ECO-based Gray codes generation for particular classes of words

Vincent VAJNOVSZKI
Université de Bourgogne
Le2i, UMR-CNRS 5158

June 25-27, 2012
Previous work

- F. Ruskey, in *ISAAC Conference, LNCS*, 1993
- F. Ruskey, Joe Sawada, Aaron Williams, in *Journal of Combinatorial Theory, Series A* 2012
\( \vartheta : \mathcal{O}_n \to 2^{\mathcal{O}_{n+1}} \)

If the operator \( \vartheta \) satisfies:

1. if \( x_1, x_2 \in \mathcal{O} \), and \( x_1 \neq x_2 \), then \( \vartheta(x_1) \cap \vartheta(x_2) = \emptyset \),
2. for each \( y \in \mathcal{O}_n, n \geq 1 \), there exists a unique \( x \in \mathcal{O}_{n-1} \) such that \( y \in \vartheta(x) \),

then \( \{\vartheta(x)\}_{x \in \mathcal{O}_{n-1}}, n \geq 1 \), is a partition of \( \mathcal{O}_n \) and \( \vartheta \) is called an ECO operator.
ECO (or succession) rules = formal system consisting of

- a root $e_0 \in \Sigma$
- a set of productions of the form

$$\{ ((k) \xrightarrow{} (e_1(k))(e_2(k)) \cdots (e_{|k|}(k)) \}_{k \in \Sigma}$$

which explain how to derive, for any given $k \in \Sigma$, its $|k|$ successors, $(e_1(k)), (e_2(k)), \ldots, (e_{|k|}(k))$
Definition

$p$-ary Dyck words which are binary words with:
- exactly $p - 1$ times as many 0’s as 1’s
- satisfying the $p$-th order suffix property: any suffix has at least $p - 1$ times as many 0’s as 1’s

$D_{np}^p = \text{set of } p$-ary Dyck words of length $np$, and $D_{2n}^2 = D_{2n}$
Definition

If the suffix property condition is dropped, then the obtained word is called Grand Dyck word / $p$-ary Grand Dyck words

$GD_{2n} =$ set of length $2n$ Grand Dyck words

$GD_{np}^p =$ length $np$, $p$-ary Grand Dyck words

The set of length $n$ binary words with exactly $m$ occurrences of 0 is denoted by $C_{n,m}$

$$D_{np}^p \subset GD_{np}^p = C_{np,n(p-1)}$$
Definition

If the suffix property condition is dropped, then the obtained word is called Grand Dyck word / \( p \)-ary Grand Dyck words

\[ GD_{2n} = \text{set of length } 2n \text{ Grand Dyck words} \]

\[ GD_{np}^p = \text{length } np, \ p\text{-ary Grand Dyck words} \]

The set of length \( n \) binary words with exactly \( m \) occurrences of 0 is denoted by \( C_{n,m} \)

\[ D_{np}^p \subset GD_{np}^p = C_{np,n(p-1)} \]
A **Motzkin word** is a word over the alphabet \{0, 1, a\} which after erasing each occurrence of \(a\) gives a Dyck word; and we denote \(M_n\) the set of length \(n\) Motzkin words.

\[ M_n = \text{set of length } n \text{ Motzkin words} \]

A **Schröder word** is a Motzkin word in which each length maximal factor of the form \(aa\ldots a\) has even length.

\[ S_{2n} = \text{the set of length } 2n \text{ Schröder words} \]
Definition

A **Motzkin word** is a word over the alphabet \{0, 1, a\} which after erasing each occurrence of \(a\) gives a Dyck word; and we denote \(M_n\) the set of length \(n\) Motzkin words

\[ M_n = \text{set of length } n \text{ Motzkin words} \]

Definition

A **Schröder word** is a Motzkin word in which each length maximal factor of the form \(aa \ldots a\) has even length

\[ S_{2n} = \text{the set of length } 2n \text{ Schröder words} \]
Definition (T. Walsh)

A **Gray code** is an infinite collection of word-lists, one list for words with same length, such that the number of positions in which two consecutive words in each list differ is bounded (independently of the word-length).

Definition

A Gray code has distance $\delta \geq 1$ if consecutive words differ in at most $\delta$ positions.
Definition (T. Walsh)

A Gray code is an infinite collection of word-lists, one list for words with same length, such that the number of positions in which two consecutive words in each list differ is bounded (independently of the word-length).

Definition

A Gray code has distance $\delta \geq 1$ if consecutive words differ in at most $\delta$ positions.
A Gray code is **circular** if the last and the first word in the list differ in the same way.

A Gray code is called **homogeneous** if the 1 and the 0 that exchange positions are separated only by 0’s.

- $(k)$ or $(\overline{k})$ labeled node in the generating tree corresponds to a word with $k$ successors.
- The successors of a $(\overline{k})$ labeled node are the same as for $(k)$, but in reverse order.

Vincent VAJNOVSZKI

ECO-based Gray codes generation for particular classes of words
A Gray code is **circular** if the last and the first word in the list differ in the same way.

A Gray code is called **homogeneous** if the 1 and the 0 that exchange positions are separated only by 0’s.

$(k)$ or $(\bar{k})$ labeled node in the generating tree corresponds to a word with $k$ successors.

The successors of a $(\bar{k})$ labeled node are the same as for $(k)$, but in reverse order.
Definition (F. Ruskey)

An algorithm for generating a list of words is called CAT (as Constant Average Time) if the number of operations necessary to transform each word into its successor in the list, is constant in average.
The succession rule for $p$-ary Dyck words

$$\begin{cases} 
(1) \\
(k) \leadsto (p)(p+1)\cdots(p+k-1)
\end{cases}$$
ith return rule?

110010111000

Vincent VAJNOVSZKI

ECO-based Gray codes generation for particular classes of words
ith return rule?

110010111000
ith return rule?

110010111000 $\rightarrow$ 11100101110000
ith return rule?

110010111000 → 11100101110000
ith return rule?

\[110010111000 \rightarrow 11100101110000 \rightarrow 11001101110000 \rightarrow 11001101110000\]
What is the return rule?

\[
\begin{align*}
110010111000 & \longrightarrow 11100101110000 \\
& \longrightarrow 11001101110000 \\
\end{align*}
\]
110010111000 $\rightarrow$ 11100101110000
$\rightarrow$ 11001101110000
$\rightarrow$ 11001011110000
$\rightarrow$ 110010111110000
ith return rule?

\[ \begin{align*}
110010111000 & \rightarrow 11100101110000 \\
& \rightarrow 11001101110000 \\
& \rightarrow 11001011110000 \\
& \rightarrow 11001011110000
\end{align*} \]
ith return rule?

110010111000 $\rightarrow$ 11100101110000
$\rightarrow$ 11001101110000
$\rightarrow$ 11001011110000
$\rightarrow$ 11001011110010
110010111000
Last descent rule!

110010111000

Vincent VAJNOVSZKI

ECO-based Gray codes generation for particular classes of words
Last descent rule!

110010111000 $\rightarrow$ 11001011110000

Vincent VAJNOVSZKI

ECO-based Gray codes generation for particular classes of words
Last descent rule!

110010111000  $\rightarrow$  11001011110000  
$\rightarrow$  11001011101000
Last descent rule!

110010111000 $\longrightarrow$ 11001011110000
$\longrightarrow$ 11001011101000
$\longrightarrow$ 11001011100100
$\longrightarrow$ 11001011100100
Last descent rule!

110010111000 → 11001011110000 → 11001011101000 → 11001011100100 → 11001011100010
Let $d = d_1d_2 \ldots d_{np}$ be a $p$-ary Dyck word of length $np$, and $\ell$ be its length-maximal 0’s suffix.

\[
d = d_1d_2 \ldots d_{np-\ell}0^\ell
\]

- $d_{np-\ell} = 1$
- the suffix $0^\ell$ is called the last descent of $d$

For each $u$, $0 \leq u \leq \ell$

\[
d_1d_2 \ldots d_{np-\ell}0^u10^{\ell-u}0^{p-1}
\]

is a $p$-ary Dyck word of length $(n + 1)p$ called a successor of $d$

last descent rule
Let $d = d_1d_2 \ldots d_{np}$ be a $p$-ary Dyck word of length $np$, and $\ell$ be its length-maximal 0’s suffix.

$$d = d_1d_2 \ldots d_{np-\ell}0^\ell$$

- $d_{np-\ell} = 1$
- the suffix $0^\ell$ is called the last descent of $d$

For each $u$, $0 \leq u \leq \ell$

$$d_1d_2 \ldots d_{np-\ell}0^u10^{\ell-u}0^{p-1}$$

is a $p$-ary Dyck word of length $(n + 1)p$ called a successor of $d$

last descent rule
Let $d = d_1 d_2 \ldots d_{np}$ be a $p$-ary Dyck word of length $np$, and $\ell$ be its length-maximal 0’s suffix.

$$d = d_1 d_2 \ldots d_{np-\ell} 0^\ell$$

- $d_{np-\ell} = 1$
- the suffix $0^\ell$ is called the last descent of $d$

For each $u$, $0 \leq u \leq \ell$

$$d_1 d_2 \ldots d_{np-\ell} 0^u 1^\ell-u 0^{p-1}$$

is a $p$-ary Dyck word of length $(n + 1)p$ called a successor of $d$

last descent rule
Let $d = d_1d_2 \ldots d_{np}$ be a $p$-ary Dyck word of length $np$, and $\ell$ be its length-maximal 0’s suffix.

$$d = d_1d_2 \ldots d_{np-\ell}0^\ell$$

- $d_{np-\ell} = 1$
- the suffix $0^\ell$ is called the last descent of $d$

For each $u$, $0 \leq u \leq \ell$

$$d_1d_2 \ldots d_{np-\ell}0^u10^{\ell-u}0^{p-1}$$

is a $p$-ary Dyck word of length $(n+1)p$ called a successor of $d$
Let $d = d_1 d_2 \ldots d_{np}$ be a $p$-ary Dyck word of length $np$, and $\ell$ be its length-maximal 0’s suffix.

$$d = d_1 d_2 \ldots d_{np-\ell} 0^\ell$$

- $d_{np-\ell} = 1$
- The suffix $0^\ell$ is called the last descent of $d$

For each $u$, $0 \leq u \leq \ell$

$$d_1 d_2 \ldots d_{np-\ell} 0^u 10^{\ell-u} 0^{p-1}$$

is a $p$-ary Dyck word of length $(n + 1)p$ called a successor of $d$

last descent rule
(k) corresponds to a Dyck word with its last descent of length \(k - 1\). Its (ordered list of) successors are obtained by:
- inserting from right to left a 1 into its last descent
- adding a \(0^{p-1}\) suffix

(\(\overline{k}\)) labeled node are the same but in reverse order
(k) corresponds to a Dyck word with its last descent of length $k - 1$. Its (ordered list of) successors are obtained by:
- inserting \textit{from right to left} a 1 into its last descent
- adding a $0^{p-1}$ suffix
\( \overline{k} \) labeled node are the same but in reverse order
\((k)\) corresponds to a Dyck word with its last descent of length \(k - 1\). Its (ordered list of) successors are obtained by:
- inserting from right to left a 1 into its last descent
- adding a \(0^{p-1}\) suffix

\((\overline{k})\) labeled node are the same but in reverse order
Theorem

The succession rule

\[
\begin{cases}
(1) \\
(k) \leadsto (p)(p+1) \cdots (p+k-1)
\end{cases}
\]

gives a circular Gray code for p-ary Dyck words, where the root of the generating tree is the empty word \( \epsilon \)
ECO-based Gray codes generation for particular classes of words
Proposition

For \( p \geq 2 \), the first and last word given by this succession rule are

1. \((10^{p-1})^n, n \geq 1, \text{ and}\)
2. \(110^{2p-2}(10^{p-1})^{n-2}, n \geq 2\)

Remark

For \( p = 2 \), the Gray code induced on \( D_{2n} \) by this succession rule is the reverse of Ruskey-Proskurowski’s Gray code (1990)
Proposition

For \( p \geq 2 \), the first and last word given by this succession rule are

\[
(10^{p-1})^n, \ n \geq 1, \text{ and }
\]
\[
110^{2p-2}(10^{p-1})^{n-2}, \ n \geq 2
\]

Remark

For \( p = 2 \), the Gray code induced on \( D_{2n} \) by this succession rule is the reverse of Ruskey-Proskurowski’s Gray code (1990)
More restrictive Gray codes

Theorem

The succession rule

\[
\begin{cases}
(1)_a \\
(k_a) \leadsto (p_a) (p+1)_b (p+2)_b \cdots (p+k-1)_b
\end{cases}
\]
More restrictive Gray codes

**Theorem**

*The succession rule*

\[
\begin{cases}
(1)_a \\
(k_a) \sim (p_a) (p + 1)_b (p + 2)_b \cdots (p + k - 1)_b \\
(k_b) \sim (p + k - 1)_a (p_a) (p + 1)_b (p + 2)_b \cdots (p + k - 2)_b
\end{cases}
\]

gives a **homogeneous** Gray code for $p$-ary Dyck words, where the root of the generating tree is the empty word $\epsilon$. 
Proposition

The first and last length $np$ word given by this succession rule are

- $(10^{p-1})^n$
- $1^n0(p-1)^n$

Remark

The Gray code induced on $D_{np}^p$ by the succession rule is Eades-McKay-Bultena-Ruskey’s Gray code (1998)
Proposition

The first and last length $np$ word given by this succession rule are

- $(10^{p-1})^n$
- $1^n0(p-1)^n$

Remark

The Gray code induced on $D_{np}^p$ by the succession rule is Eades-McKay-Bultena-Ruskey’s Gray code (1998)
ith descent rule
ith descent rule

01011000
ith descent rule

01011000
Combinations and Grand Dyck words

ith descent rule

01011000 → 010111000
Combinations and Grand Dyck words

\( i \text{th descent rule} \)

\[
\begin{align*}
01011000 & \rightarrow 010111000 \\
& \rightarrow 010110100 \\
& \rightarrow 010110100
\end{align*}
\]
Combinations and Grand Dyck words

\textit{ith descent rule}

\begin{align*}
01011000 & \rightarrow 010111000 \\
& \rightarrow 010110100 \\
& \rightarrow 010110010
\end{align*}
Combinations and Grand Dyck words

ith descent rule

01011000 → 010111000
→ 010110100
→ 010110010
→ 010110001
Let \( d = d_1d_2 \ldots d_n \) be a binary word in \( C_{n,m} \), and \( \ell \) be its length-maximal 0’s suffix

\[
d = d_1d_2 \ldots d_{n-\ell}0^\ell
\]

- \( d_{n-\ell} = 1 \)
- the suffix \( 0^\ell \) is called the last descent of \( d \)

For each \( u, 0 \leq u \leq \ell \),

\[
d_1d_2 \ldots d_{n-\ell}0^u10^{\ell-u}
\]

is a length \((n + 1)\) binary word in \( C_{n+1,m} \), called a successor of \( d \).

last descent rule
Let $d = d_1 d_2 \ldots d_n$ be a binary word in $C_{n,m}$, and $\ell$ be its length-maximal 0’s suffix

$$d = d_1 d_2 \ldots d_{n-\ell} 0^\ell$$

- $d_{n-\ell} = 1$
- the suffix $0^\ell$ is called the last descent of $d$

For each $u$, $0 \leq u \leq \ell$,

$$d_1 d_2 \ldots d_{n-\ell} 0^u 1^\ell-u$$

is a length $(n + 1)$ binary word in $C_{n+1,m}$, called a successor of $d$.

last descent rule
Let $d = d_1 d_2 \ldots d_n$ be a binary word in $C_{n,m}$, and $\ell$ be its length-maximal 0’s suffix

\[ d = d_1 d_2 \ldots d_{n-\ell} 0^\ell \]

- $d_{n-\ell} = 1$
- the suffix $0^\ell$ is called the last descent of $d$

For each $u$, $0 \leq u \leq \ell$,

\[ d_1 d_2 \ldots d_{n-\ell} 0^u 1^{\ell-u} \]

is a length $(n + 1)$ binary word in $C_{n+1,m}$, called a successor of $d$.

last descent rule
Let \( d = d_1d_2 \ldots d_n \) be a binary word in \( C_{n,m} \), and \( \ell \) be its length-maximal 0’s suffix

\[
d = d_1d_2 \ldots d_{n-\ell}0^\ell
\]

- \( d_{n-\ell} = 1 \)
- the suffix \( 0^\ell \) is called the last descent of \( d \)

For each \( u, 0 \leq u \leq \ell \),

\[
d_1d_2 \ldots d_{n-\ell}0^u10^{\ell-u}
\]

is a length \( (n + 1) \) binary word in \( C_{n+1,m} \), called a successor of \( d \).
Let \( d = d_1 d_2 \ldots d_n \) be a binary word in \( C_{n,m} \), and \( \ell \) be its length-maximal 0’s suffix

\[
d = d_1 d_2 \ldots d_{n-\ell} 0^\ell
\]

- \( d_{n-\ell} = 1 \)
- the suffix \( 0^\ell \) is called the last descent of \( d \)

For each \( u, 0 \leq u \leq \ell \),

\[
d_1 d_2 \ldots d_{n-\ell} 0^u 1^\ell-u
\]

is a length \((n + 1)\) binary word in \( C_{n+1,m} \), called a successor of \( d \).

last descent rule
Theorem

For a fixed $m \geq 1$, the succession rule

\[
\begin{cases}
(m + 1) \\
(k) \sim (1)(2) \cdots (k)
\end{cases}
\]

gives a (circular) Gray code for $C_{n,m}$, $n \geq m$ where the size zero object is $0^m$
ECO-based Gray codes generation for particular classes of words
In particular, for a given $p \geq 2$, when $n = \frac{m}{p-1} \cdot p$, (that is, at level $n - m$ in the generating tree) the succession this rule yields a Gray code for $GD_n^p = C_{n,m}$.

Remark

The Gray code induced on $C_{n,m}$ by the succession rule (9) is the revolving door Gray code Liu-Tang/Nijenhuis-Wilf
In particular, for a given $p \geq 2$, when $n = \frac{m}{p-1} \cdot p$, (that is, at level $n - m$ in the generating tree) the succession this rule yields a Gray code for $GD_n^p = C_{n,m}$.

**Remark**

*The Gray code induced on $C_{n,m}$ by the succession rule (9) is the revolving door Gray code Liu-Tang/Nijenhuis-Wilf*
Motzkin words

Last descent rule?
Motzkin words

Last descent rule?

1a1a1aa000

ECO-based Gray codes generation for particular classes of words
Motzkin words

Last descent rule?

1a1a1aa000
Motzkin words

Last descent rule?

1a1a1aa000 $\rightarrow$ 1a1a1aa000a
Motzkin words

Last descent rule?

1a1a1aa000 → 1a1a1aa000a
   → 1a1a1aa00a0
Motzkin words

Last descent rule?

$1 a1a1aa000 \rightarrow 1a1a1aa000a$

$\rightarrow 1a1a1aa00a0$

$\rightarrow 1a1a1aa0a00$
Motzkin words

**Last descent rule?**

\[ 1a1a1aa000 \rightarrow 1a1a1aa000a \]
\[ \rightarrow 1a1a1aa00a0 \]
\[ \rightarrow 1a1a1aa0a00 \]
\[ \rightarrow 1a1a1aaa000 \]
Motzkin words

Last descent rule?

1a1a1aa000 → 1a1a1aa000a
→ 1a1a1aa00a0
→ 1a1a1aa0a00
→ 1a1a1aa000
→ 1a1a1aa00010
Motzkin words

Last descent rule?

\[1a1a1aa000 \rightarrow 1a1a1aa000a\]
\[\rightarrow 1a1a1aa00a0\]
\[\rightarrow 1a1a1aa0a00\]
\[\rightarrow 1a1a1aa0000\]
\[\rightarrow 1a1a1aa00010\]
\[\rightarrow 1a1a1aa00100\]
Motzkin words

Last descent rule?

1a1a1aa000 \rightarrow 1a1a1aa000a
\rightarrow 1a1a1aa00a0
\rightarrow 1a1a1aa0a00
\rightarrow 1a1a1aa000a0
\rightarrow 1a1a1aa00010
\rightarrow 1a1a1aa00100
\rightarrow 1a1a1aa01000
Motzkin words

Last descent rule?

\[11a1aa000 \rightarrow 11a1aa000a\]
\[\rightarrow 11a1aa00a0\]
\[\rightarrow 11a1aa0a00\]
\[\rightarrow 11a1aa0000\]
\[\rightarrow 11a1aa00010\]
\[\rightarrow 11a1aa00100\]
\[\rightarrow 11a1aa01000\]
\[\rightarrow 11a1aa10000\]
Motzkin words

Last descent rule?

$1a1a1aa000 \rightarrow 1a1a1aa000a$

$\rightarrow 1a1a1aa00a0$

$\rightarrow 1a1a1aa0a00$

$\rightarrow 1a1a1aaa000$

$\rightarrow 1a1a1aa00010$

$\rightarrow 1a1a1aa00100$

$\rightarrow 1a1a1aa01000$

$\rightarrow 1a1a1aa10000$
Last ascent-free rule!

Let $w \in M_n$ be a length-$n$ Motzkin word, $s$ its length-maximal suffix which does not contain the letter 1, and

$$k = |s|_a + 1 = \text{the number of successors of } w$$
Last ascent-free rule!

Let $w \in M_n$ be a length-$n$ Motzkin word, $s$ its length-maximal suffix which does not contain the letter 1, and

$$k = |s|_a + 1 = \text{the number of successors of } w$$

The word $wa$ is a Motzkin word of length $n + 1$ with $k + 1$ successors;
Last ascent-free rule!

Let $w \in M_n$ be a length-$n$ Motzkin word, $s$ its length-maximal suffix which does not contain the letter 1, and

$$k = |s|_a + 1 = \text{the number of successors of } w$$

- The word $wa$ is a Motzkin word of length $n + 1$ with $k + 1$ successors;
- If $|s|_a > 0$, then for a given $j$, $|s|_a \geq j > 0$ let $w'as''$ be the factorization of $w$ where $s''$ is the suffix of $w$ with exactly $j - 1$ occurrences of $a$. The word $v = w'1s''0$ is a Motzkin word of length $n + 1$; it has $j$ successors.
11a01a0a0
11a01a0a0
$11a01a0a0 \rightarrow 11a01a0a0a$
$11a01a0a0 \rightarrow 11a01a0a0a$
$\rightarrow 11a0110a00$
Vincent VAJNOVSZKI

ECO-based Gray codes generation for particular classes of words
Theorem

The succession rule

\[
\begin{cases}
(1) \\
(k) \leadsto (1)(2) \cdots (k-1)(k+1)
\end{cases}
\]

gives a Gray code for $M_n$ with distance 4
Let $w \in S_{2n}$, and $k$ the length of its length-maximal 0’s suffix, plus one; $w = w'0^{k-1}$


$w$ has $2k$ successors and the following gives the successors of $w$ (an ECO operator for the set of Schröder words)

- $waa \in S_{2n+2}$; it has with 2 successors
- $w'100^{k-1} \in S_{2n+2}$; it has $2k + 2$ successors

In addition, if $k > 1$, then for any $j, 1 \leq j \leq k - 1$ we have

- $w'0^{k-1-j}aa0^j \in S_{2n+2}$; it has $2j + 2$ successors
- $w'0^{k-j}100^{j-1} \in S_{2n+2}$; it has $2j + 2$ successors
Let $w \in S_{2n}$, and $k$ the length of its length-maximal 0’s suffix, plus one; $w = w'0^{k-1}$

$w$ has $2k$ successors and the following gives the successors of $w$ (an ECO operator for the set of Schröder words)

- $waa \in S_{2n+2}$; it has with $2$ successors
- $w'10^{k-1} \in S_{2n+2}$; it has $2k + 2$ successors

In addition, if $k > 1$, then for any $j$, $1 \leq j \leq k - 1$ we have

- $w'0^{k-1-j}aa0^j \in S_{2n+2}$; it has $2j + 2$ successors
- $w'0^{k-j}100^{j-1} \in S_{2n+2}$; it has $2j + 2$ successors
Let $w \in S_{2n}$, and $k$ the length of its length-maximal 0’s suffix, plus one; $w = w'0^{k-1}$

$w$ has $2k$ successors and the following gives the successors of $w$ (an ECO operator for the set of Schröder words)

- $waa \in S_{2n+2}$; it has with 2 successors
- $w'100^{k-1} \in S_{2n+2}$; it has $2k + 2$ successors

In addition, if $k > 1$, then for any $j, 1 \leq j \leq k - 1$ we have

- $w'0^{k-1-j}aa0^j \in S_{2n+2}$; it has $2j + 2$ successors
- $w'0^{k-j}100^{j-1} \in S_{2n+2}$; it has $2j + 2$ successors
Let \( w \in S_{2n} \), and \( k \) the length of its length-maximal 0’s suffix, plus one; \( w = w'0^{k-1} \).

\( w \) has \( 2k \) successors and the following gives the successors of \( w \) (an ECO operator for the set of Schröder words):

- \( waa \in S_{2n+2} \); it has with 2 successors
- \( w'100^{k-1} \in S_{2n+2} \); it has \( 2k + 2 \) successors

In addition, if \( k > 1 \), then for any \( j, 1 \leq j \leq k - 1 \) we have

- \( w'0^{k-1-j}aa0^j \in S_{2n+2} \); it has \( 2j + 2 \) successors
- \( w'0^{k-j}100^{j-1} \in S_{2n+2} \); it has \( 2j + 2 \) successors
Let \( w \in S_{2n} \), and \( k \) the length of its length-maximal 0’s suffix, plus one; \( w = w'0^{k-1} \).

\( w \) has \( 2k \) successors and the following gives the successors of \( w \) (an ECO operator for the set of Schröder words):

- \( waa \in S_{2n+2} \); it has 2 successors
- \( w'100^{k-1} \in S_{2n+2} \); it has \( 2k + 2 \) successors

In addition, if \( k > 1 \), then for any \( j, 1 \leq j \leq k - 1 \) we have

- \( w'0^{k-1-j}aa0^j \in S_{2n+2} \); it has \( 2j + 2 \) successors
- \( w'0^{k-j}100^{j-1} \in S_{2n+2} \); it has \( 2j + 2 \) successors
Let $w \in S_{2n}$, and $k$ the length of its length-maximal 0’s suffix, plus one; $w = w'0^{k-1}$.

$w$ has $2k$ successors and the following gives the successors of $w$ (an ECO operator for the set of Schröder words):

- $waa \in S_{2n+2}$; it has with 2 successors
- $w'100^{k-1} \in S_{2n+2}$; it has $2k + 2$ successors

In addition, if $k > 1$, then for any $j$, $1 \leq j \leq k - 1$ we have

- $w'0^{k-1-j}aa0^j \in S_{2n+2}$; it has $2j + 2$ successors
- $w'0^{k-j}100^{j-1} \in S_{2n+2}$; it has $2j + 2$ successors
11aa1aa000
11aa1aa000
last descent rule

\[ 11aa1aa000 \rightarrow 11aa1aa000aa \]
last descent rule

11aa1aa000 → 11aa1aa000aa
     →  11aa1aa00aa0
last descent rule

11aa1aa000 → 11aa1aa000aa
     → 11aa1aa00aa0
     → 11aa1aa0aa00
Vincent VAJNOVSZKI

ECO-based Gray codes generation for particular classes of words
last descent rule

\[
\begin{align*}
11aa1aa000 & \rightarrow 11aa1aa000aa \\
& \rightarrow 11aa1aa00aa0 \\
& \rightarrow 11aa1aa0aa00 \\
& \rightarrow 11aa1aaa0000 \\
& \rightarrow 11aa1aa00010
\end{align*}
\]
last descent rule

\[11aa1aa000 \rightarrow 11aa1aa000aa\]
\[\rightarrow 11aa1aa00aa0\]
\[\rightarrow 11aa1aa0aa00\]
\[\rightarrow 11aa1aaa000\]
\[\rightarrow 11aa1aa00010\]
\[\rightarrow 11aa1aa00100\]
last descent rule

11aa1aa000 → 11aa1aa000aa
   → 11aa1aa00aa0
   → 11aa1aa0aa00
   → 11aa1aaaa000
   → 11aa1aa00010
   → 11aa1aa00100
   → 11aa1aa01000
last descent rule

\[ 11aa1aa000 \rightarrow 11aa1aa000aa \]
\[ \rightarrow 11aa1aa00aa0 \]
\[ \rightarrow 11aa1aa0aa00 \]
\[ \rightarrow 11aa1aaaa000 \]
\[ \rightarrow 11aa1aa00010 \]
\[ \rightarrow 11aa1aa00100 \]
\[ \rightarrow 11aa1aa01000 \]
\[ \rightarrow 11aa1aa10000 \]
(8)

11aa1aa000 $\longrightarrow$ 11aa1aa000aa
$\longrightarrow$ 11aa1aa00aa0
$\longrightarrow$ 11aa1aa0aa00
$\longrightarrow$ 11aa1aaaa000
$\longrightarrow$ 11aa1aa00010
$\longrightarrow$ 11aa1aa00100
$\longrightarrow$ 11aa1aa01000
$\longrightarrow$ 11aa1aa10000

Vincent VAJNOVSZKI  
ECO-based Gray codes generation for particular classes of words
last descent rule

\[(8)\]

\[11aa1aa000 \rightarrow 11aa1aa000aa\]  \hspace{1cm} (2)

\[\rightarrow 11aa1aa00aa0\]

\[\rightarrow 11aa1aa0aa00\]

\[\rightarrow 11aa1aaaaa000\]

\[\rightarrow 11aa1aa00010\]

\[\rightarrow 11aa1aa00100\]

\[\rightarrow 11aa1aa01000\]

\[\rightarrow 11aa1aa10000\]
last descent rule

(8)

11aa1aa000 \rightarrow 11aa1aa000aa \quad (2)
\rightarrow 11aa1aa00aa0 \quad (4)
\rightarrow 11aa1aa0aa00
\rightarrow 11aa1aaaa000
\rightarrow 11aa1aa00010
\rightarrow 11aa1aa00100
\rightarrow 11aa1aa01000
\rightarrow 11aa1aa10000
last descent rule

(8)

\[ 11\text{aa}1\text{aa}000 \rightarrow 11\text{aa}1\text{aa}000\text{aa} \quad (2) \]
\[ \rightarrow 11\text{aa}1\text{aa}00\text{aa}0 \quad (4) \]
\[ \rightarrow 11\text{aa}1\text{aa}0\text{aa}00 \quad (6) \]
\[ \rightarrow 11\text{aa}1\text{aa}\text{aa}000 \]
\[ \rightarrow 11\text{aa}1\text{aa}00010 \]
\[ \rightarrow 11\text{aa}1\text{aa}00100 \]
\[ \rightarrow 11\text{aa}1\text{aa}01000 \]
\[ \rightarrow 11\text{aa}1\text{aa}10000 \]
last descent rule

\[
\begin{align*}
11aa1aa000 & \rightarrow 11aa1aa000aa & (2) \\
& \rightarrow 11aa1aa00aa0 & (4) \\
& \rightarrow 11aa1aa0aa00 & (6) \\
& \rightarrow 11aa1aaaa000 & (8) \\
& \rightarrow 11aa1aa00010 \\
& \rightarrow 11aa1aa00100 \\
& \rightarrow 11aa1aa01000 \\
& \rightarrow 11aa1aa10000
\end{align*}
\]
last descent rule

\(11aa1aa000\)  \(\rightarrow\)  \(11aa1aa000aa\)  \(\text{(2)}\)
\(\rightarrow\)  \(11aa1aa00aa0\)  \(\text{(4)}\)
\(\rightarrow\)  \(11aa1aa0aa00\)  \(\text{(6)}\)
\(\rightarrow\)  \(11aa1aaaa000\)  \(\text{(8)}\)
\(\rightarrow\)  \(11aa1aa00010\)  \(\text{(4)}\)
\(\rightarrow\)  \(11aa1aa00100\)
\(\rightarrow\)  \(11aa1aa01000\)
\(\rightarrow\)  \(11aa1aa10000\)
last descent rule

\[
\begin{align*}
11a\bar{a}1a\bar{a0}000 &\rightarrow 11a\bar{a}1a\bar{a0}000a\bar{a} & (2) \\
&\rightarrow 11a\bar{a}1a\bar{a0}000a\bar{a}0 & (4) \\
&\rightarrow 11a\bar{a}1a\bar{a0}a\bar{a}00 & (6) \\
&\rightarrow 11a\bar{a}1a\bar{a}a\bar{a}000 & (8) \\
&\rightarrow 11a\bar{a}1a\bar{a}00010 & (4) \\
&\rightarrow 11a\bar{a}1a\bar{a}00100 & (6) \\
&\rightarrow 11a\bar{a}1a\bar{a}01000 \\
&\rightarrow 11a\bar{a}1a\bar{a}10000
\end{align*}
\]
last descent rule

(8)

\[ 11aa1aa000 \quad \rightarrow \quad 11aa1aa000aa \quad (2) \]
\[ \quad \rightarrow \quad 11aa1aa00aa0 \quad (4) \]
\[ \quad \rightarrow \quad 11aa1aa0aa00 \quad (6) \]
\[ \quad \rightarrow \quad 11aa1aaa000 \quad (8) \]
\[ \quad \rightarrow \quad 11aa1aa00010 \quad (4) \]
\[ \quad \rightarrow \quad 11aa1aa00100 \quad (6) \]
\[ \quad \rightarrow \quad 11aa1aa01000 \quad (8) \]
\[ \quad \rightarrow \quad 11aa1aa10000 \]
last descent rule

\[ 11aa1aa000 \rightarrow 11aa1aa000aa \quad (2) \]
\[ \rightarrow 11aa1aa00aa0 \quad (4) \]
\[ \rightarrow 11aa1aa0aa00 \quad (6) \]
\[ \rightarrow 11aa1aaaa000 \quad (8) \]
\[ \rightarrow 11aa1aa00010 \quad (4) \]
\[ \rightarrow 11aa1aa00100 \quad (6) \]
\[ \rightarrow 11aa1aa01000 \quad (8) \]
\[ \rightarrow 11aa1aa10000 \quad (10) \]
Theorem

The succession rule

\[
\begin{align*}
(2) & \sim (2)(4)(4)(6)(6) \cdots (2k)(2k)(2k + 2) \\
\end{align*}
\]

gives a Gray code for $S_n$ with distance 5
Theorem

The succession rule

\[
\left\{
\begin{aligned}
(2) \\
(2k) \rightsquigarrow (2)(4)(4)(6)(6) \cdots (2k)(2k)(2k+2)
\end{aligned}
\right.
\]

gives a Gray code for \( S_n \) with distance 5

- \((\bar{j}) : \ldots aa\ldots\)
- \((j) : \ldots 10\ldots\)
Algorithmic implementation

\[
\left\{ \begin{array}{l}
(1) \\
(k) \rightsquigarrow (p)(p + 1) \cdots (p + k - 1)
\end{array} \right.
\]

gen_Dyck_up generates $D_{np}^p$

- words are stored in the global array $d$
- initialized by $0^{np}$
- main call is: gen_Dyck_up(0, 1)
  
  0 is the size
  1 is the degree

- gen_Dyck_down
  
  executes the statements of gen_Dyck_up in reverse order
  replaces the calls of gen_Dyck_up by gen_Dyck_down and vice-versa
Algorithmic implementation

\[
\left\{ \begin{array}{l}
(1) \\
(k) \mapsto (p)(p+1) \cdots (p+k-1)
\end{array} \right.
\]

gen_Dyck_up generates $D_{np}^p$

- words are stored in the global array $d$
- initialized by $0^{np}$
- main call is: gen_Dyck_up $(0, 1)$
  - 0 is the size
  - 1 is the degree

- gen_Dyck_down
  executes the statements of gen_Dyck_up in reverse order
  replaces the calls of gen_Dyck_up by gen_Dyck_down and vice-versa
Algorithmic implementation

\[
\begin{align*}
(1) & \rightarrow (p)(p+1) \cdots (p+k-1) \\
(2) & \rightarrow (p)(p+1) \cdots (p+k-1)
\end{align*}
\]

g_{\text{Dyck up}} \text{ generates } D_{np}^p

- words are stored in the global array \( d \)
- initialized by \( 0^{np} \)
- main call is: \( g_{\text{Dyck up}}(0, 1) \)
  
  0 is the size  
  1 is the degree

\( g_{\text{Dyck down}} \)

executes the statements of \( g_{\text{Dyck up}} \) in reverse order
replaces the calls of \( g_{\text{Dyck up}} \) by \( g_{\text{Dyck down}} \) and vice-versa
Algorithmic implementation

\[
\begin{cases}
(1) \\
(k) \leadsto (p)(p+1) \cdots (p+k-1)
\end{cases}
\]

gen_Dyck_up generates \( D_{np}^p \)

- words are stored in the global array \( d \)
- initialized by \( 0^{np} \)
- main call is: \( \text{gen_Dyck_up}(0, 1) \)
  
  0 is the size
  1 is the degree

- \( \text{gen_Dyck_down} \)
  
  executes the statements of \( \text{gen_Dyck_up} \) in reverse order
  replaces the calls of \( \text{gen_Dyck_up} \) by \( \text{gen_Dyck_down} \) and vice-versa

Vincent VAJNOVSZKI

ECO-based Gray codes generation for particular classes of words
Algorithmic implementation

\[
\begin{cases}
(1) \\
(k) \mapsto (p)(p+1)\cdots(p+k-1)
\end{cases}
\]

\texttt{gen\_Dyck\_up} generates \(D_{np}^p\)

- words are stored in the global array \(d\)
- initialized by \(0^{np}\)
- main call is: \texttt{gen\_Dyck\_up} \((0, 1)\)
  
  0 is the size
  1 is the degree

\texttt{gen\_Dyck\_down}

executes the statements of \texttt{gen\_Dyck\_up} in reverse order

replaces the calls of \texttt{gen\_Dyck\_up} by \texttt{gen\_Dyck\_down} and vice-versa
\begin{equation}
\begin{cases}
(1) \\
(k) \Rightarrow (p)(p+1) \cdots (p+k-1)
\end{cases}
\end{equation}

\textit{gen\_Dyck\_up} generates $D_{np}^p$

- words are stored in the global array $d$
- initialized by $0^{np}$
- main call is: \textit{gen\_Dyck\_up}(0, 1)
  
  0 is the size  
  1 is the degree

\textit{gen\_Dyck\_down}

executes the statements of \textit{gen\_Dyck\_up} in reverse order
replaces the calls of \textit{gen\_Dyck\_up} by \textit{gen\_Dyck\_down} and
vice-versa
Algorithmic implementation

\[
\left\{
\begin{array}{l}
(1) \\
(k) \mapsto (p)(p + 1) \cdots (p + k - 1)
\end{array}
\right.
\]

`gen_Dyck_up` generates \( D_{np}^p \)

- words are stored in the global array \( d \)
- initialized by \( 0^{np} \)
- main call is: `gen_Dyck_up(0, 1)`
  - 0 is the size
  - 1 is the degree

`gen_Dyck_down`

executes the statements of `gen_Dyck_up` in reverse order
replaces the calls of `gen_Dyck_up` by `gen_Dyck_down` and vice-versa
procedure gen_Dyck_up(size, k)
local i;
if size = n · p then Print(d);
else d[size + 1] := 1;
gen_Dyck_up(size + p, p);
d[size + 1] := 0;
for i from p + 1 to k + p − 1 do
d[size + p + 1 − i] := 1;
gen_Dyck_down(size + p, i);
d[size + p + 1 − i] := 0;
end do
end if
end procedure.
Vincent VAJNOVSZKI

ECO-based Gray codes generation for particular classes of words
Proposition

Procedure `gen_Dyck_up` generates $p$-ary Dyck words of length $np$ in constant average time.

- the total amount of computation in each call is proportional with the number of direct calls produced by this call
- each non-terminal call, except the root, produces at least two recursive calls (i.e., there is no call of degree one, except to the main call)
- each terminal call (degree-zero call) produces a new permutation
Proposition

Procedure \texttt{gen\_Dyck\_up} generates $p$-ary Dyck words of length $np$ in constant average time.

- the total amount of computation in each call is proportional with the number of direct calls produced by this call
- each non-terminal call, except the root, produces at least two recursive calls (i.e., there is no call of degree one, except to the main call)
- each terminal call (degree-zero call) produces a new permutation
Proposition

Procedure \texttt{gen\_Dyck\_up} generates \(p\)-ary Dyck words of length \(np\) in constant average time.

- the total amount of computation in each call is proportional with the number of direct calls produced by this call
- each non-terminal call, except the root, produces at least two recursive calls (i.e., there is no call of degree one, except to the main call)
- each terminal call (degree-zero call) produces a new permutation
Proposition

Procedure $\text{gen\_Dyck\_up}$ generates $p$-ary Dyck words of length $np$ in constant average time.

- the total amount of computation in each call is proportional with the number of direct calls produced by this call
- each non-terminal call, except the root, produces at least two recursive calls (i.e., there is no call of degree one, except to the main call)
- each terminal call (degree-zero call) produces a new permutation
Remark

*Similar algorithms can be designed for*

- homogeneous Gray code for $p$-ary Dyck words
- Motzkin words
- Schröder words
\( C_{n,m} \)

\[
\begin{cases}
(m + 1) \\
(k) \mapsto (1)(\bar{2}) \cdots (k)
\end{cases}
\]

The implementation of this succession rule does not give a CAT algorithm.

**Remark (numerical evidences)**

For an integer \( p \geq 2 \) and for \( n = \frac{mp}{p-1} \) we have

\[
\frac{\text{total amount of computation}}{\text{number of generated words}} \leq 2
\]

and this rule yields a CAT algorithm for the set of \( p \)-ary Grand Dyck words
The implementation of this succession rule does not give a CAT algorithm.

Remark (numerical evidences)

For an integer \( p \geq 2 \) and for \( n = \frac{mp}{p-1} \) we have

\[
\frac{\text{total amount of computation}}{\text{number of generated words}} \leq 2
\]

and this rule yields a CAT algorithm for the set of \( p \)-ary Grand Dyck words.
$C_{n,m}$

\[
\begin{cases}
(m + 1) \\ (k) \rightsquigarrow (1)(\overline{2}) \cdots (\overline{k})
\end{cases}
\]

The implementation of this succession rule does not give a CAT algorithm.

Remark (numerical evidences)

For an integer $p \geq 2$ and for $n = \frac{mp}{p-1}$ we have

\[
\frac{\text{total amount of computation}}{\text{number of generated words}} \leq 2
\]

and this rule yields a CAT algorithm for the set of $p$-ary Grand Dyck words.
\( C_{n,m} \)

\[
\begin{cases}
(m + 1) \\
(k) \mapsto (1)(\overline{2}) \cdots (\overline{k})
\end{cases}
\]

The implementation of this succession rule does not give a CAT algorithm.

**Remark (numerical evidences)**

For an integer \( p \geq 2 \) and for \( n = \frac{mp}{p-1} \) we have

\[
\frac{\text{total amount of computation}}{\text{number of generated words}} \leq 2
\]

and this rule yields a CAT algorithm for the set of \( p \)-ary Grand Dyck words.