# COOLER THAN COOL:
# COOL-LEX ORDER FOR GENERATING NEW COMBINATORIAL OBJECTS

PAUL LAPEY

## 1. Introduction

1.1. **Combinatorial Generation: Let's Look at all the Possibilities.** Combinatorial generation is defined as the exhaustive listing of combinatorial objects of various types. Frank Ruskey duly notes in his book *Combinatorial Generation* that the phrase "Let's look at all the possibilities" sums up the outlook of his book and the field as a whole [**?**]. Examining all possibilities fitting certain criteria is frequently necessary in fields ranging from mathematics to chemistry to operations research. Combinatorial generation as an area of study seeks to find an underlying combinatorial structure to these possibilities and utilize it to obtain an algorithm to efficiently enumerate an appropriate representation of them [Rus03].

A quintessential result of the combinatorial generation in practice is Frank Gray's reflected binary code, or Gray code. Gray codes give a "reflected" ordering of binary strings such that each successive string in the ordering differs from the previous string by exactly one bit. This is notably different from a lexicographic ordering of binary strings, in which a n-digit binary string can differ by up to n digits from its predecessor and will differ by approximately two (more precisely $\sum_{i=0}^{n} 2^i$, which is 1.9375 for 4 bit values and 1.996 for 8 bit values) bits on average[1]. The binary reflected Gray code, therefore, provides an ordering that requires as many bit switches as the more intuitive lexicographic order. Binary reflected Gray codes are widely used in electromechanical switches to reduce error and prevent spurious output associated with asynchronous bit switches. Crucially, Frank Gray's reflected binary code achieved a tangible benefit in error reduction through the use of an alternative method of enumerating binary strings. The technique of reflecting all or certain parts of a string to generate new strings has become one of the most widely used techniques in combinatorial generation.

1.2. **Cool-Lex Order.** More recently, cool-lex order has introduced the idea of rotating sublists to enumerate languages. Different versions of cool-lex order have been shown to enumerate several sets of combinatorial objects, including binary strings, fixed weight binary strings, Dyck words, and multiset permutations. Cool-lex orders often lead to algorithms that are faster and simpler than standard lexicographic order. For example, the "multicool" package in R uses a loopless cool-lex algorithm to efficiently enumerate multiset permutations. The package started using cool-lex order for multiset permutations in versoin 1.1 and as of version 1.12 has been downloaded nearly a million times [CWKB21].

[1]Consecutive pairs of binary digits in lexicographic order will differ in the bit at position i with probability $\frac{1}{2^i}$. Therefore, the average number of differing bits between two binary strings of length n is $\sum_{i=0}^{n} 2^i$, which converges to 2 as n grows large.

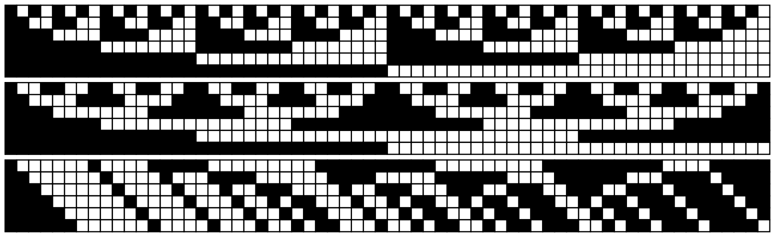


Figure 1. Lexicographic (top), binary reflected Gray code (middle), and cool-lex (bottom) enumerations of 6-bit binary strings. Individual strings are read vertically from top to bottom.

1.3. **Goals of this Thesis.** Cool-lex has been shown to provide a minimal-change cyclic ordering for the sets of fixed-weight binary strings, multiset permutations, binary and k-ary Dyck words, and other languages [Wil09b]. A common thread in the cool-lex algorithms for combinatorial generation is their focus on the *first increase* of string, or the longest prefix of a string such that each successive symbol in the prefix is less than or equal to the previous symbol in the string.

This thesis will examine the use of cool-lex orders to enumerate other languages. Among these are Lukasiewicz, Motzkin, and Schröder paths, which are lattice paths that share similarities with Dyck paths. Shift Gray codes for enumerating these languages have been developed and are given in 1.3.

Dyck, Motzkin, Schröder, and Lukasiewicz paths all share bijections with various combinatorial objects. For example, Dyck paths of length $2n$ share a bijection with binary trees with $n$ nodes. Ruskey and Williams found that the cool-lex successor rule for enumerating Dyck words corresponded directly to a loopless successor rule for enumerating binary trees with a constant number of pointer changes [RW08]. This thesis will examine the efficiency of using cool-lex order to enumerate other sets of combinatorial objects in bijective correspondence with these languages.

## 2. Background

In this background section, we will discuss the sets of combinatorial objects the thesis examines and introduce the reader to existing cool-lex algorithms.

2.1. **Dyck Words.** The language of binary Dyck words is the set of sequences of binary digits that satisfy the following conditions: The sequence has an equal number of ones and zeroes and there is no prefix of the sequence in which the number of zeroes exceeds the number of ones. The Dyck language can equivalently be thought of as the set of balanced parentheses, with ones representing open parentheses and zeroes representing closing parentheses. In addition to balanced parentheses, Dyck words of length $2n$ are also in bijective correspondence with extended binary trees with $n$ internal nodes. Given an extended binary tree $B$ with n internal nodes, a Dyck word can be obtained by traversing B in preorder and recording each internal node as a 1 and each leaf with a 0, ignoring the final leaf of the tree.

2.2. **Generalizations of Dyck words: Motzkin, Schröder, and Łukasiewicz paths.** Motzkin, Schröder, and Łukasiewicz paths provide generalizations of Dyck words.

In addition to representing balanced parentheses, Dyck paths can be thought of as paths on a cartesian plane. Dyck paths are paths from $(0, 0)$ to $(2n, 0)$ that use $2n$ steps of either $(1, 1)$ (northeast) or $(1, -1)$ (southeast) and never cross below the x axis. In the binary string representation of Dyck words, ones correspond to $(1, 1)$ steps and zerores correspond to $(1, -1)$ steps.

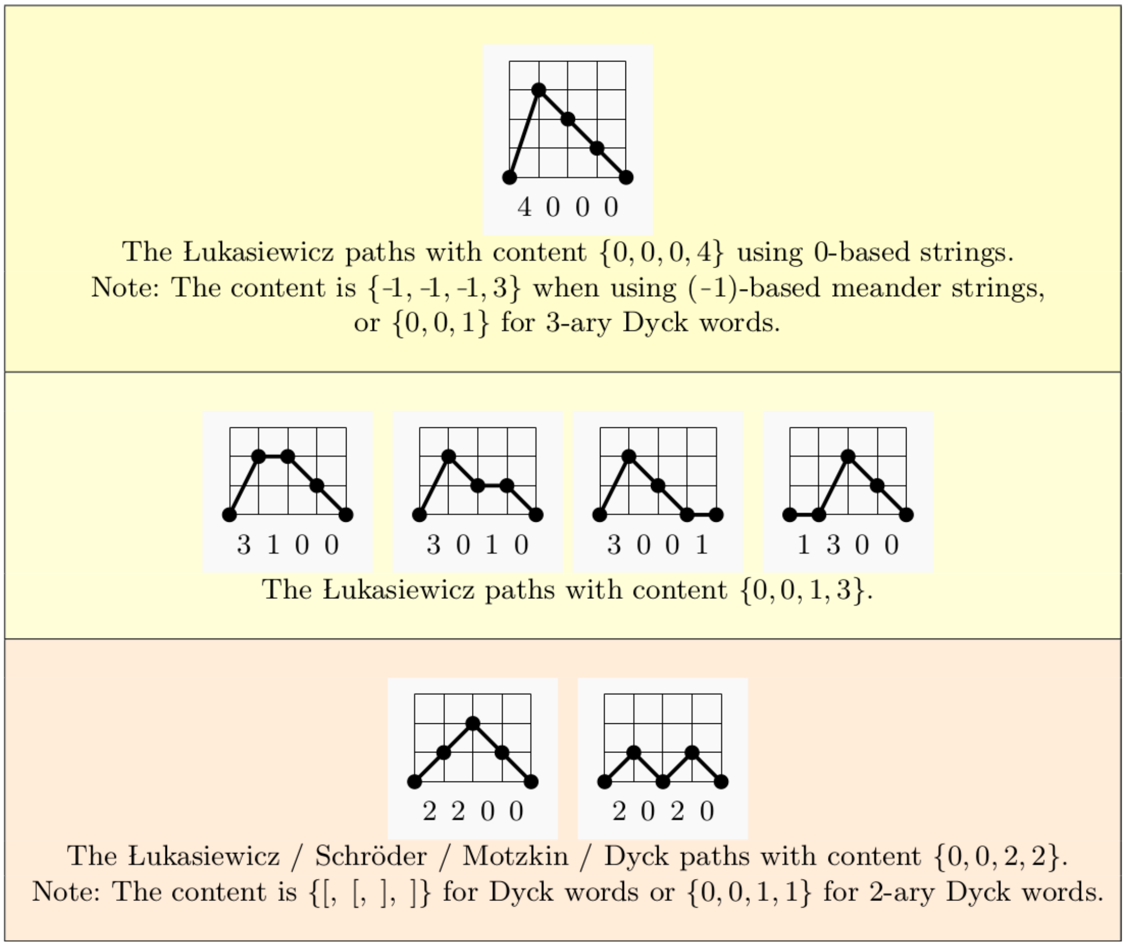
Motzkin paths allow for (1,0) horizontal steps in addition to (1,1) and (1,-1) steps. Schröder paths are identical to Motzkin paths except they allow for $(2, 0)$ horizontal steps instead of $(1, 0)$. Łukasiewicz paths allow (1,-1) steps, (1,0) steps and any (1,k) step where k is a positive integer. All three languages retain the requirement that the path start at the origin, end on the x axis, and never step below the x axis.

These paths can be encoded in a number of different ways. In a *-1-based encoding*, each $(1, i)$ step is encoded as i, and every prefix must have a nonnegative sum. In a *0-based encoding*, each $(1, i)$ step is encoded as $i + 1$, and the sum of every prefix must be as large as its length. We primarily use the 0-based encoding. See Fig. 2 for examples of these paths using the 0-based encoding.

We refer to Motzkin, Schröder, and Lukasiewicz paths ending at $(n, 0)$ as paths of *order n*. This contrasts slightly with the classification of Dyck words of order n, which terminate at $(2n, 0)$

In the context of fixed-content generation, Motzkin and Schröder paths are identical: Both will have northeast steps encoded as twos, horizontal steps encoded as ones, and southeast steps encoded as zeroes. However, their graphical representations Notably, Łukasiewicz are a generalization of Motzkin and Schröder paths, as any Motzkin or Schröder path is also a Lukasiewicz path.

The number of Dyck words with n zeroes and n ones are counted by the nth Catalan number. Similarly, the number of Motzkin and Schröder paths of order n are counted by the nth Motzkin and big Schröder number respectively. The number of Lukasiewicz paths of order n are counted by the n Motzkin, Schröder, and Lukasiewicz paths bear a number of interesting bijective correspondences with other combinatorial objects. Richard Stanely's *Catalan Objects* outlines hundreds of interesting examples.



The Łukasiewicz paths with content $\{0,0,0,4\}$ using 0-based strings.
Note: The content is $\{$-1, -1, -1, 3$\}$ when using (-1)-based meander strings, or $\{0,0,1\}$ for 3-ary Dyck words.

The Łukasiewicz paths with content $\{0,0,1,3\}$.

The Łukasiewicz / Schröder / Motzkin / Dyck paths with content $\{0,0,2,2\}$.
Note: The content is $\{$[, [, ], ]$\}$ for Dyck words or $\{0,0,1,1\}$ for 2-ary Dyck words.

Figure 2

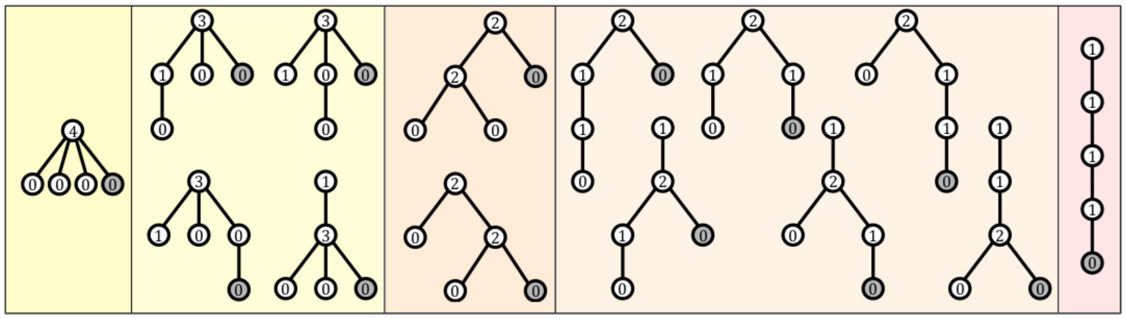


Figure 3. The $\mathcal{C}_4$=14 Lukasiewicz paths of order $n = 4$ are in bijective correspondence with the 14 rooted ordered trees with $n + 1 = 5$ nodes. Given a tree, the corresponding word is obtained by recording the number of children of each node in preorder traversal; the zero from the rightmost leaf is omitted. For example, the two trees in the middle section correspond to 2200 (top) and 2020 (bottom) respectively.

Lukasiewicz paths of order n bear a particularly nice correspondence to rooted ordered trees with $n+1$ nodes. See Fig. 3 for an illustration of this.

2.2.1. *Combinations: Fixed-Weight Binary Strings.* Generating all binary strings with $s$ zeroes and $t$ ones is often referred to as combinations, since each string can be used to represent a choice of $t$ elements from a set of size of $s+t$. The cool-lex successor rule for generating all fixed-weight binary strings was given by Aaron Williams in his Ph. D thesis and is as follows[Wil09b]:

Let $S$ be a binary string of length $n$.
Let y be the position of the leftmost zero in S and x be the position of the leftmost 1 in S such that $x \geq y$. Additionally, note that $S_1 ... S_{x-1}$ is the non-increasing prefix of S.

Let $\mathsf{leftshift}(S, x)$ be a function that rotates the first i bits of a string S left circularly by one.

More formally, $\mathsf{leftshift}(S, x) = S_2, S_3, ..., S_i, S_1, S_{i-1}, S_{i+1}, S_{i+2}, ..., S_{2n}$

$$\overleftarrow{\text{cool}}(S) = \begin{cases} \mathsf{leftshift}(S, x) & \text{if } S_{x+1} = 1 \\ \mathsf{leftshift}(S, x+1) & \textit{otherwise} \end{cases}$$

Note that $S_1 ... S_{x-1}$ must be exactly $1^{y-1}0^{x-y}$, where exponentiation denotes repeated symbols. Because of this, the two left-shift operations can be replaced with can be replaced with either one or two symbol transpositions.
Let $\mathsf{transpose}(S, i, j)$ with $1 \leq i \leq j \leq n$ be a function that swaps $S_i$ an $S_j$. More formally, $\mathsf{transpose}(S, i, j) = S_1, S_2, \ldots, S_{i-1}, S_j S_{i+1} \ldots S_{j-1} S_i S_{j+1} \ldots S_n$ The left-shift rule can be re-stated as follows:

$$\overleftarrow{\text{cool}}(S) = \begin{cases} \mathsf{transpose}(S, y, x) & \text{if } S_{x+1} = 1 \\ \mathsf{transpose}(\mathsf{transpose}(S, y, x), 1, x+1) & \textit{otherwise} \end{cases}$$

2.2.2. *Cool Lex Order on Dyck Paths and Binary Trees.* Ruskey and Williams found the following successor rule for enumerating binary Dyck words, dubbed "CoolCat" due to its cool-lex order and (cat)alan numbers [RW08]: We will use $\mathbf{B}_n$ to denote binary Dyck words with $n$ ones and $n$ zeroes. Note that the length of any string in $\mathbf{B}_n$ is thus $2n$.
Let $S \in \mathbf{B}_n$
Let the $i$th prefix shift of S, denoted by $\mathsf{preshift}(S, i)$, be a function that rotates the second through ith symbols of S one to the right circularly. More formally,
$\mathsf{preshift}(S, i) = S_1, S_i, S_2, ..., S_{i-1}, S_{i+1}, S_{i+2}, ..., S_{2n}$
Let $k$ be the index of the 1 in the leftmost 01 substring in S if it exists. Note that if $S$ has no 01 substring, then $S = 1^n 0^n$. The successor rule for $S$ is as follows:

$$\overleftarrow{\text{coolCat}}(S) = \begin{cases} \mathsf{preshift}(S, 2n) & \text{if } S \text{ has no 01 substring} \\ \mathsf{preshift}(S, k+1) & \text{if } \mathsf{preshift}(S, k+1) \in \mathbf{B}_n \\ \mathsf{preshift}(S, k) & \textit{otherwise} \end{cases}$$

Ruskey and Williams's algorithm can also enumerate a broader set of strings: The algorithm enumerates any set $\mathbf{B}_{s,t}$ where for any $S \in \mathbf{B}_{s,t}$ satisfies the constraint that each prefix of S has as many ones as zeroes. This is slightly broader than the language of Dyck words, as it does not have the requirement that a string have an equal number of ones and zeroes. We will focus on $\mathbf{B}_n$ languages due to their correspondce with Dyck words.

Evaluating whether $\mathsf{preshift}(S, k+1) \in B$ can be determined by looking $S_{k+1}$ and the sum of the first k symbols of S:

The above algorithm can be Let $S' = \mathsf{preshift}(S, k+1)$

Note that we know $S \in \mathbf{B}_n$.

Since preshift only rotates symbols, $S'$ will automaticallly satisfy the requirement that strings in $\mathbf{B}_n$ must have an equal number of zeroes and ones since S satisfied that requirement. Thus, $S' \in \mathbf{B}_n$ will be determined by whether or not all prefixes of $S'$ have at least as many ones as zeroes.

If $S_{k+1}$ is a 1, then every prefix i of $S'$ will have at least as many ones as the corresponding ith prefix of S. Thus, $S'$ must be $\in \mathbf{B}_n$, as rotating a 1 to earlier in the string will never invalidate the requirement that every prefix of the string has at least as many ones as zeroes.

Note that the kth prefix of S must be of the form $1^a 0^b 1$, as otherwise there would be an earlier 01 prefix. Fruthermore, $a \geq b$ as otherwise the bth prefix of S would have more zeroes than ones and S would not be a valid Dyck word.

If $S_{k+1}$ is a 0, then $S' \notin \mathbf{B}_n$ if and only if rotating a 0 to index 2 creates a prefix of S with more zeroes than ones. This will only happen if the k-1th prefix is exactly $1^{\frac{k-1}{2}} 0^{\frac{k-1}{2}}$

$$\overleftarrow{\text{coolCat}}(S) = \begin{cases} \mathsf{preshift}(S, 2n) & \text{if } S \text{ has no 01 substring} \\ \mathsf{preshift}(S, k+1) & S_{k+1} = 1 \text{ or } S \text{ starts with exactly } \frac{k-1}{2} \text{ ones} \\ \mathsf{preshift}(S, k) & \textit{otherwise} \end{cases}$$

Since $k$ is the index of the first 01 substring in S, $\sum_{i=1}^{k} S_i$ is actually just the number of consecutive ones to start S, which simplifies the evaluation of this conditional even further.

Ruskey and Williams provided a pseudocode implementation of CoolCat that utilized this fact to enumerate any $\mathbf{B}_{s,t}$ using at most 2 conditionals per successor [RW08].

Due to its simplicity and efficienty, Don Knuth included the cool-lex algorithm for Dyck words in his 4th volume of *The Art of Computer Programming* and also provided an implementation of it for his theoretical MMIX processor architecture [Knu15].

2.2.3. *Multiset Permutations.* Cool-lex order has also been shown to enumerate multiset permutations via prefix shifts. The rule given by Williams is as follows [Wil09a]:

Let S be a multiset of length n.
Let i be the maximum value such that $S_{j-1} \geq s_j$ for all $2 \leq j \leq i$. In other words, i is the length of the non-increasing prefix of S.
Let $\sigma_j(S)$ be a function that shifts the ith value of S into the first position, or equivalently rotates the first i elements of S right circularly. More formally,
$\sigma_j(S) = S_j, S_1, S_1, \ldots, S_{j-1}, S_j + 1, \ldots, S_n$
Then

$$\text{nextPerm}(S) = \begin{cases} \sigma_{i+1}(S) & \text{if } i \leq n-2 \text{ and } s_{i+2} > s_i \\ \sigma_{i+2}(S) & \text{if } i \leq n-2 \text{ and } s_{i+2} \leq s_i \\ \sigma_n(S) & \textit{otherwise} \end{cases}$$

See Fig. 4 for an example comparison of cool-lex and lexicographic order for two multisets.

This successor rule has the nice property of ensuring that length of the successor's non-increasing prefix is easy to find.

In particular, if $S_{i+2}$ is shifted, then the length of the non-increasing prefix is either 1 if $S_{i+2} \leq S_1$ or $i+1$ otherwise.

Similarly, if $S_{i+1}$ is shifted, then the length of the non-increasing prefix is either 1 if $S_{i+1} \leq S_1$ or $i+1$ otherwise.

This allows for a loopless implementation of the successor rule, as scanning the string to find the length of the non-increasing prefix is not required. Due to the simplicity and efficiency of this rule, it is used in the "multicool" package in R, which is used for generating multiset permutations, Bell numbers, and other combinatorial objects [CWKB21]. Further information on the package is available here: https://www.rdocumentation.org/packages/multicool/versions/0.1-12

## 3. New Results

This thesis provides successor rules and implementations for enumerating the following languages: Ordered trees with a fixed number of nodes, Lukasiewicz words with fixed content, and Motzkin/Schroder words with fixed content. The algorithm for ordered trees is loopless and The algorithm for enumerating Lukasiewicz paths also provides a generalization of the cool-lex successor rule for multiset permutations, given in section 2.

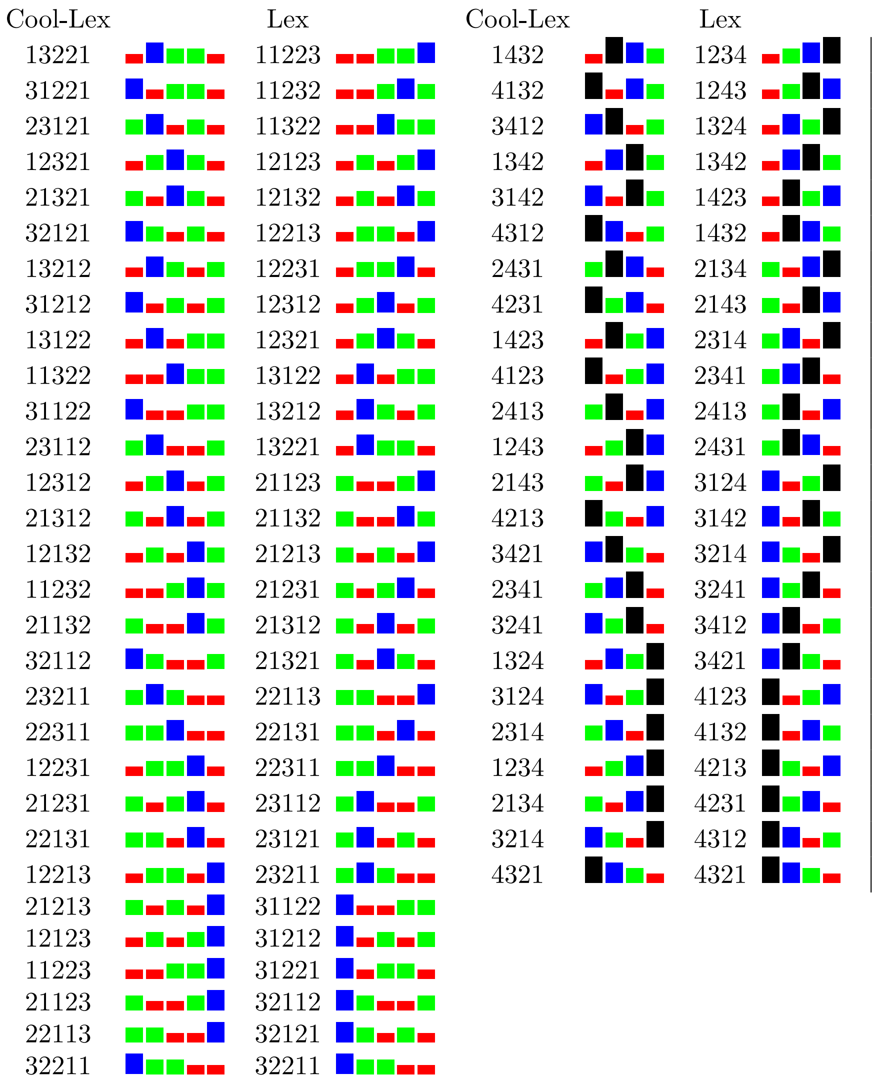
### 3.1. **Ordered Tree Successor Rule.**

### 3.2. **Lukasiewicz Path Successor Rule.**

The successor rule for Lukasiewicz paths is as follows:

Let $S$ be a multiset whose sum is equal to its length. Let $\mathcal{L}(S)$ denote the set of valid Lukasiewicz words with content equal to S. Let $\alpha \in \mathcal{L}(S)$.

Let $m$ be the maximum value such that $\alpha_{i-1} \geq \alpha_i$ for all $2 \leq i \leq m$. In other words, let m be the length of the non-increasing prefix of $\alpha$.



| Cool-Lex | Lex | Cool-Lex | Lex |
|---|---|---|---|
| 13221 | 11223 | 1432 | 1234 |
| 31221 | 11232 | 4132 | 1243 |
| 23121 | 11322 | 3412 | 1324 |
| 12321 | 12123 | 1342 | 1342 |
| 21321 | 12132 | 3142 | 1423 |
| 32121 | 12213 | 4312 | 1432 |
| 13212 | 12231 | 2431 | 2134 |
| 31212 | 12312 | 4231 | 2143 |
| 13122 | 12321 | 1423 | 2314 |
| 11322 | 13122 | 4123 | 2341 |
| 31122 | 13212 | 2413 | 2413 |
| 23112 | 13221 | 1243 | 2431 |
| 12312 | 21123 | 2143 | 3124 |
| 21312 | 21132 | 4213 | 3142 |
| 12132 | 21213 | 3421 | 3214 |
| 11232 | 21231 | 2341 | 3241 |
| 21132 | 21312 | 3241 | 3412 |
| 32112 | 21321 | 1324 | 3421 |
| 23211 | 22113 | 3124 | 4123 |
| 22311 | 22131 | 2314 | 4132 |
| 12231 | 22311 | 1234 | 4213 |
| 21231 | 23112 | 2134 | 4231 |
| 22131 | 23121 | 3214 | 4312 |
| 12213 | 23211 | 4321 | 4321 |
| 21213 | 31122 | | |
| 12123 | 31212 | | |
| 11223 | 31221 | | |
| 21123 | 32112 | | |
| 22113 | 32121 | | |
| 32211 | 32211 | | |

FIGURE 4. Illustration comparing cool-lex and lexicographic order for permutations of the multisets with content {1,1,2,2,3} and {1,2,3,4}

$$\overleftarrow{\text{luka}}(\alpha) = \begin{cases} \mathsf{leftshift}(n,2) & \text{if } m = n \\ \mathsf{leftshift}(m+1,1) & \text{if } m = n-1 \text{ or } \alpha_m < \alpha_{m+2} \text{ or} \\ & (\alpha_{m+2} = 0 \text{ and } \sum \rho = m) \\ \mathsf{leftshift}(m+2,1) & \text{if } \alpha_{m+2} \neq 0 \\ \mathsf{leftshift}(m+2,2) & \text{otherwise} \end{cases}$$

In addition to generating Lukasiewicz words, this successor rule also

3.3. **Loopless Motzkin Generation.** Since Lukasiewicz words are a generalization of Motzkin words, the same algorithm can be used to generate Motzkin words by restricting the content set S to be strictly zeroes, ones, and twos. However, the additional restrictions on Motzkin words allow for a simpler implementation of the rule. Pseudocode for loopless generation of Motzkin words is given below in Fig. 5. Further investigation may uncover a similar loopless implementation for the more general Lukasiewicz successor rule.

## REFERENCES

[CWKB21] James Curran, Aaron Williams, Jerome Kelleher, and Dave Barber. Package ‘multicool’, Jun 2021.
[Knu15] Donald Ervin Knuth. The art of computer programming: Combinatorial algorithms, vol. 4, 2015.

**Algorithm 1** Motzkin

```
function coolMotzkin($s, t$)
$n \leftarrow 2 * s + t$
$b \leftarrow 2^1 0^1 2^{s-1} 1^t 0^{s-1}$
$x \leftarrow 3$
$y \leftarrow 2$
$z \leftarrow 2$
visit($b$)
while $x <= n$ do
    $q \leftarrow b_{x-1}$
    $r \leftarrow b_x$
    $b_x \leftarrow b_{x-1}$
    $b_y \leftarrow b_{y-1}$
    $b_z \leftarrow b_{z-1}$
    $b_1 \leftarrow r$
    $x \leftarrow x + 1$
    $y \leftarrow y + 1$
    $z \leftarrow y + 1$
    if $b_x = 0$ then
        if $z - 2 > x - y$ then
            $b_1 = 2$
            $b_2 = 0$
            $b_x = r$
            $x \leftarrow 3$
            $y \leftarrow 2$
            $z \leftarrow 2$
        else
            $x \leftarrow x + 1$
    else if $q \geq b[x]$ then
        $b_x \leftarrow 2$
        $b_{x-1} \leftarrow 1$
        $b_1 \leftarrow 1$
        $z \leftarrow 1$
    visit($b$)
```

FIGURE 5. Pseudocode algorithm for loopless enumeration of Motzkin words

[Rus03] Frank Ruskey. Combinatorial generation. *Preliminary working draft. University of Victoria, Victoria, BC, Canada*, 11:20, 2003.

[RW08] Frank Ruskey and Aaron Williams. Generating balanced parentheses and binary trees by prefix shifts. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77*, pages 107–115, 2008.

[Wil09a] Aaron Williams. Loopless generation of multiset permutations by prefix shifts. In *SODA 2009, Symposium on Discrete Algorithms*, 2009.

[Wil09b] Aaron Michael Williams. *Shift gray codes.* PhD thesis, 2009.