Generating de Bruijn Sequences with Many Factor-Rich Prefixes

D. Repke, W. Rytter

Abstract

A binary word is *factor-rich* iff it has the largest number of distinct factors among binary words with the same length. Each linear binary de Bruijn word of rank n has length $\Delta_n = 2^n + n - 1$ and is (as the whole word) factor-rich. A binary de Bruijn word of rank n is called here *abundant* iff each of its prefixes of size m is factor-rich for $\Delta_{n-1} < m \leq \Delta_n$. In this paper we show an $O(N)$ time algorithm constructing binary abundant de Bruijn words of each rank n, where N is the length of the produced word. We show $O(\log N)$ time algorithm computing compact description of abundundant de Bruijn words and factor-rich words of length N. The description is a straight-line program using operations of concatenating, taking prefixes and special simple arithmetic transformation (denoted by Ψ, which corresponds to Lempel homomorphism. Our recursive construction of de Bruijn words can be viewed as a syntactic version of Lempel graph-theoretic algorithm for de Bruijn words.

1 Introduction

In this paper we consider binary strings which are *factor-rich*: they contain the largest number of distinct factors among binary strings of the same length. A construction of a factor-rich word of a given length was given in [**?**] using graph theoretic properties of de Bruijn graphs. We show here an alternative syntactic algorithm producing a growing family $\mathbf{U}(n)$, $n \geq 1$, of de Bruijn words having a lot of factor-rich words as their prefixes: each prefix of size $m > |\mathbf{U}(n-1)|$ of $\mathbf{U}(n)$ is factor-rich. In case of ternary alphabet it is even possible to construct de Bruijn words with every factor-rich prefix, however as shown in [**?**] it is not possible for binary alphabet. Surprisingly the case of binary alphabet is much harder than that of larger alphabets.

By an n-string we mean a binary word of length n. Denote by $\mathrm{BIN}(n)$ the set of all binary strings of length n.

For a linear word w denote by $\mathcal{F}_n(w)$ the set of all n-strings that are substrings of w. Similarly, denote by $\mathcal{C}_n(w)$ the set of all n-strings that are cyclic substrings of w.
Example 1. $\mathcal{F}_2(abc) = \{ab, bc\}$, $\mathcal{C}_2(abc) = \{ab, bc, ca\}$.

A de Bruijn word of rank n is a binary word w containing in the cyclic sense each binary word of length n exactly once, in other words $|\mathcal{C}_n(w)| = |w| = 2^n$. Denote by $\mathbf{DB}_n$ the set of binary cyclic de Bruijn words of rank n.

A linear de Bruijn word of rank n is a shortest binary word containg each n-string exactly once as a linear (standard) string, in other words $|\mathcal{F}_n(w)| = 2^n$. The length of a binary linear de Bruijn word of rank n is Δ_n, where $\Delta_k = 2^k + k - 1$.
Example 2. Binary de Bruijn word of rank 4 has length 2^4 and linear binary de Bruijn word of rank 4 has length $\Delta_4 = 2^4 + 3 = 19$.

We say that a linear de Bruijn word of length Δ_n is *abundant* iff each of its prefixes of size m is factor-rich for $\Delta_{n-1} < m \leq \Delta_n$.

Example 3. The linear de Bruijn word 0111000101 of rank 3 is abundant, because its prefix 011100 of size $\Delta_2 + 1$ is factor-rich. However the word 0001011100 is not abundant.

Zdefiniowa straingt-line program

Our results. We show that there is a family of *abundant* linear binary de Bruijn words $\mathbf{U}(n)$, and these words can be computed by a simple recursive linear-time algorithm. Moreover the words $\mathbf{U}(n)$ have compact representation of size $O(n)$, which is logarithmic with respect to $|\mathbf{U}(n)|$.

2 Preliminaries

If $|u| \geq k$ denote by $Pref_k(u), Suf_k(u)$ the prefix, respectively suffix of u of size k. For $y \in \mathbf{DB}_n$ denote:

$$\mathbf{lin}(y) = y \cdot Pref_{n-1}(y).$$

Observation 1.
(a) x s a linear binary de Bruijn word of rank n iff $x = \mathbf{lin}(y)$, where $y \in \mathbf{DB}_n$.
(b) If $y \in \mathbf{DB}_n$ then $\mathcal{C}_n(y) = \mathcal{F}_n(\mathbf{lin}(y))$.

Lemma 1. [**?**]
A binary word w of size $\Delta_{n-1} < m \leq \Delta_n$ is factor-rich iff it contains, as a factor, each binary word of size $n-1$ and all its factors of size n are distinct, in other words:

$$|\mathcal{F}_{n-1}(w)| = 2^{n-1} \wedge |\mathcal{F}_n(w)| = m - n + 1.$$

An immediate consequence is the following fact.
Corollary 1. *A de Bruijn word w of rank n is abundant iff its prefix of size $\Delta_{n-1} + 1$ contains, as a factor, each binary word of size $n-1$.*

We introduce two rather strange operations on words. Both operations work on binary words of size 2^n.

2.1 Abstract operation $\oplus$.

Denote by $SHIFT(x, \gamma)$ cyclic shift of the word x having the suffix equals γ, assuming γ occurs exactly once as a cyclic factor of x.

Example. $SHIFT(001011101,\, 011) \;=\; 101001011$.

For two words $x, y \in \mathrm{BIN}(2^n)$ define

$$x \oplus y \;=\; x \cdot SHIFT(y, Suf_n(x)).$$

Observation 2. If u, v have a common prefix of length $0 \leq m \leq n$ and common suffix of length $n - m$ then

$$\mathcal{C}_{n+1}(u \cdot v) \;=\; \mathcal{C}_{n+1}(u) \cup \mathcal{C}_{n+1}(v).$$

Lemma 2. *Assume* $x, y \in \mathrm{BIN}(2^n)$, $\mathcal{C}_{n+1}(x) \cup \mathcal{C}_{n+1}(y) \;=\; \mathrm{BIN}(n+1)$ *and* $Suf_n(x) \in \mathcal{C}_n(y)$. *Then* $x \oplus y \in \mathbf{DB}_{n+1}$

Proof. It follows directly from the Observation 2. □

2.2 Abstract operation Ψ

The crucial operation is a special algebraic operation $\Psi(w)$ on words, the result of this operation is the binary word v of length $|w|$, where for every $1 \leq i \leq |w|$:

$$v[i] \;=\; (\, w[1] + w[2] + \ldots w[i] \,) \bmod 2.$$

The operation Ψ was used in the algorithm R, presented in Knuth's 4-th volume, see [**?**]. It also implicitly appears in Lempel's algorithm for de Bruijn words using homomorphism of de Bruijn graphs, see [**?**]. In fact the recursive algorithm presented in this section can be viewed as syntactic version of Lempel's graph-theoretic algorithm.

Example 4. If $w = 0010111011$ then $\Psi(w) \;=\; 0011010010$.

Denote by $\bar{x}$ the bitwise negation of the word x. We also use (equivalent) notation $\neg x$.

It is possible that $\neg\Psi(x) \neq \Psi(\neg x)$. The basic properties of this operation are:

Observation 3. Assume $x \in \mathbf{DB}_n$ and $y = \Psi(x)$. Then:
(1) All cyclic $(n+1)$-factors of y are pairwise distinct.
(2) if z is a cyclic $(n+1)$-factor of y then $\bar{z}$ is not a cyclic $(n+1)$-factor of y.
(3) $\mathcal{C}_{n+1}(y) \cap \mathcal{C}_{n+1}(\bar{y}) \;=\; \emptyset$.

Introduce also an operation τ which, in a certain sense, is a reverse of Ψ. For an $(n+1)$-string w define $\tau(w)$ as an n-string v such that $\Psi(w[1]\, v) = w$.

Observation 4.
(a) $\tau(w) = \tau(\bar{w})$; **(b)** $\tau(w) = \tau(w') \Leftrightarrow w = w'$ or $w' = \bar{w}$.
(c) If $\Psi(x)$ ends with 0 then $\mathcal{C}_n(x) = \{\tau(w) : w \in \mathcal{C}_{n+1}(\Psi(x))\}$.

3 Recursive construction of de Bruijn words

For $x \in \mathbf{DB}_n$ ending with 1^n, where $n \geq 2$, denote:

$$NEXT(x) = \Psi(x) \oplus \overline{\Psi(x)}.$$

If x does not end with 1^n then assume that it is shifted to satisfy this condition as initial part of the operation $NEXT$.

Example 5. $NEXT(00010111) = 00011010 \cdot 11110010$. Observe that before the group of 1^{n+1} we have the factor α_4. However $\alpha_4 \cdot 1 = \beta_5$, hence the resulting text $00011010 \cdot 11110010$ contains β_5.

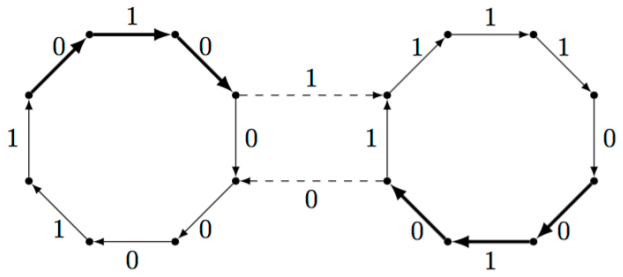

Figure 1: Graphical interpretation of computing $NEXT(00010111)$. Two cycles correspond to $\Psi(00010111) = 00011010$, $\neg\Psi(x)) = 11100101$. $SHIFT(11100101, 010) = 11110010$. The operation $\oplus$ applied to these two cyclic words corresponds to interchanging successors of two nodes (dotted arrows). The result is the cyclic word $00011010\,11110010 \in \mathbf{DB}_4$. The edges corresponding to the synchronizing word are shown in bold.

Lemma 3. [Recursive construction of de Bruijn words]
If $x \in \mathbf{DB}_n$, $n \geq 2$ *then* $NEXT(x) \in \mathbf{DB}_{n+1}$.

Proof. It is enough to show that words $\Psi(x)$ and its negation satisfy the assumptions of Lemma 2.

Claim 1. *If* $x \in \mathbf{DB}_n$ *then* $\mathcal{C}_{n+1}(\Psi(x)) \cup \mathcal{C}_{n+1}(\overline{\Psi(x)}) = \mathrm{BIN}(n+1)$.

The word x is de Bruijn word of rank n, so we know that $|\mathcal{C}_n(x)| = 2^n$. Due to Observation 3 (c) the size of set $\mathcal{C}_{n+1}(\Psi(x))$ is also 2^n. Moreover, if $\mathcal{C}_{n+1}(\Psi(x))$ contains word w then it does not contain word $\bar{w}$, because $\tau(w) = \tau(\bar{w})$, so it would decrease the size of $\mathcal{C}_n(x)$. It means that the sets of cyclic factors of word $\Psi(x)$ and of its negation are disjoint. Therefore, union of these sets is of size 2^{n+1}. □

4 Complementary de Bruijn words

The key component of the algorithm is the construction of pairs of de Bruijn words of rank n which have the smallest number of common $(n+1)$-factors. Two words in $\mathbf{DB}_n$ are said to be *complementary* if the set of their common cyclic factors of length $n+1$ is:

$$Common(n) = \{\, 0^n1,\, 1^n0,\, 01^n,\, 10^n \,\}.$$

Observe that each word in $\mathbf{DB}_n$ contains, as its factors, all words in $Common(n)$.

Example 6. De Bruijn words 00011101 and 00010111 are complementary.

Observe first that every two words in $\mathbf{DB}_n$ with $n \geq 2$ have at least these four common $(n+1)$-factors, since every de Bruijn word contains words 01^n0 and 10^n1.

Moreover, two complementary words in $\mathbf{DB}_n$ contain all cyclic factors of length $n+1$ except:

$$Unused(n) = \{\, 0^{n+1},\, 1^{n+1},\, 01^{n-1}0,\, 10^{n-1}1 \,\}.$$

Similarly, none of words in $\mathbf{DB}_n$ contains any of these four unused $(n+1)$-factors.

Denote by α_n the word of length n which does not contain 00 or 11 and ends with 0. Let β_n denote biwise negation of α_n.

Example 7. $\alpha_3 = 010,\ \beta_3 = 101,\ \alpha_4 = 1010,\ \beta_4 = 0101$..

The following fact follows directely from definitions.

Observation 5. Assume $z \in \mathbf{DB}_n$ ends with 1^n, then $NEXT(z)$ contains β_{n+2} as a factor.

For a set Z of binary words denote $\bar{Z} = \{\bar{u} \,:\, u \in Z\}$.

Denote by D_k the set of 4 words in BIN(k) containing α_{k-1} or β_{k-1} having the last or the first two letters equal and by R_k the set of 4 words ...??.

Example 8. Let $x = 00010111$ and $n = 3$, then

$$\mathcal{C}_{n+2}(\Psi(x)) \cup \mathcal{C}_{n+2}(\overline{\Psi(x)}) \;=\; A_x \,\dot\cup\, \overline{A_x} \,\dot\cup\, D_{n+2},\ \text{where}$$

$$D_{n+2} \;=\; \{\, 10100,\, 11010,\, 01011,\, 00101\},\ A_x \;=\; \{\, 00001,\, 00011,\, 0010,\, 01101,\, 01000,\, 10000\}.$$

Observe that $D_{n+2} = \overline{D_{n+2}}$. However

$$\mathcal{C}_{n+2}(NEXT(x)) \;=\; \mathcal{C}_{n+2}(\Psi(x) \oplus \overline{\Psi(x)}) \;=\; A_x \,\dot\cup\, \overline{A_x} \,\dot\cup\, R_{n+2},$$

where $R_{n+2} = \{10101,\, 11010,\, 00100,\, 01011\}$. Observe that now $R_{n+2} \cap \overline{R_{n+2}} = \emptyset$ (these sets are *complementary*, while D_{n+2}, $\overline{D_{n+2}}$ are not). The main effect of $NEXT$ is replacement of D_{n+2} by R_{n+2}, see also Figure 1.

The following lemma gives directely a recursive linear time construction of complementary words starting with two complementary words of rank $n = 3$.

Lemma 4. *Assume that the words $x, y \in \mathbf{DB}_n$ are complementary with rank $n \geq 2$, and each of them ends with 1^n. Then $NEXT(x)$ and $\neg(NEXT(y))$ are complementary and are in $\mathbf{DB}_{n+1}$.*

Proof. Assume (without loss of generality) that $n = 2k + 1$, where k is odd. We investigate the structure of the set of $(n+2)$-cyclic-factors of $NEXT(x)$, $NEXT(y)$. The sets of $(n+1)$-cyclic-factors of x, y have exactly 4 elements in common, consequently, since x, y are complementary, the sets of $(n+1)$-cyclic-factors of $NEXT(x)$, $NEXT(y)$ have at most 8 elements in common. We show explicitely the set $R_{n+2} = R_y$ of 4 common $(n+2)$-cyclic-factors.

For each $x, y \in \mathbf{DB}_n$ which end with 1^n both de Bruijn words $NEXT(x)$, $NEXT(y)$) are of the following form:

$$0.....1\,(01)^k\,0\;\; 1.....0\,(01)^k\,0.$$

Figure 2: Structure of $NEXT(z)$, where $z \in \mathbf{DB}(n)$ ends with 1^n and $n = 2k+1$, for odd k.

We have:

$$\mathcal{C}_{n+2}(NEXT(x)) \;=\; A_x \,\dot\cup\, \overline{A_x} \,\dot\cup\, R_{n+2},$$
$$\mathcal{C}_{n+2}(NEXT(y)) \;=\; A_y \,\dot\cup\, \overline{A_y} \,\dot\cup\, R_y,$$

Due to the structure of the operation $NEXT$ we have

$$R_{n+2} \;=\; R_y \;=\; \{\, 1\,(01)^k 01,\; 0\,(01)^k 00,\; 0\,(10)^k 11,\; 1\,(10)^k 10 \,\}.$$

The last two words are in R_{n+2} as negations of cyclic factors $1\,(01)^k 00,\; 0\,(01)^k 01 \;\in\; \Psi(x) \cup \overline{\Psi(x)}$ (this set is closed under biwise negation).

When we take negation of $NEXT(y)$ then R_y will become $\overline{R_{n+2}}$, consequently 4 common $(n+2)$-factors of $NEXT(x)$, $NEXT(y)$ are no longer common, since $R_{n+2} \cap \overline{R_y} = \emptyset$, see also Example 8. Hence $NEXT(x)$, $\overline{NEXT(y)}$ are complementary since they have now only 4 $(n+2)$-factors in common. This completes the proof. □

Example 9. Let us look at the operation NEXT from the graph-theoretic perspective. Let $x = \texttt{00010111}$ and $y = \texttt{01000111}$. These two words are complementary de Bruijn words of rank 3.

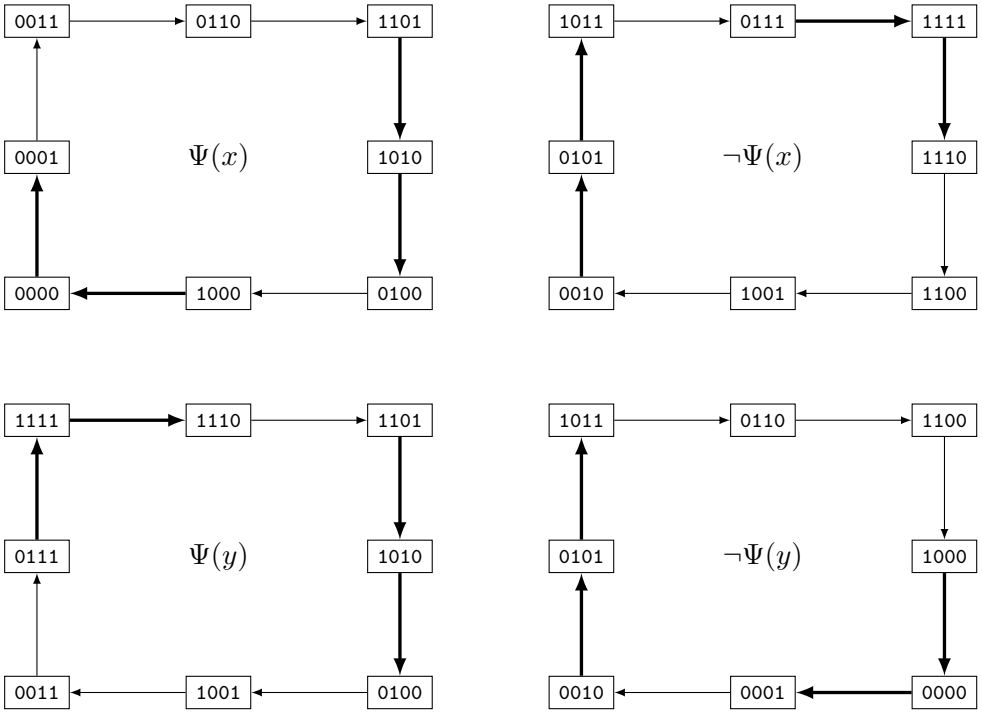

Figure 3: Illustration of words $\Psi(x), \neg\Psi(x), \Psi(y)$ and $\neg\Psi(y)$.

The bold edges are formed from the $Common(n)$ edges i.e. `0001`, `0111`, `1000` and `1110`. These are the only common edges between the $\Psi(x) \cup \neg\Psi(x)$ and $\Psi(y) \cup \neg\Psi(y)$. There are always exactly 8.

We know that the word $\Psi(x)$ ends with `1010` and after this word we put the word $\neg\Psi(x)$ in our operation. It means that we have to go to the vertex `0101`, visit all vertices of $\neg\Psi(x)$ and go back to the word $\Psi(x)$.

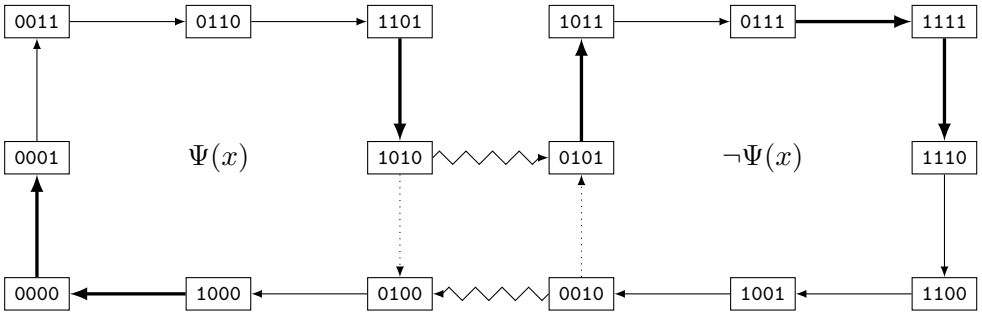

Figure 4: Illustration of the operation $\textsc{Next}(x)$.

Observe that we are no longer using two out of eight common edges — `10100` and `00101` — we replace them with `10101` and `00100`. However, after $\textsc{Next}(y)$ we will also use these two new edges. That is why we have to negate the second output of Next. In fact, this time we first put the word $\neg\Psi(y)$, which ends with `0101`. After this word we have to traverse the whole word $\Psi(y)$ and then go back to the word $\neg\Psi(y)$ with the edge `11011`.

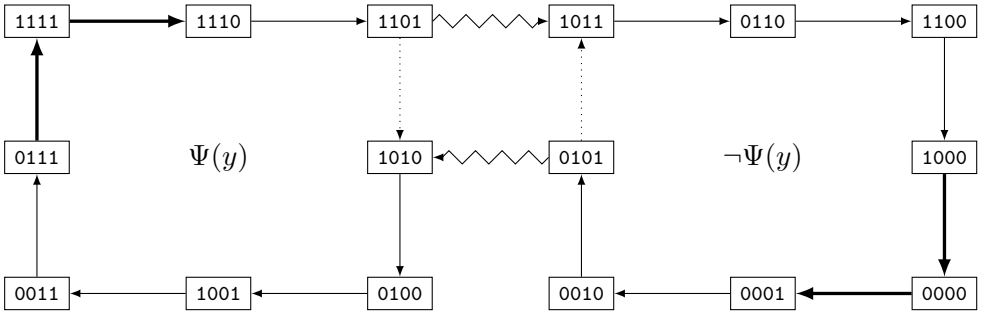

Figure 5: Illustration of the operation $\neg\textsc{Next}(y)$.

When we traverse both graphs and collect last symbols of every vertex we obtain:

$$\textsc{Next}(x) = \texttt{0001101011110010},$$

$$\neg\textsc{Next}(y) = \texttt{1000010100111101}.$$

As you can see, we have only four common edges in $\textsc{Next}(x)$ and $\neg\textsc{Next}(y)$ and they are $Common(n+1)$. It means that these two words are complementary.

5 Constructing abundant de Bruijn words

Assume that a word x contains a single group of p^k ($p \in \{0,1\}$), where k is maximal number of consecutive p's. Define two opertions:

- $add_p(x)$ is the word resulting by adding single bit p in the group p^k
- $rem_p(x)$ is the word resulting by removing a single p in the group p^k

Example 10. $add_1(0110) = 01110,\ rem_0(00010111) = 0010111.$

Define the following operation on two de Bruijn words of rank n, both starting with 0^n:

$$u \otimes w \ = \ rem_0(add_1(u)) \cdot rem_1(add_0(w)).$$

If u, w does not start with 0^n then assume that the operation $\otimes$ stats by shifting these words cyclically to fulfill this condition. In other words in the operation $\otimes$ we move one zero from the first group of 0^n to the second group, and single one from the second group of 1^n to the first group.

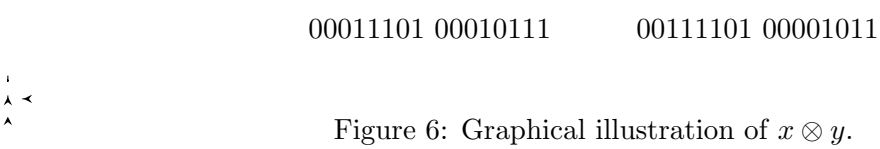

Figure 6: Graphical illustration of $x \otimes y$.

The following lemma reduces construction of abundant de Bruijn words to construction of a pair of complementary words.

Lemma 5. *Assume u, w are complementary de Bruijn words of rank n both starting with 0^n. Then $\mathbf{lin}(u \otimes w)$ is a binary abundant linear de Bruijn word of rank $n+1$.*

Proof. Let $u' = rem_0(add_1(u))$ and $w' = rem_1(add_0(w))$. First we show the following fact:

Claim 2. $u \otimes w \in \mathbf{DB}_{n+1}$.

Note that both u' and w' start with 0^{n-1} and end with 1. It means that they have common prefix of length $n-1$ and common suffix of length 1, so by Observation 2 we have:

$$\mathcal{C}_{n+1}(u \otimes w) \ = \ \mathcal{C}_{n+1}(u') \cup \mathcal{C}_{n+1}(w').$$

Consider how adding and removing bits change the set of cyclic $(n+1)$-factors of words $u, w \in \mathbf{DB}_n$. Operation add_p ($p \in \{0,1\}$) just adds extra $(n+1)$-factor p^{n+1}. Moreover, operation rem_p removes factors $\bar{p}p^n$ and $p^n\bar{p}$ and adds new factor $\bar{p}p^{n-1}\bar{p}$.

Hence:

$$\mathcal{C}_{n+1}(u') = (\mathcal{C}_{n+1}(u) \cup \{10^{n-1}1,\ 1^{n+1}\}) - \{10^n,\ 0^n1\}$$

$$\mathcal{C}_{n+1}(w') = (\mathcal{C}_{n+1}(w) \cup \{01^{n-1}0,\ 0^{n+1}\}) - \{01^n,\ 1^n0\}$$

Notice that every removed factor belongs to $Common(n)$, so each of them still belongs to the union of $\mathcal{C}_{n+1}(u')$ and $\mathcal{C}_{n+1}(w')$. Furthermore, every element of $Unused(n)$ is added, so finally we have:

$$\mathcal{C}_{n+1}(u \otimes w) \ = \ \mathcal{C}_{n+1}(u) \cup \mathcal{C}_{n+1}(w) \cup Unused(n) \ = \ \mathrm{BIN}(n+1).$$

Hence $u \otimes w \in \mathbf{DB}_{n+1}$, which proves the claim.

Now (by Corollary 1) it is enough to show:

Claim 3. *Prefix of* $\mathbf{lin}(u \otimes w)$ *of length* $\Delta_n + 1$ *contains, as a factor, each binary word of size* n*.*

Notice that this prefix is of the form $u' \cdot 0^n$ and its set of n-factors is the same as the set of cyclic n-factors of word $u' \cdot 0$ or any of its shift, e.g. $0 \cdot u' = add_1(u)$. The word u is a de Bruijn word of rank n, so it contains all n-factors. Obviously, adding a single one in the group 1^n does not remove any of n-factors, so $\mathcal{C}_n(add_1(u)) = \mathcal{F}_n(u' \cdot 0^n) = \mathrm{BIN}(n)$. □

We combine the operations $\oplus$, $\otimes$ and complementary sequences as components of our main algorithm:

```
function ConstructAbundant(n)
    if n = 1 then return 01
    if n = 2 then return 00110
    x := y := 0011

    for i = 2 to n − 1 do
        x := SHIFT(x, 1^i); x := NEXT(x)
        comment: x := Ψ(x) ⊕ Ψ(x)‾
        y := SHIFT(y, 1^i); y := ¬NEXT(y)
        comment: y := ¬(Ψ(y) ⊕ Ψ(y)‾)

    return lin(x ⊗ y)
```

Example 11. The cyclic de Bruijn words $u = 00011101$, $w = 00010111$ are complementary. Then the word

$$\mathbf{lin}(u \otimes w) \ = \ 00111101\ 00001011\ 001$$

is the abundant de Bruijn word $\mathbf{U}(4)$. It has size $\Delta_4 = 19$. Its nine longest prefixes are factor-rich.

We have:

$$\begin{aligned}
\text{ConstructAbundant}(3) &= 0111000101,\\
\text{ConstructAbundant}(4) &= 0010111100001101001,\\
\text{ConstructAbundant}(5) &= 000110101111100100000101001110110001.
\end{aligned}$$

Theorem 1. *The word $w = ConstructAbundant(n)$ is a linear de Bruijn word of rank n. It can be constructed in $O(N)$, where $N = |w|$ (N is exponential with respect to n). $O(n)$-sized description of w (as a straight-line program using operations of concatenation, prefix/suffix cutiing and Ψ) can be computed in $O(n)$ time..*

Proof. □

6 Final remarks

By $\mathbf{sh}(x, \alpha)$ we denote the one-step cyclic shift of word x which moves the last letter to the front of the word. Using this simple operation one can shorten the description of operation $NEXT$ and describe it in more syntactic (but rather cryptic) form.

Observation 6. Assume $x \in \mathbf{DB}(n)$ ends with the largest block of 1's. Then

$$NEXT(x) \;=\; \Psi(x \cdot \mathbf{sh}(x)),$$

However we feel that defining it as $NEXT(x) = \Psi(x) \oplus \overline{\Psi(x)}$ was more structural.

Our algorithm not only constructes abundant words in linear time, it also gives compact representation of such words of logarithmic size in terms of straight-line programs using concatenation, taking prefixes/suffixes and operation Ψ.

Our main approach is an application of a powerful arithmetic operation on words related to Lempel homomorphism, see [**?**]. Lempel algorithm for de Bruijn words is an ingenious recursive construction. It is interesting that other algorithms for constructing de Bruijn words are not suitable for the construction of abundant de Bruijn words (we performed computer experiments which confimed that for large graphs).

References

[1] Donald E. Knuth, *The Art of Computer Programming: Generating All Tuples and Permutations*. Stoughton, Massachusetts (2005) 24.

[2] Abraham Lempel, *On a Homomorphism of the de Bruijn Graph and Its Applications to the Design of Feedback Shift Registers*. IEEE Transactions on Computers C-19 (12) (1970) 1204–1209.

[3] Vernica Becher, Pablo Ariel Heiber *On extending de Bruijn sequences*. Information Processing Letters 111 (2011) 930–932.

[4] Jeffrey Shallit, On the maximum number of distinct factors of a binary string. Graphs and Combinatorics 9(2-4): 197-200 (1993)