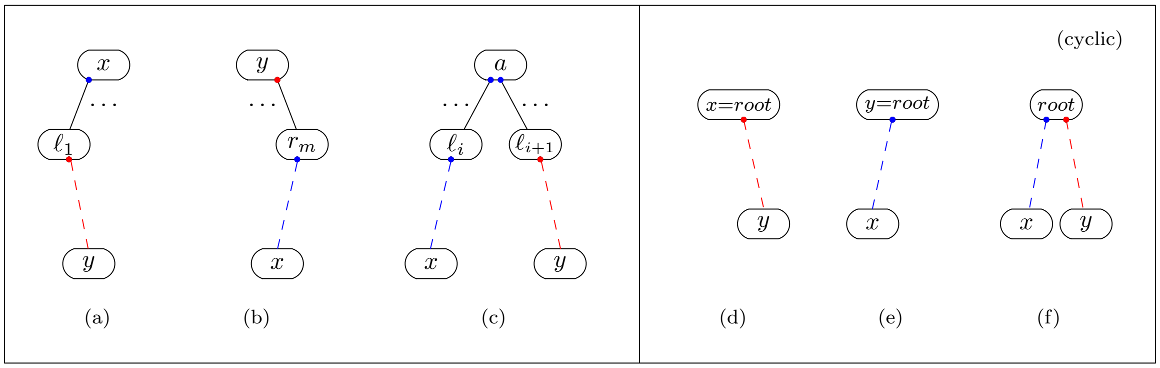


Figure 10 Illustrating the six cases outlined in Remark 16 for when y follows x in an RCL traversal. The final three cases hold when the traversal sequence is considered to be cyclic (i.e., x comes last and y comes first). In these images, ℓ_i and r_i refer to the ith left and right-child of their parent, respectively, and r_m refers to the last right-child of its parent. Dashed lines indicate leftmost right-descendants (red) and rightmost left-descendants (blue).

(c) ℓ_i clearly has prefix β_1. The change index of ℓ_i is strictly less than the change index of ℓ_{i+1} and the two nodes differ only at those two indices. Thus, β_1 is a prefix of ℓ_{i+1}. Since y can only differ from ℓ_{i+1} in indices between the change index of ℓ_{i+1} and c_{j+1}, it must also have the prefix β_1.
(e) Trivial.
(f) All the nodes on the path from x up to the root and down to y must have change index greater than or equal to c_j. Thus each node, including y will have prefix β_1.

◀

▷ Claim 19. α_{j-1} has suffix $\mathtt{x}\beta_2$.

Proof. Since α_j is a leaf, we consider the four cases (a)(c)(d)(f) from Remark 16 following notation from Figure 10. Let $x = \alpha_{j-1}$ and $y = \alpha_j$. Recall the parent of α_j is $\beta_1\mathtt{y}'\beta_2$. First, suppose $\mathcal{T}$ is a *left* concatenation tree, recalling $\mathtt{x} = \mathtt{y}'$.

(a) If $\ell_1 = y$, the result is immediate. Suppose $\ell_1 \neq y$. From the definition of $\mathtt{x}$, ℓ_1 has suffix $\mathtt{x}\beta_2$ and change index strictly less than c_j. Since ℓ_1 differs from its parent x only at its change index, x must also have suffix $\mathtt{x}\beta_2$.
(c) If $\ell_{i+1} = y$, then it is already established that its parent a has suffix $\mathtt{x}\beta_2$. Otherwise, ℓ_{i+1} has suffix $\mathtt{x}\beta_2$ and change index less than c_j, which means that a again has suffix $\mathtt{x}\beta_2$. Since the change index of ℓ_i is less than the change index of ℓ_{i+1}, clearly x also has suffix $\mathtt{x}\beta_2$.
(d) Follows since $\mathcal{T}$ is a left concatenation tree.
(f) Let α_r be the root of $\mathcal{T}$. Clearly, α_r has suffix $\mathtt{x}\beta_2$ and $c_r < c_j$. Thus, x also will have suffix $\mathtt{x}\beta_2$.

Now suppose $\mathcal{T}$ is a *right* concatenation tree recalling $\mathtt{x} = \text{first}(\mathtt{y}'\beta_1\beta_2)$. This implies that all nodes on the path from $\beta_1\mathtt{x}\beta_2$ to $y = \alpha_j$ have change index c_j and the change index of $\beta_1\mathtt{x}\beta_2$ is not equal to c_j.

(a) If $\ell_1 = y$, then the change index of ℓ_1 is strictly less than the change index of x and the result follows as $\mathtt{x} = \mathtt{y}'$. Suppose $\ell_1 \neq y$. If the change index of ℓ_1 is strictly less than c_j, then by the definition of $\mathtt{x}$, ℓ_1 has suffix $\mathtt{x}\beta_2$. Thus, clearly x also has suffix $\mathtt{x}\beta_2$. Otherwise, the change index of ℓ_1 must be equal to c_j, and since it is a left-child of x, the change index of x is not equal to c_j. Thus, by the definition of $\mathtt{x}$, x will be precisely $\beta_1\mathtt{x}\beta_2$.
(c) Recall this covers two cases where the children of a can be either be both left-children or both right-children. In either case, the change index of a can not be the same as the change index for $\ell i + 1$. Thus, following the same argument from (a), the node a will have suffix $\mathtt{x}\beta_2$. Since the change index of ℓ_i is less than the change index of ℓ_{i+1}, clearly x also has suffix $\mathtt{x}\beta_2$.
(d) Follows since $\mathcal{T}$ is a right concatenation tree.
(f) Let α_r be the root of $\mathcal{T}$. Clearly, α_r has suffix $\mathtt{x}\beta_2$ and $c_r \leq c_j$. Since all left descendants of the root will have change index strictly less than c_r, it follows that x also will have suffix $\mathtt{x}\beta_2$.

◀

If α_{j-1} and α_{j+1} are aperiodic, then by Claim 18 and Claim 19 we are done. If $t = 2$, then we are also done since U_1 is considered cyclically. It remains to be considered the cases where α_{j-1} or α_{j+1} is periodic and $t > 2$. These cases apply the "acceptable range".

Case: α_{j+1} is periodic

Suppose α_{j+1} has period p and acceptable range $kp+1, \ldots, kp+p$. To handle this case, we demonstrate the following: (i) $c_j \leq kp+p$, and (ii) $ap(\alpha_{j+1})^{k+1}$ is a prefix of U_1. The first point implies that β_1 is a prefix of $ap(\alpha_{j+1})^{k+1}$ since β_1 is a prefix of α_{j+1} from Claim 18. This, in combination with the second point, implies β_1 is a prefix of U_1.

Proof of (i). Since α_j is a leaf, we step through cases (b), (c), (e), and (f) from Remark 16 following notation from Figure 10 where $x = \alpha_j$ and $y = \alpha_{j+1}$. (b) The change index for r_m must be less than or equal to $kp+p$, and because α_j is a left descendant of r_m, c_j must be less than or equal to the change index of r_m. Thus, $c_j \leq kp+p$. (c) c_j is less than or equal to the change index of ℓ_i, which is less than the change index of ℓ_{i+1}, which is less than or equal to c_{j+1}. Thus, $c_j < c_{j+1} \leq kp+p$. (e) α_j is a left-descendant of α_{j+1} so clearly $c_j < c_{j+1} \leq kp+p$. (f) c_j is less than or equal to the change index of the root, which is less than or equal to c_{j+1}. Thus, $c_j < c_{j+1} \leq kp+p$. ◀

Proof of (ii). We start by proving a general claim for consecutive nodes in the RCL traversal of $\mathcal{T}$ and use that to prove a stronger claim. For convenience, consider the notation used in acceptable ranges to be independent of earlier definitions.

▷ Claim 20. If α_i is periodic with period p and acceptable range $kp+1, \ldots, kp+p$, then $ap(\alpha_i)^k$ is a prefix of α_{i+1}.

Proof. If α_i is not an ancestor of α_{i+1}, the inequality $kp < c_i$ and Claim 18 together imply $ap(\alpha_i)^k$ is a prefix of α_{i+1}. It remains to consider cases (a) and (d) from Remark 16 where $x = \alpha_i$ is an ancestor of $y = \alpha_{i+1}$. For case (a), y is the leftmost right-descendent of x's first left-child ℓ_1. Since x is periodic, the change index of ℓ_1 is in α_i's acceptable range; it is greater than kp. y is a right descendant of ℓ_1 and thus $c_{i+1} > kp$, which means y differs from ℓ_1 only in indices greater than kp. For (d) clearly y differs only in indices greater than or equal to c_i, which means $c_{i+1} > kp$. Thus, for each case, $ap(\alpha_i)^k$ is a prefix of α_{i+1}. ◀

▷ Claim 21. If α_i is periodic with period p and acceptable range $kp+1, \ldots, kp+p$, then $ap(\alpha_i)^{k+1}$ is a prefix of $ap(\alpha_i)\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_{i-1})$, which is a rotation of $\mathrm{RCL}(\mathcal{T})$, considered cyclically.

Proof. Note that $|ap(\alpha_i)^{k+1}| \leq n$. The proof is by induction on the number of nodes t. If $t = 1$, the result is trivial. Suppose the claim holds for any tree with less than $t > 1$ nodes. Let $\mathcal{T}$ have t nodes and let α_i be a leaf node of $\mathcal{T}$. If there are no periodic nodes, we are done. Otherwise, we first consider α_i, then all other periodic nodes in $\mathcal{T}$.

Suppose α_i is periodic with period p and acceptable range $kp+1, \ldots, kp+p$. From Claim 20, $ap(\alpha_i)^k$ is a prefix of α_{i+1}. If α_{i+1} is aperiodic, then we are done. Suppose, then, that α_{i+1} is periodic with period p' and acceptable range $k'p'+1, \ldots, k'p'+p'$. Let $\mathcal{T}'$ be the tree resulting from $\mathcal{T}$ when α_i is removed. It follows from (i) that $kp < c_i \leq k'p'+p'$, which implies $ap(\alpha_i)^k$ is a prefix of $ap(\alpha_{i+1})^{k'+1}$. Additionally, since $\mathcal{T}'$ has less than t nodes and α_{i+1} is periodic, $ap(\alpha_{i+1})^{k'+1}$ is a prefix of $ap(\alpha_{i+1})\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_{i-1})$ by our inductive assumption. Therefore, $ap(\alpha_i)^{k+1}$ is a prefix of $ap(\alpha_i)ap(\alpha_{i+1})\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_{i-1})$.

Now consider α_{i-1}. If it is aperiodic, then by induction, the claim clearly holds for all periodic nodes in $\mathcal{T}'$. Thus, assume α_{i-1} is periodic. By showing that $ap(\alpha_{i-1})ap(\alpha_i)\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_{i-2})$ has the desired prefix, then repeating the same arguments will prove the claim holds for every other periodic node in $\mathcal{T}'$. Let α_{i-1} have period p'' and acceptable range $k''p''+1, \ldots, k''p''+p''$. If α_i is aperiodic, Claim 20 implies that $ap(\alpha_{i-1})^{k''}$ is a prefix of $\alpha_i = \mathrm{ap}(\alpha_i)$ and thus the claim holds for α_{i-1}. If α_i is periodic with period p and acceptable range $kp+1, \ldots, kp+p$, we already demonstrated that $ap(\alpha_i)^{k+1}$ is a prefix of $ap(\alpha_i)\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_{i-1})$. From Claim 20, $ap(\alpha_{i-1})^{k''}$ is a prefix of α_i. Note that (i) and its proof handles cases (b)(c)(e)(f) from Remark 16 implying that $c_{i-1} < kp+p$ for these cases. Since α_{i-1} is not necessarily a leaf, we must also consider (a) and (d). In both cases, clearly $k''p'' < c_i$. Either way, $k''p'' < kp+p$, which means $ap(\alpha_{i-1})^{k''}$ is a prefix of $ap(\alpha_i)^{k+1}$. Thus, $ap(\alpha_{i-1})^{k''+1}$ is a prefix of $ap(\alpha_{i-1})ap(\alpha_i)\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_{i-2})$. ◀

Applying Claim 21 to α_{j+1} gives the desired result. ◀

Case: α_{j-1} is periodic

The proof mirrors the case for α_{j+1}. Suppose α_{j-1} has period p and acceptable range $kp+1,\dots,kp+p$. To handle this case we demonstrate the following: (i) $c_j > kp$, and (ii) $ap(\alpha_{j-1})^{q-k}$ is a suffix of U_1. The first point implies that β_2 is a suffix of $ap(\alpha_{j-1})^{q-k}$ since β_2 is a suffix of α_{j-1} from Claim 19. This, in combination with the second point, implies β_2 is a suffix of U_1.

Proof of (i). Since α_j is a leaf, we step through cases (a), (c), (d), and (f) from Remark 16 following notation from Figure 10 where $x = \alpha_{j-1}$ and $y = \alpha_j$. (a) By the acceptable range, the change index for ℓ_1 must be greater than kp. Because α_j is a right descendant of ℓ_1, c_j must be greater than or equal to the change index of ℓ_1. Thus, $c_j > kp$. (c) c_{j-1} is less than or equal to the change index of ℓ_i, which is less than the change index of ℓ_{i+1}, which is less than or equal to c_j. Thus, $kp < c_{j-1} < c_j$. (d) α_j is a right-descendant of α_{j-1} so clearly $kp < c_{j-1} < c_j$. (f) c_{j-1} is less than or equal to the change index of the root, which is less than c_j. Thus, $kp < c_{j-1} < c_j$. ◀

Proof of (ii). As with the prefix section, we start by proving a general claim for consecutive nodes in the RCL traversal of $\mathcal{T}$ and use that to prove a stronger claim. For convenience, consider the notation used in acceptable ranges to be independent of earlier definitions.

▷ Claim 22. If α_i is periodic with period p and acceptable range $kp+1,\dots,kp+p$, then $ap(\alpha_i)^{q-k-1}$, where $q = n/p$, is a suffix of α_{i-1}.

Proof. If α_i is not an descendant of α_{i-1}, the inequality $c_i \le kp+p$ and Claim 19 together imply $ap(\alpha_i)^k$ is a prefix of α_{i-1}. It remains to consider cases (b) and (e) from Remark 16 where $y = \alpha_i$ is an ancestor of $x = \alpha_{i-1}$. For case (b), x is the rightmost left-descendent of y's last right-child r_m. Since y is periodic, the change index of r_m is in α_i's acceptable range; it is less than or equal to $kp+p$. x is a left descendant of r_m and thus $c_{i-1} \le kp+p$, which means x differs from r_m only in indices less than or equal to $kp+p$. For (e) clearly x differs only in indices less than or equal to c_i, which means $c_{i-1} \le kp+p$. Thus, for each case, $ap(\alpha_i)^{q-k-1}$ is a suffix of α_{i-1}. ◀

▷ Claim 23. If α_i is periodic with period p and acceptable range $kp+1,\dots,kp+p$, then $ap(\alpha_i)^{n/p-k}$, is a suffix of $ap(\alpha_{i+1})\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_i)$, which is a rotation of RCL($\mathcal{T}$), considered cyclically.

Proof. Let $q = n/p$. Note that $|ap(\alpha_i)^{q-k}| \le n$. The proof is by induction on t. If $t = 1$, the result is trivial. Suppose the claim holds for any tree with less than $t > 1$ nodes. Let $\mathcal{T}$ have t nodes and let α_i be a leaf node of $\mathcal{T}$. If there are no periodic nodes, we are done. Otherwise, we first consider α_i, then all other periodic nodes in $\mathcal{T}$.

Suppose α_i is periodic with period p and acceptable range $kp+1,\dots,kp+p$. From Claim 22, $ap(\alpha_i)^{q-k-1}$ is a suffix of α_{i-1}. If α_{i-1} is aperiodic, then we are done. Suppose, then, that α_{i-1} is periodic with period p' and acceptable range $k'p'+1,\dots,k'p'+p'$. Let $\mathcal{T}'$ be the tree resulting from $\mathcal{T}$ when α_i is removed. It follows from (i) that $k'p' < c_i \le kp+p$, or $n-kp-p < n-k'p'$, which implies $ap(\alpha_i)^{q-k-1}$ is a suffix of $ap(\alpha_{i-1})^{q'-k'}$, where $q' = n/p'$. Additionally, since $\mathcal{T}'$ has less than t nodes and α_{i-1} is periodic, $ap(\alpha_{j-1})^{q'-k'}$ is a suffix of $ap(\alpha_{i+1})\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_{i-1})$ by our inductive assumption. Therefore, $ap(\alpha_i)^{q-k}$ is a suffix of $ap(\alpha_{i+1})\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_{i-1})ap(\alpha_i)$.

Now consider α_{i+1}. If it is aperiodic, then by induction the claim clearly holds for all periodic nodes in $\mathcal{T}'$. Thus, assume α_{i+1} is periodic. By showing $ap(\alpha_{i+2})\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_i)ap(\alpha_{i+1})$ has the desired suffix, then repeating the same arguments will prove the claim holds for every other periodic node in $\mathcal{T}'$. Let α_{i+1} have period p'' and acceptable range $k''p''+1,\ldots,k''p''+p''$. If α_i is aperiodic, Claim 22 implies that $ap(\alpha_{i+1})^{q''-k''-1}$ is a suffix of $\alpha_i = \mathrm{ap}(\alpha_i)$ and thus the claim holds for α_{i+1}. If α_i is periodic with period p and acceptable range $kp+1,\ldots,kp+p$, we already demonstrated that $ap(\alpha_i)^{q-k}$ is a suffix of $ap(\alpha_{i+1})\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_i)$. From Claim 22, $ap(\alpha_{i+1})^{q''-k''-1}$ is a suffix of α_i. Note that (i) and its proof handles cases (a)(c)(d)(f) from Remark 16 implying that $c_{i+1} > kp$ for these cases. Since α_{i+1} is not necessarily a leaf, we must also consider (b) and (e). In both cases, clearly $c_i \leq k''p''+p''$. Either way, $kp < k''p''+p''$, which means $\mathrm{ap}(\alpha_{i+1})^{q''-k''-1}$ is a suffix of $\mathrm{ap}(\alpha_i)^{q-p}$. Thus, $\mathrm{ap}(\alpha_{i+1})^{q''-k''}$ is a suffix of $ap(\alpha_{i+2})\cdots ap(\alpha_t)ap(\alpha_1)\cdots ap(\alpha_i)ap(\alpha_{i+1})$. ◀

Applying Claim 23 to α_{j-1} gives the desired result. ◀

References

1 De Bruijn sequence and universal cycle constructions (2023). http://debruijnsequence.org.

2 The On-Line Encyclopedia of Integer Sequences, published electronically at https://oeis.org, 2010, sequence A006013.

3 ALHAKIM, A. A simple combinatorial algorithm for de Bruijn sequences. *The American Mathematical Monthly 117*, 8 (2010), 728–732.

4 BANKEVICH, A., BZIKADZE, A. V., KOLMOGOROV, M., ANTIPOV, D., AND PEVZNER, P. A. Multiplex de bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature Biotechnology 40*, 7 (Jul 2022), 1075–1081.

5 BOOTH, K. S. Lexicographically least circular substrings. *Inform. Process. Lett. 10*, 4/5 (1980), 240–242.

6 BROCKMAN, G., KAY, B., AND SNIVELY, E. On universal cycles of labeled graphs. *Electronic Journal of Combinatorics 17* (200), R4.

7 CAMERON, B., GÜNDOĞAN, A., AND SAWADA, J. Cut-down de Bruijn sequences, https://arxiv.org/abs/2205.02815, 2022.

8 CHUNG, F., DIACONIS, P., AND GRAHAM, R. Universal cycles for combinatorial structures. *Discrete Mathematics 110*, 1-3 (1992), 43–59.

9 COMPEAU, P. E., PEVZNER, P. A., AND TESLER, G. How to apply de bruijn graphs to genome assembly. *Nature biotechnology 29*, 11 (2011), 987–991.

10 CURTIS, D., HINES, T., HURLBERT, G., AND MOYER, T. Near-universal cycles for subsets exist. *SIAM Journal on Discrete Mathematics 23*, 3 (2009), 1441–1449.

11 DAI, Z.-D., MARTIN, K., ROBSHAW, B., AND WILD, P. Orientable sequences. In *Cryptography and Coding III (M.J.Ganley, ed.)* (1993), Oxford University Press, pp. 97–115.

12 DRAGON, P. B., HERNANDEZ, O. I., SAWADA, J., WILLIAMS, A., AND WONG, D. Constructing de Bruijn sequences with co-lexicographic order: The k-ary Grandmama sequence. *European Journal of Combinatorics 72* (2018), 1–11.

13 ELDERT, C., GRAY, H., GURK, H., AND RUBINOFF, M. Shifting counters. *AIEE Trans. 77* (1958), 70–74.

14 ETZION, T. An algorithm for generating shift-register cycles. *Theoret. Comput. Sci. 44*, 2 (1986), 209–224.

15 ETZION, T. Self-dual sequences. *Journal of Combinatorial Theory, Series A 44*, 2 (1987), 288 – 298.

16 ETZION, T., AND LEMPEL, A. Construction of de Bruijn sequences of minimal complexity. *IEEE Transactions on Information Theory 30*, 5 (September 1984), 705–709.

17 FREDRICKSEN, H. Generation of the Ford sequence of length 2^n, n large. *J. Combin. Theory Ser. A 12*, 1 (1972), 153–154.

18 FREDRICKSEN, H., AND KESSLER, I. Lexicographic compositions and de Bruijn sequences. *J. Combin. Theory Ser. A 22*, 1 (1977), 17 – 30.

19 FREDRICKSEN, H., AND MAIORANA, J. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discrete Math. 23* (1978), 207–210.

20 GABRIC, D., AND SAWADA, J. A de Bruijn sequence construction by concatenating cycles of the complemented cycling register. In *Combinatorics on Words - 11th International Conference, WORDS 2017, Montréal, QC, Canada, September 11-15, 2017, Proceedings* (2017), pp. 49–58.

21 GABRIC, D., AND SAWADA, J. Constructing de Bruijn sequences by concatenating smaller universal cycles. *Theoretical Computer Science 743* (2018), 12–22.

22 GABRIC, D., SAWADA, J., WILLIAMS, A., AND WONG, D. A framework for constructing de Bruijn sequences via simple successor rules. *Discrete Mathematics 241*, 11 (2018), 2977–2987.

23 Gabric, D., Sawada, J., Williams, A., and Wong, D. A successor rule framework for constructing k -ary de Bruijn sequences and universal cycles. *IEEE Transactions on Information Theory 66*, 1 (2020), 679–687.

24 Hierholzer, C. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen 6* (1873), 30–32.

25 Higgns, Z., Kelley, E Sieben, B., and Godbole, A. Universal and near-universal cycles of set partitions. *Electronic Journal of Combinatorics 22*, 4 (2015), P4.48.

26 Holroyd, A. E., Ruskey, F., and Williams, A. Shorthand universal cycles for permutations. *Algorithmica 64*, 2 (2012), 215–245.

27 Huang, Y. A new algorithm for the generation of binary de Bruijn sequences. *J. Algorithms 11*, 1 (1990), 44–51.

28 Hurlbert, G. On universal cycles for k-subsets of an n-set. *SIAM Journal on Discrete Mathematics 7*, 4 (1994), 598–604.

29 Jackson, B. W. Universal cycles of k-subsets and k-permutations. *Discrete mathematics 117*, 1 (1993), 141–150.

30 Jansen, C. J. A., Franx, W. G., and Boekee, D. E. An efficient algorithm for the generation of DeBruijn cycles. *IEEE Transactions on Information Theory 37*, 5 (Sep 1991), 1475–1478.

31 Johnson, J. R. Universal cycles for permutations. *Discrete Mathematics 309*, 17 (2009), 5264–5270.

32 Kak, S. Yamatarajabhanasalagam: an interesting combinatoric sutra. *Indian Journal of History of Science 35*, 2 (2000), 123–128.

33 Knuth, D. E. *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms*. Addison-Wesley Professional, 2011.

34 Mahadik, K., Wright, C., Kulkarni, M., Bagchi, S., and Chaterji, S. Scalable genome assembly through parallel de Bruijn graph construction for multiple k-mers. *Scientific Reports 9*, 1 (Oct 2019), 14882.

35 Martin, M. H. A problem in arrangements. *Bull. Amer. Math. Soc. 40*, 12 (1934), 859–864.

36 Mitchell, C. J., and Wild, P. R. Constructing orientable sequences. *IEEE Trans. Inf. Theory 68*, 7 (2022), 4782–4789.

37 Nellore, A., and Ward, R. Arbitrary-length analogs to de Bruijn sequences. In *33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022)* (Dagstuhl, Germany, 2022), H. Bannai and J. Holub, Eds., vol. 223 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 9:1–9:20.

38 Ruskey, F., and Sawada, J. An efficient algorithm for generating necklaces with fixed density. *SIAM Journal on Computing 29*, 2 (1999), 671–684.

39 Ruskey, F., Sawada, J., and Williams, A. De Bruijn sequences for fixed-weight binary strings. *SIAM J. Discrete Math. 26*, 2 (2012), 605–617.

40 Ruskey, F., and Williams, A. An explicit universal cycle for the $(n$-$1)$-permutations of an n-set. *ACM Trans. Algorithms 6*, 3 (July 2010), 1–12.

41 Sala, E., Sawada, J., and Alhakim, A. Efficient constructions of the Prefer-same and Prefer-opposite de Bruijn sequences. *CoRR abs/2010.07960* (2020).

42 Sawada, J., and Williams, A. Constructing the first (and coolest) fixed-content universal cycle. *Algorithmica 85*, 6 (Jun 2023), 1754–1785.

43 Sawada, J., Williams, A., and Wong, D. Universal cycles for weight-range binary strings. In *Combinatorial Algorithms - 24th International Workshop, IWOCA 2013, Rouen, France, July 10-12, 2013, LNCS 8288* (2013), pp. 388–401.

44 Sawada, J., Williams, A., and Wong, D. The lexicographically smallest universal cycle for binary strings with minimum specified weight. *Journal of Discrete Algorithms 28* (2014), 31–40.

45 Sawada, J., Williams, A., and Wong, D. A surprisingly simple de Bruijn sequence construction. *Discrete Math. 339* (2016), 127–131.

46 Sawada, J., and Wong, D. Efficient universal cycle constructions for weak orders. *Discrete Mathematics 343*, 10 (2020), 112022.

47 Song, L., Geng, F., Gong, Z.-Y., Chen, X., Tang, J., Gong, C., Zhou, L., Xia, R., Han, M.-Z., Xu, J.-Y., Li, B.-Z., and Yuan, Y.-J. Robust data storage in DNA by de Bruijn graph-based de novo strand assembly. *Nature Communications 13*, 1 (Sep 2022), 5361.

48 Stein, S. K. The mathematician as an explorer. *Scientific American 204*, 5 (1961), 148–161.

49 Stein, S. K. *Mathematics: the man-made universe*. Courier Corporation, 2013.

50 Van Nooten, B. Binary numbers in indian antiquity. *Journal of Indian philosophy* (1993), 31–50.

51 Wong, D. A new universal cycle for permutations. *Graph. Comb. 33*, 6 (Nov. 2017), 1393–1399.

52 Zerbino, D. R., and Birney, E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res. 18*, 5 (May 2008), 821–829.

A Enumeration of bifurcated ordered trees (BOTs)

In OEIS A006013 [2], the enumeration sequence for BOTs computed in the following C program is equivalent to the enumeration sequence for the following objects (among others):

- enumerates pairs of ternary trees,
- permutations avoiding 213 in the classical sense which can be realized as labels on an increasing strict binary tree with $2n-1$ nodes,
- connections of $2n-2$ points labeled $1, 2, ..., 2n-2$ in a line with 0 or more noncrossing arcs above the line such that each maximal contiguous sequence of isolated points has even length, and
- projective dependency trees with n nodes.

Discovering a mapping between BOTs and these objects is the topic of current research.

```c
 //===========================================================================
// Dynamic Programming Approach to Enumerating Bifurcated Ordered Trees (BOTs)
// The sequence matches all entries to the sequence https://oeis.org/A006013
//===========================================================================

#include<stdio.h>
long int T[100], total, a[100];

void BOT(int t, int n, int m) {
long int i,temp=0;

    if (n > 0) {
        for (i=1; i<=n; i++)  {  a[t] = i;  BOT(t+1,n-i, m); }
    }
    else if (m > 0)  {
        for (i=1; i<=m; i++)  {  a[t] = i;  BOT(t+1,n, m-i); }
    }
    else {
        temp = T[ a[1] ];
        for (i=2;i<=t-1; i++) temp = temp *  T[ a[i] ];
        total += temp;
    }
}
//-------------------------------------
int main() {
int i,j;

    T[1] = 1; T[2] = 2;     // Base cases
    for (i=3; i<=23; i++) {
        total = 0;
        for (j=0; j<=i-1; j++) BOT(1,j,i-1-j);
        T[i] = total;
    }
    for (i=1; i<=23; i++) printf("T[%d] = %ld\n", i, T[i]);
}
```