A Framework for Constructing De Bruijn Sequences Via Simple Successor Rules

Daniel Gabric  Joe Sawada  Aaron Williams  Dennis Wong

July 5, 2018

Abstract

We present a simple framework for constructing de Bruijn sequences, and more generally, universal cycles, via successor rules. The framework is based on the often used method of joining disjoint cycles. It generalizes four previously known de Bruijn sequence constructions and is applied to derive three new and simple de Bruijn sequence constructions. Four of the constructions apply the pure cycling register and three apply the complemented cycling register. The correctness of each new construction is easily proved using the new framework. Each of the three new de Bruijn sequence constructions can be generated in $O(n)$-time per bit using $O(n)$-space.

1 Introduction

Let $\mathbf{B}(n)$ denote the set of binary strings of length $n$. A de Bruijn sequence is a binary string of length $2^n$ that when considered cyclicly, contains every string in $\mathbf{B}(n)$ as a substring. More generally, given a subset $\mathcal{S}$ of $\mathbf{B}(n)$, a universal cycle for $\mathcal{S}$ is a binary string of length $|\mathcal{S}|$ that when considered cyclicly contains every string in $\mathcal{S}$ as a substring. In this paper, we present a framework for constructing de Bruijn sequences, and more generally, universal cycles, by applying the often used approach of joining disjoint cycles. We apply our framework to generalize:

- the construction of the lexicographically least de Bruijn sequence given by Fredricksen [10] that we call the Granddaddy, a description first used by Knuth [20],
- the construction by Dragon et al. [4] which was named the Grandmama in [5], and
- two instances from the construction by Jansen et al. [19] that we label J1 and J2.

One of the instances in the generic construction in [19] is equivalent to the binary construction given by Sawada et al. [26]$^{1}$ Each of these constructions applies a successor rule that can be used to generate each bit in the sequence from the previous $n$ bits. In addition to these four sequences, we apply the framework to find three new successor rules based de Bruijn constructions. Examples of these seven sequences for $n = 6$ are given in Table 1. The first four constructions are based on the pure cycling register (PCR) and necklaces. The final three are based on the complemented cycling register (CCR) and co-necklaces. Previously, Huang [18] and Etzion [7] presented constructions using the CCR, but their resulting successor rules are significantly more complex than the ones presented in this paper.

In addition to these successor rule based constructions, there are several other known constructions. There are three well-known greedy approaches: the prefer-smallest (or equivalently, prefer-largest) [22, 9], the prefer-same [6, 11], and the prefer-opposite [1]. Each of these approaches is generalized in [2], but they all

---

$^{1}$The fact that the binary version of the construction in [26] is equivalent to a special case of the construction in [19] was not previously observed.
Table 1: Eight de Bruijn sequences for \( n = 6 \), each based on a different successor rule.

<table>
<thead>
<tr>
<th>Method</th>
<th>de Bruijn sequence for ( n = 6 ) starting with 000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCR1 (Granddaddy)</td>
<td>0000001000010011001011110110110111011111111111111111</td>
</tr>
<tr>
<td>PCR2 (Grandmama)</td>
<td>0000001001010101010011001000101001101001100110111111111111</td>
</tr>
<tr>
<td>PCR3 (J1)</td>
<td>000000111111011100111001100110111001101100110011011011011</td>
</tr>
<tr>
<td>PCR4</td>
<td>000000111111011010010011011101010110010100010111100111000110001</td>
</tr>
<tr>
<td>CCR1</td>
<td>000000111111000110111001100101011010100100010011101100001011111</td>
</tr>
<tr>
<td>CCR2</td>
<td>0000001111110001001110110011000010111101001010110101000110111001</td>
</tr>
<tr>
<td>CCR3 (J2)</td>
<td>0000001001110110001010110101001011110100001100110111001000111111</td>
</tr>
</tbody>
</table>

have the drawback of requiring an exponential amount of memory\(^2\). Interestingly, not only is the Granddaddy equivalent to the prefer-smallest greedy construction, but it is also equivalent to a very efficient necklace concatenation approach [13, 14, 23]. Another necklace concatenation construction is based on cool-lex order in [24]. A construction based on lexicographic compositions [12] is equivalent to the prefer-same construction up to \( n = 6 \). For \( n > 7 \), the two sequences are conjectured to share a long prefix. The sequences for these other constructions for \( n = 6 \) is shown in Table 2. There are also known classes of de Bruijn sequences [8, 21], but they lack the simplicity of the constructions discussed in this paper.

Table 2: Three more de Bruijn sequences for \( n = 6 \) based on different constructions.

<table>
<thead>
<tr>
<th>Method</th>
<th>de Bruijn sequence for ( n = 6 ) starting with 000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefer-opposite</td>
<td>0000001010100101101011101000100110110010000110001110011110111111</td>
</tr>
<tr>
<td>Prefer-same / lex-comp</td>
<td>0000001111111000100011000101111011100111010011011001001010110101</td>
</tr>
<tr>
<td>Cool-lex</td>
<td>0000001100011110111111001110100110110101011001011100001010001001</td>
</tr>
</tbody>
</table>

Observe that of the ten de Bruijn sequences presented in Tables 1 and 2, no two are equivalent. We consider two sequences to be equivalent if one sequence can be obtained from the other by some combination of rotation, reversal, or symbol remapping (swapping 0s and 1s).

In the next section, we present background on necklaces and co-necklaces along with the pure and complemented cycling registers. In Section 3, we present our universal cycle construction framework. Then in Section 4, we present the seven successor rules with proofs that apply the framework. In Section 5 we provide a short discussion on the implementation of the successor rules, proving that each of the new de Bruijn sequence constructions can be generated in \( O(n) \)-time per bit. We conclude with avenues for future research in Section 6.

## 2 Necklaces, Co-necklaces, and Cycling Registers

A necklace is the lexicographically smallest string in an equivalence class of strings under rotation. Let \( N(n) \) denote the set of all length \( n \) necklaces. Let \( \text{Neck}(\alpha) \) denote the set containing \( \alpha \) along with each of its rotations. The necklace representative of \( \text{Neck}(\alpha) \) is the unique necklace in \( \text{Neck}(\alpha) \). For example, \( \text{Neck}(01010) = \{01010, 10101, 01010, 10010, 01010\} \) and its necklace representative is 00101.

Let \( x \in \{0, 1\} \) and let \( \overline{x} \) denote the bitwise complement of \( x \). Given a string \( \alpha = a_1a_2\cdots a_n \) let \( \overline{\alpha} \) denote the complement of \( \alpha \) defined to be \( \overline{a_1}\overline{a_2}\cdots\overline{a_n} \). We say \( \alpha \) is a co-necklace if \( \alpha\overline{\alpha} \) is a necklace. Let \( \text{coNeck}(\alpha) \) denote the set containing all length \( n \) subcircles of the circular string \( \alpha\overline{\alpha} \). The co-necklace representative of \( \text{coNeck}(\alpha) \) is the unique co-necklace in \( \text{coNeck}(\alpha) \). For example, \( \text{coNeck}(00111) = \)

\(^2\)See [27] for a new XNOR greedy de Bruijn sequence construction.
{00111, 01111, 11111, 11110, 11100, 10000, 00000, 00001, 00011}, and its co-necklace representative is 00000.

Let $\alpha = a_1a_2\cdots a_n$. Closely related to necklaces and co-necklaces are the pure cycling register (PCR) and the complemented cycling register (CCR) defined on all binary strings to be PCR$(\alpha) = a_1$ and CCR$(\alpha) = \overline{a_1}$ respectively. The PCR partitions $B(n)$ into the equivalence classes $\{\text{Neck}(\alpha) \mid \alpha \in N(n)\}$ via the function $F(\alpha) = a_2a_3\cdots a_n\text{PCR}(\alpha)$. Similarly, the CCR partitions $B(n)$ into the equivalence classes $\{\text{coNeck}(\alpha) \mid \alpha$ is a co-necklace of length $n\}$ [16] via the function $F'(\alpha) = a_2a_3\cdots a_n\text{CCR}(\alpha)$.

### Example 1
(Left) A partition of $B(4)$ using the PCR. The first string in each part is a necklace. (Right) A partition of $B(4)$ using the CCR. The first string in each part is a co-necklace.

<table>
<thead>
<tr>
<th></th>
<th>0000</th>
<th>0001</th>
<th>0011</th>
<th>0101</th>
<th>0111</th>
<th>1111</th>
<th>0000</th>
<th>0010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>0110</td>
<td>1010</td>
<td>1110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>1100</td>
<td></td>
<td>1101</td>
<td>0111</td>
<td>1011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1001</td>
<td></td>
<td>1011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1111</td>
<td>1101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1110</td>
<td>1010</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1100</td>
<td>0100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
<td>1001</td>
<td></td>
</tr>
</tbody>
</table>

### 3. A Framework to Derive Successor Rules that Construct Universal Cycles

Our framework for constructing universal cycles hinges on three definitions: a UC-successor, a UC-partition, and a spanning sequence. For each definition and subsequent result, assume that $S$ is a non-empty subset of $B(n)$.

**Definition 3.1** A function $f : B(n) \to \{0, 1\}$ is a UC-successor of $S$ if there exists a universal cycle $U$ for $S$ such that each string $\alpha \in S$ is followed by the bit $f(\alpha)$ in $U$.

In this definition the domain of $f$ is defined to be $B(n)$, not $S$, to simplify the proof of our upcoming main theorem. As an example, $f(a_1a_2a_3a_4) = \overline{a_1}$ is a UC-successor for $S = \{0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000\}$ since $U = 0001111$ is a universal cycle for $S$. In the special case where $S = B(n)$ we say a UC-successor is a *de Bruijn successor*.

Now we formalize our approach for joining smaller universal cycles into larger universal cycles. We join $m$ smaller cycles together using $m-1$ binary strings of length $n-1$. Each such string will appear as a substring in two distinct universal cycles. In particular, $x\beta$ will be in one cycle and $\overline{x}\beta$ will be in another cycle, where $x \in \{0, 1\}$ and $\beta \in B(n-1)$. This approach is somewhat similar to Hierholzer’s algorithm [17] for constructing an Euler cycle in a related transition graph. The main advance of this framework is that when applied effectively (as in the next section) it requires only $O(n)$ space compared to the exponential space required to store the graph.

**Definition 3.2** A partition of $S$ into subsets $S_1, S_2, \ldots, S_m$ is called a UC partition with respect to $f$ if $f$ is a UC-successor for each $S_i$ where $i \in \{1, 2, \ldots, m\}$.

**Definition 3.3** Let $S_1, S_2, \ldots, S_m$ be an ordered partition of $S$. A sequence $\beta_2, \beta_3, \ldots, \beta_m$ of unique strings in $B(n-1)$ is a spanning sequence of the partition if for each $i \in \{2, 3, \ldots, m\}$ there is an $x \in \{0, 1\}$ such that $x\beta_i \in S_i$ and $\overline{x}\beta_i \in S_j$ for some $j < i$.

The following lemma, which follows from [25, Lemma 3], describes how to join two universal cycles together.
Lemma 3.4 If $S_1, S_2$ is an ordered UC partition of $S$ with respect to $f$ and $\beta$ is a spanning sequence (consisting of one string) of the partition then the following function $f' : B(n) \rightarrow \{0, 1\}$ is a UC-successor for $S$:

$$f'(\alpha) = \begin{cases} \overline{f(\alpha)} & \text{if } \alpha \in \{0\beta, 1\beta\}; \\ f(\alpha) & \text{otherwise.} \end{cases}$$

Repeated application of Lemma 3.4 leads to our main result.

Theorem 3.5 If $S_1, S_2, \ldots, S_m$ is an ordered UC partition of $S$ with respect to $f$ and $\beta_2, \beta_3, \ldots, \beta_m$ is a spanning sequence of the partition, then the following function $g : B(n) \rightarrow \{0, 1\}$ is a UC-successor for $S$:

$$g(\alpha) = \begin{cases} \overline{f(\alpha)} & \text{if } \alpha \in \{0\beta_2, 1\beta_2, 0\beta_3, 1\beta_3, \ldots, 0\beta_m, 1\beta_m\}; \\ f(\alpha) & \text{otherwise.} \end{cases}$$

Proof. If $m = 1$ then $g = f$ is a UC-successor for $S$ by definition. If $m > 1$, then observe that $S_1, S_2$ is a UC partition of $S_1 \cup S_2$ with respect to $f$ and $\beta_2$ is a spanning sequence for this partition. By applying Lemma 3.4 the following function $f'$ is a UC-successor for $S_1 \cup S_2$:

$$f'(\alpha) = \begin{cases} \overline{f(\alpha)} & \text{if } \alpha \in \{0\beta_2, 1\beta_2\}; \\ f(\alpha) & \text{otherwise.} \end{cases}$$

Observe that this function is also a UC-successor for each of $S_3, S_4, \ldots, S_m$. Thus, we can repeat this process for each $S_1 \cup S_2 \cup \cdots \cup S_{i-1}$ and $S_i$ with spanning sequence $\beta_i$ for $i = 3, 4, \ldots, m$. At the end of these $m - 1$ applications of Lemma 3.4, the resulting function is equivalent to $g$, and moreover it is a UC-successor for $S_1 \cup S_2 \cup \cdots \cup S_m = S$. \hfill \square

3.1 An Example

By applying Theorem 3.5, the following steps can be used to derive a universal cycle for a subset $S$ of $B(n)$.

1. Select a (simple) function $f$ that can be used to partition $S$ into subsets $S_1, S_2, \ldots, S_m$ such that $f$ is a successor rule to construct a universal cycle for each $S_i$.
2. Define an ordering of $S_1, S_2, \ldots, S_m$ so that for each subset $S_i$, where $2 \leq i \leq m$, there exists a string $x\beta_i \in S_i$ such that $x\beta_i \in S_j$ for some $j < i$.
3. Use the strings $\beta_2, \beta_3, \ldots, \beta_m$ to adapt $f$ to create a function $g$ that constructs a universal cycle for $S$.

We apply these three steps to derive a de Bruijn successor in the following example.

Example 2

1. The function $f = PCR$ is a UC-successor for $B(n)$. It partitions $B(n)$ into its necklace equivalence classes $\{S_1, S_2, \ldots, S_m\}$.
2. Order the subsets $S_1, S_2, \ldots, S_m$ in reverse lexicographic order based on their necklace representatives. Thus $S_1 = \{1^n\}$ and $S_m = \{0^n\}$. For $i > 1$, if the necklace representative of $S_i$ is $b_1b_2\cdots b_n$, then let $\beta_i = b_2b_3\cdots b_n$. Since $i > 1$, $b_1 = 0$. Therefore, the necklace representative of $1b_2b_3\cdots b_n$ will be larger than $0b_2\cdots b_n$, and thus it will appear in a subset $S_j$ where $j < i$. This implies that the sequence $\beta_2, \beta_3, \ldots, \beta_m$ is a spanning sequence of $S_1, S_2, \ldots, S_m$. A visualization of this spanning sequence for $n = 6$ is provided in Figure 1.
3. Using the definition of PCR and \( \beta_2, \beta_3, \ldots, \beta_m \), we derive the following de Bruijn successor for each
\( \alpha = a_1a_2\cdots a_n \).

\[
g(\alpha) = \begin{cases} 
\pi_1 & \text{if } 0a_2a_3\cdots a_n \text{ is a necklace;} \\
a_1 & \text{otherwise.}
\end{cases}
\]

This de Bruijn successor is equivalent to the PCR4 successor stated in the next section.

4 **De Bruijn Successors**

The de Bruijn successors defined in this section are based on the PCR and the CCR. First, we present four
de Bruijn successors based on the PCR. Then we present three de Bruijn successors based on the CCR. The
proof of correctness for each de Bruijn successor applies Theorem 3.5.

4.1 **The Pure Cycling Register and Necklaces**

In this subsection, we present four de Bruijn successors based on the PCR and necklaces.

**PCR1 (Granddaddy) successor** \( g_1 \)

Let \( j \) be the smallest index of \( \alpha = a_1a_2\cdots a_n \) such that \( a_j = 0 \) and \( j > 1 \), or \( j = n+1 \) if no such index
exists. Let \( \gamma = a_ja_{j+1}\cdots a_n0a_2\cdots a_{j-1} = a_ja_{j+1}\cdots a_n01^{j-2} \).
\[ g_1(\alpha) = \begin{cases} \overline{\alpha}_1 & \text{if } \gamma \text{ is a necklace;} \\ a_1 & \text{otherwise.} \end{cases} \]

It has already been shown that \( g_1 \) is a de Bruijn successor in [10]. We provide an alternate proof using our framework.

**Theorem 4.1** The function \( g_1 \) is a de Bruijn successor.

**Proof.** Consider the UC-partition \( S_1, S_2, \ldots, S_m \) of \( B(n) \) with respect to the PCR. Suppose that the subsets are ordered in reverse lexicographic order with respect to their necklace representatives. Thus, \( S_1 = \{1^n\} \) and the necklace representative \( \gamma_i \) for \( S_i \) must contain a 0 for \( i \in \{2, 3, \ldots, m\} \). Let \( \gamma_i = a_ja_{j+1} \cdots a_n01^{j-2} \) and let \( \beta_i = a_2a_3 \cdots a_n = 1^{j-2}a_ja_{j+1} \cdots a_n \). Since \( \beta_i \) is a rotation of \( \gamma_i \), it is in \( S_i \). Also, observe that the necklace representative of Neck(\( \beta_i \)) is clearly larger than \( \gamma_i \). Thus \( \beta_2, \beta_3, \ldots, \beta_m \) is a spanning sequence for the partition. Furthermore, observe that \( g_1(a_1a_2\cdots a_n) = \overline{\alpha}_1 \) for any \( a_1 \) since \( \gamma_i \) is a necklace. Thus, by Theorem 3.5, \( g_1 \) is a UC-successor of \( B(n) \).

**PCR2 (Grandmama) successor \( g_2 \)**

Let \( j \) be the largest index of \( \alpha = a_1a_2\cdots a_n \) such that \( a_j = 1 \), or \( j = 0 \) if no such index exists. Let \( \gamma = a_{j+1}a_{j+2}\cdots a_n1a_2\cdots a_j = 0^{n-j}1a_2\cdots a_j \).

\[ g_2(\alpha) = \begin{cases} \overline{\alpha}_1 & \text{if } \gamma \text{ is a necklace;} \\ a_1 & \text{otherwise.} \end{cases} \]

It has already been shown that \( g_2 \) is a de Bruijn successor in [4]. We provide an alternate proof using our framework.

**Theorem 4.2** The function \( g_2 \) is a de Bruijn successor.

**Proof.** Consider the UC-partition \( S_1, S_2, \ldots, S_m \) of \( B(n) \) with respect to the PCR. Suppose that the subsets are ordered in lexicographic order with respect to their necklace representatives. Thus, \( S_1 = \{0^n\} \) and the necklace representative \( \gamma_i \) for \( S_i \) must contain a 1 for \( i \in \{2, 3, \ldots, m\} \). Let \( \gamma_i = 0^{n-j}1a_2\cdots a_j \) and let \( \beta_i = a_2a_3\cdots a_n = a_2\cdots a_j0^{n-j} \). Since \( 1\beta_i \) is a rotation of \( \gamma_i \), it is in \( S_i \). Also, observe that the necklace representative of Neck(\( \beta_i \)) is clearly smaller than \( \gamma_i \). Thus \( \beta_2, \beta_3, \ldots, \beta_m \) is a spanning sequence for the partition. Furthermore, observe that \( g_2(a_1a_2\cdots a_n) = \overline{\alpha}_1 \) for any \( a_1 \) since \( \gamma_i \) is a necklace. Thus, by Theorem 3.5, \( g_2 \) is a UC-successor of \( B(n) \).

**PCR3 (J1) successor \( g_3 \)**

Let \( \alpha = a_1a_2\cdots a_n \) and let \( \gamma = a_2a_3\cdots a_n1 \).

\[ g_3(\alpha) = \begin{cases} \overline{\alpha}_1 & \text{if } \gamma \text{ is a necklace;} \\ a_1 & \text{otherwise.} \end{cases} \]
It has already been shown that \( g_3 \) is a de Bruijn successor in both [19] and [26]. We provide an alternate, simpler proof using our framework.

**Theorem 4.3** The function \( g_3 \) is a de Bruijn successor.

**Proof.** Consider the UC-partition \( S_1, S_2, \ldots, S_m \) of \( B(n) \) with respect to the PCR. Suppose the subsets are ordered in lexicographic order with respect to their necklace representatives. Thus, \( S_1 = \{0^n\} \) and the necklace representative \( \gamma_i \) for \( S_i \) must end with 1 for \( i \in \{2, 3, \ldots, m\} \). Let \( \gamma_i = a_2 a_3 \cdots a_n 1 \) and let \( \beta_i = a_2 a_3 \cdots a_n \). Since \( 1 \beta_i \) is a rotation of \( \gamma_i \), it is in \( S_i \). Also, observe that the necklace representative of \( \text{Neck}(0 \beta_i) \) is clearly smaller than \( \gamma_i \). Thus \( \beta_2, \beta_3, \ldots, \beta_m \) is a spanning sequence for the partition. Furthermore, observe that \( g_3(a_1 a_2 \cdots a_n) = \overline{a_1} \) for any \( a_1 \) since \( \gamma_i \) is a necklace. Thus, by Theorem 3.5, \( g_3 \) is a UC-successor of \( B(n) \). \( \square \)

**PCR4 successor \( g_4 \)**

Let \( \alpha = a_1 a_2 \cdots a_n \) and let \( \gamma = 0a_2 a_3 \cdots a_n \).

\[
g_4(\alpha) = \begin{cases} 
\overline{a_1} & \text{if } \gamma \text{ is a necklace;} \\
\alpha & \text{otherwise.}
\end{cases}
\]

A proof of the following theorem was provided in Example 2.

**Theorem 4.4** The function \( g_4 \) is a de Bruijn successor.

### 4.2 The Complemented Cycling Register and Co-necklaces

In this subsection, we present three de Bruijn successors based on the CCR and co-necklaces.

**CCR1 successor \( g_5 \)**

Let \( \alpha = a_1 a_2 \cdots a_n \). Let \( j \) be the smallest index of \( a_2 a_3 \cdots a_n \) such that \( a_j = 0 \), or \( j = n+1 \) if no such index exists. Let \( \gamma = a_j a_{j+1} \cdots a_n \overline{a_{j+1}} \cdots \overline{a_{j-1}} = a_j a_{j+1} \cdots a_n 0^{j-2} \).

\[
g_5(\alpha) = \begin{cases} 
a_1 & \text{if } \gamma \text{ is a co-necklace;} \\
\overline{a_1} & \text{otherwise.}
\end{cases}
\]

**Theorem 4.5** The function \( g_5 \) is a de Bruijn successor.

**Proof.** Consider the UC-partition \( S_1, S_2, \ldots, S_m \) of \( B(n) \) with respect to the CCR. Suppose the subsets are ordered in lexicographic order with respect to their co-necklace representatives. Thus, the co-necklace representative of \( S_1 \) is \( 0^n \) and the co-necklace representative \( \gamma_i \) for \( S_i \) must contain a 1 for \( i \in \{2, 3, \ldots, m\} \).

Let \( \gamma_i = a_j a_{j+1} \cdots a_n 0^{j-2} \) and let \( \beta_i = a_2 a_3 \cdots a_n = 1^{j-2} a_j a_{j+1} \cdots a_n \). Since \( 0 \beta_i \) is in \( \text{coNeck}(\gamma_i) \), it is in \( S_i \). Observe that \( a_j a_{j+1} \cdots a_n 0^{j-1} \) is in \( \text{coNeck}(1 \beta_i) \).

Since \( a_j a_{j+1} \cdots a_n 0^{j-1} \) is less than \( \gamma_i \) it must be that \( 1 \beta_i \) appears in some \( S_j \) where \( j < i \). Thus \( \beta_2, \beta_3, \ldots, \beta_m \) is a spanning sequence for the partition. Furthermore, observe that \( g_5(a_1 a_2 \cdots a_n) = \overline{a_1} \) for any \( a_1 \) since \( \gamma_i \) is a co-necklace. Thus, by Theorem 3.5, \( g_5 \) is a UC-successor of \( B(n) \). \( \square \)
Theorem 4.6 The function \( g_6 \) is a de Bruijn successor.

Proof. Consider the UC-partition \( S_1, S_2, \ldots, S_m \) of \( B(n) \) with respect to the CCR. Suppose the subsets are ordered in lexicographic order with respect to their co-necklace representatives. Thus, the co-necklace representative of \( S_1 \) is \( 0^n \) and the co-necklace representative \( \gamma_i \) for \( S_i \) must contain a \( 1 \) for \( i \in \{2, 3, \ldots, m\} \). Let \( \gamma_i = 0^{n-j}1\pi_2\pi_3\cdots\pi_j \) and let \( \beta_i = a_2a_3\cdots a_n = a_2a_3\cdots a_{j}0^{n-j} \). Since \( 1\beta_i \) is in \( \text{coNeck}(\gamma_i) \), it is in \( S_i \). Observe that \( 0^{n-j+1}\pi_2\pi_3\cdots\pi_j \), which is in \( \text{coNeck}(0\beta_i) \), is clearly less than \( \gamma_i \). Thus, \( 1\beta_i \) appears in some \( S_j \) where \( j < i \). Thus \( \beta_2, \beta_3, \ldots, \beta_m \) is a spanning sequence for the partition. Furthermore, observe that \( g_6(a_1a_2\cdots a_n) = \overline{a}_1 \) for any \( a_1 \) since \( \gamma_i \) is a co-necklace. Thus, by Theorem 3.5, \( g_6 \) is a UC-successor of \( B(n) \). \( \square \)

Although stated differently, it has been previously shown that \( g_7 \) is a de Bruijn successor in [19]. We provide an alternate proof using our framework.

Theorem 4.7 The function \( g_7 \) is a de Bruijn successor.

Proof. Consider the UC-partition \( S_1, S_2, \ldots, S_m \) of \( B(n) \) with respect to the CCR. Suppose the subsets are ordered in lexicographic order with respect to their co-necklace representatives. Thus, the co-necklace representative of \( S_1 \) is \( 0^n \). Note also that every co-necklace representative \( \gamma_i \) for \( S_i \) must end with 0 for all \( i \in \{2, 3, \ldots, m\} \). Let \( \gamma_i = a_2a_3\cdots a_{n}0 \neq 0^n \) and let \( \beta_i = a_2a_3\cdots a_{n} \). Since \( 1\beta_i \) is in \( \text{coNeck}(\gamma_i) \), it is in \( S_i \). Also, since \( \gamma_i \neq 0^n \), we have \( 0\beta_i \) is smaller than \( \gamma_i \). Thus \( 0\beta_i \) appears in some \( S_j \) where \( j < i \). Thus \( \beta_2, \beta_3, \ldots, \beta_m \) is a spanning sequence for the partition. Furthermore, observe that \( g_7(a_1a_2\cdots a_n) = \overline{a}_1 \) for any \( a_1 \) since \( \gamma_i \neq 0^n \) is a co-necklace. Thus, by Theorem 3.5, \( g_7 \) is a UC-successor of \( B(n) \). \( \square \)

Note the similarities between \( g_1 \) and \( g_5 \), \( g_2 \) and \( g_6 \), and \( g_3 \) and \( g_7 \). It is interesting that a function similar to \( g_7 \) did not yield a de Bruijn successor based on the CCR. The reason is as follows. For necklaces, the first bit of a necklace \( \alpha \neq 1^n \) is 0. When you flip this bit, the resulting string belongs to a necklace class whose necklace representative is always lexicographically larger than \( \alpha \). The same is not true for co-necklaces. For example, flipping the first bit of the co-necklace 01010 yields the string 11010 and the representative of \( \text{coNeck}(11010) \) is 00010 which is smaller than 01010. On the other hand, flipping the first bit of the co-necklace 00010 yields the string 10010 and the representative of \( \text{coNeck}(10010) \) is 00100 which is larger than 00010.
5 Efficient Implementation

Given a UC-successor \( f \) for \( S \), the following algorithm \textsc{ConstructUC}\((f, \sigma)\) will construct a UC for \( S \) starting with some arbitrary string \( \sigma \in S \).

\begin{algorithm}
\caption{Constructing a UC for \( S \) starting with \( \sigma \in S \) given a UC-successor \( f \).}
\begin{algorithmic}[1]
\Procedure{ConstructUC}{\( f, \sigma \)}
\State \( \alpha \leftarrow \sigma \)
\Repeat
\State \( \triangleright \alpha \) re-indexed to \( a_1 a_2 \cdots a_n \)
\State \( \text{Print}(a_1) \)
\State \( \alpha \leftarrow a_2 \cdots a_n f(\alpha) \)
\Until \( \alpha = \sigma \)
\EndProcedure
\end{algorithmic}
\end{algorithm}

Each de Bruijn successor presented in this paper test whether or not a string is a necklace or a co-necklace. Testing whether or not a string is a necklace can be done in \( O(n) \) time [3]. This immediately implies that we can also test if a string is a co-necklace in \( O(n) \)-time by the definition of a co-necklace. Such a necklace testing function \( \text{ISNECKLACE} \) is shown in Algorithm 2.

\begin{algorithm}
\caption{A membership function that tests whether or not a string is a necklace.}
\begin{algorithmic}[1]
\Function{ISNECKLACE}{\( a_1 a_2 \cdots a_n \)}
\State \( p \leftarrow 1 \)
\For {\( j \) from 2 to \( n \)}
\If {\( a_j < a_{j-p} \)} \Return{FALSE} \EndIf
\If {\( a_j > a_{j-p} \)} \State \( p \leftarrow j \) \EndIf
\EndFor
\If {\( n \mod p \neq 0 \)} \Return{FALSE} \EndIf
\Ensure{TRUE}
\EndFunction
\end{algorithmic}
\end{algorithm}

\textbf{Theorem 5.1} The de Bruijn successors \( g_1, g_2, g_3, g_4, g_5, g_6 \) and \( g_7 \) can be used to construct de Bruijn sequences in \( O(n) \)-time per bit using \( O(n) \)-space.

A complete C implementation for these successors is given in the Appendix.

6 Summary and Future Work

In this paper, we presented a framework to construct universal cycles for binary languages via successor rules. Unlike greedy and necklace concatenation approaches, a successor rule based construction can start from any arbitrary string of length \( n \). This framework was used to identify and prove three new and simple de Bruijn sequence constructions based on the PCR and the CCR. For each new de Bruijn sequence construction \( D = d_1 d_2 \cdots d_{2^n} \), the following open problems are of interest to many researchers:

1. (Ranking) Given a length \( n \) string \( \alpha \), efficiently determine the index \( j \) such that the length \( n \) string in \( D \) beginning from index \( j \) is \( \alpha \).

2. (Unranking) Given an index \( j \), efficiently determine the length \( n \) string in \( D \) that begins at index \( j \).

In future work, we generalize our results to other feedback shift registers including the pure summing register and the complemented summing register. We also generalize this framework to languages over an alphabet of arbitrary size, although the formulation is slightly more complex [15].
7 Acknowledgement

The research of Joe Sawada is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) grant RGPIN 400673-2012. The research of Dennis Wong is supported by the MSIT (Ministry of Science and ICT), Korea, under the ICT Consilience Creative Program (IITP-2017-R0346-16-1007) supervised by the IITP (Institute for Information & Communications Technology Promotion).

References


Appendix - C code

```c
#include <stdio.h>
#define MAX 100

// Test if a[1..n] = 0^n
int Zeros(int a[], int n) {
    for (int i=1; i<=n; i++)
        if (a[i] == 1) return 0;
    return 1;
}

// Test if b[1..n] is a necklace
int IsNecklace(int b[], int n) {
    int i, p=1;
    for (i=2; i<=n; i++)
        if (b[i-p] > b[i]) return 0;
        if (b[i-p] < b[i]) p = i;
    if (n % p != 0) return 0;
    return 1;
}

// Necklace Successor Rules
int Granddaddy(int a[], int n) {
    int i, j, b[MAX];
    j = 2;
    while (j<n && a[j] == 1) j++;
    for (i=j; i<=n; i++) b[i-j+1] = a[i];
    b[n-j+2] = 0;
    for (i=2; i<j; i++) b[n-j+i+1] = a[i];
    if (IsNecklace(b,n)) return 1-a[1];
    return a[1];
}

int Grandmama(int a[], int n) {
    int i, j, k, b[MAX];
    j = 1;
    while (j<n && a[n-j+1] == 0) b[j++] = 0;
    b[j] = 1;
    k = 2;
    for (i=j+1; i<=n; i++) b[i] = a[k++];
    if (IsNecklace(b,n)) return 1-a[1];
    return a[1];
}

int PCR3(int a[], int n) {
    int i, b[MAX];
    for (i=1; i<n; i++) b[i] = a[i+1];
    b[n] = 1;
    if (IsNecklace(b,n)) return 1-a[1];
    return a[1];
}

int PCR4(int a[], int n) {
    int i, b[MAX];
    for (i=1; i<n; i++) b[i] = a[i+1];
    b[n] = 1;
    if (IsNecklace(b,n)) return 1-a[1];
    return a[1];
}
```
b[1] = 0;
for (i=2; i<=n; i++) b[i] = a[i];
if (IsNecklace(b,n)) return 1-a[1];
return a[1];
}
// Co-necklace Successor Rules
// Co-necklace Successor Rules
int CCR1(int a[], int n) {
    int i,j,b[MAX],c=1;
    for (i=2; i<=n; i++)
        if (a[i] == 0) break;
    for (j=i; j<=n; j++) b[c++] = a[j];
    b[c++] = 1;
    for (j=2; j<i; j++) b[c++] = 1-a[j];
    for (i=1; i<=n; i++) b[n+i] = 1-b[i];
    if (IsNecklace(b,2*n)) return a[1];
    return 1-a[1];
}
// -------------------------------
int CCR2(int a[], int n) {
    int i,j,b[MAX],c=1;
    i = n;
    while(a[i] == 0 && i > 1) i--;
    if (i == 0) i = n;
    for (j=i+1; j<=n; j++) b[c++] = 0;
    b[c++] = 1;
    for (j=2; j<=i; j++) b[c++] = 1-a[j];
    for (j=1; j<=n; j++) b[n+j] = 1-b[j];
    if (IsNecklace(b,2*n)) return a[1];
    return 1-a[1];
}
// -------------------------------
int CCR3(int a[], int n) {
    int i,b[MAX];
    for (i=1; i<n; i++) b[i] = a[i+1];
    b[n] = 0;
    for (i=1; i<n; i++) b[n+i] = 1-b[i];
    if (IsNecklace(b,2*n) & & !Zeros(b,n)) return a[1];
    return 1-a[1];
}
// Generate de Bruijn sequences by iteratively applying a successor rule
// Generate de Bruijn sequences by iteratively applying a successor rule
void DB(int seq, int n) {
    int i, new_bit, a[MAX];
    for (i=1; i<=n; i++) a[i] = 0; // First n bits
    do {
        printf("%d", a[1]);
        switch (seq) {
            case 1: new_bit = Granddaddy(a,n); break;
            case 2: new_bit = Grandmama(a,n); break;
            case 3: new_bit = PCR3(a,n); break;
            case 4: new_bit = PCR4(a,n); break;
            case 5: new_bit = CCR1(a,n); break;
            case 6: new_bit = CCR2(a,n); break;
            case 7: new_bit = CCR3(a,n); break;
            default: break;
        }
        for (i=1; i<=n; i++) a[i] = a[i+1];
        a[n] = new_bit;
    }
while (!Zeros(a, n));

int main() {
    int i, n;

    printf("Enter n: "); scanf("%d", &n);
    for (i=1; i<=7; i++) {
        switch (i) {
            case 1: printf("Granddaddy (lex minimal):
"); break;
            case 2: printf("Grandmama:
"); break;
            case 3: printf("PCR3:
"); break;
            case 4: printf("PCR4:
"); break;
            case 5: printf("CCR1:
"); break;
            case 6: printf("CCR2:
"); break;
            case 7: printf("CCR3:
"); break;
            default: break;
        }
        DB(i, n);
    }
    printf("\n\n");
}