Ranking and Unranking of a Generalized Dyck Language and the Application to the Generation of Random Trees

Jens Liebehenschel
Johann Wolfgang Goethe-Universität, Frankfurt am Main
Fachbereich Informatik
D-60054 Frankfurt am Main, Germany
e-mail: jens@sads.informatik.uni-frankfurt.de

Abstract

Given two disjoint alphabets $T_{[}$ and $T_{]}$ and a relation $\mathcal{R} \subseteq T_{[} \times T_{]}$, the generalized Dyck language $D^{\mathcal{R}}$ over $T_{[} \cup T_{]}$ consists of all words $w \in (T_{[} \cup T_{]})^{\star}$ which are equivalent to the empty word ε under the congruence δ defined by $x\,y \equiv \varepsilon \bmod \delta$ for all $(x, y) \in \mathcal{R}$.
If the Dyck words are arranged according to the lexicographical order, then ranking means to determine the rank, i. e. the position, of a Dyck word. Unranking creates the Dyck word from its position.
We present a ranking and an unranking algorithm for the generalized Dyck language. Given a Dyck word, the ranking algorithm reads a prefix as short as possible to compute the rank. Thus, this algorithm is optimal with respect to that mesure. The unranking algorithm creates the Dyck word symbol by symbol from left to right.
Further, we analyze the ranking algorithm. For the computation of the rank of a Dyck word for arbitrary $\mathcal{R}$, we compute the s-th moments of the random variable describing the length of the prefix to be read. No computation is necessary for the analysis of the unranking algorithm, because the whole Dyck word has to be created.
The generalized Dyck language can be used for the coding of trees. Not only the shape of the tree is coded but also its labels of the nodes and/or edges. The algorithms discussed here can be used for the ranking and unranking of different types of trees. With a random number generator and the unranking algorithm we are able to generate random trees with diverse properties. Some classes of labelled trees will be discussed.

Keywords: Dyck language, ranking, unranking, random trees, lexicographical order, one-to-one correspondence, average-case analysis.

1 Introduction and Basic Definitions

In this paper we examine the generalized Dyck language that was introduced in [Ke96b]. In comparison with the normal Dyck language, the generalized Dyck language has more than only one type of brackets. An ordering on the alphabet will be extended to an ordering on the Dyck words, called lexicographical order.

In [Li98], an algorithm for the generation of all Dyck words according to their lexicographical order was presented. Following a frequently used method [Ke98], the algorithm computes from one Dyck word its successor according to the lexicographical order by reading and changing a suffix of the Dyck word. The algorithm works without any knowledge about the Dyck words generated before.
In general, the length of the suffix to be read and changed is as short as possible, if the words are generated according to their lexicographical order. The algorithm presented in [Li98] is optimal with respect to the length of the suffix to be read and changed. As there is a connection between [Li98] and this paper, we consider the Dyck words to be ordered lexicographically in this paper, too.

There is a nice property of lexicographically ordered lists of words: Normally, one can find a simple algorithm to obtain a list of shorter words by deleting words and shortening the remaining words, such that the words in this list are also arranged according to the lexicographical order. This is another reason for using the lexicographical order here.

Given a set of ordered words, every word has a position that is called rank; the computation of the rank is called ranking. Unranking is the inverse operation, i. e. the generation of a word from its position.

First, we introduce the generalized Dyck language and the lexicographical order, then we give a short example. In the next section, the ranking and unranking algorithms are presented. In Section 3, the algorithms are analyzed. In Section 4, we give an application for the unranking algorithm of Dyck words: The generation of random trees for several classes of labelled trees.

Various representations of the shape of trees are known, for example the integer sequences: level-representation, leaf-representation and leaf-level-representation, e. g. [Ke98]. These representations do not allow to code the labels of the nodes and/or edges. Thus, the labels have to be coded in an additional sequence. Another possibility to code trees is the normal Dyck language, e. g. [Za80]. In Section 4, we will see that the generalized Dyck language is a handy coding for several classes of labelled trees, because the shape and the labels of the tree are coded in a Dyck word, not in two independent parts, one for coding the shape of the tree, the other for the representation of its labels. Nevertheless, the structure of the tree and its labels can be determined from its coding in a very simple way using a unique factorization of Dyck words.

Definition 1:
Let $t_1,\, t_2 \in \mathbb{N}$ and $T_[:= \{[_1, [_2, \ldots, [_{t_1}\}$ $\big($resp. $T_] := \{]_1,]_2, \ldots,]_{t_2}\}\big)$ be the *set of opening* (resp. *closing*) *brackets*. Let $|S|$ be the cardinality of the set S, so $\big|T_[\big| = t_1$ and $\big|T_]\big| = t_2$. With $T := T_[\cup T_]$ and a relation $\mathcal{R} \subseteq T_[\times T_]$ we obtain the *generalized Dyck language associated with* $\mathcal{R}$ by $D^{\mathcal{R}} := \{w \in T^\star \mid w \equiv \varepsilon \bmod \delta\}$, where ε denotes the *empty word* and δ is the congruence over T which is defined by $\big(\forall [_a,]_b \in T\big)\big([_a\,]_b \equiv \varepsilon \bmod \delta \iff ([_a,]_b) \in \mathcal{R}\big)$. The *set of all Dyck words of length* $2n$ is given by $D_{2n}^{\mathcal{R}} := D^{\mathcal{R}} \cap T^{2n}$. □

Remark 1:
Obviously, to each opening (resp. closing) bracket there is a unique *corresponding closing* (resp. *opening*) *bracket* in every Dyck word $w \in D^{\mathcal{R}}$.
The closing bracket corresponding to an opening bracket in a Dyck word $w \in D^{\mathcal{R}}$ can be found by searching for the first closing bracket behind the shortest word $w_2 \in D^{\mathcal{R}}$ (w_2 might be ε) on the right side of the opening bracket. If the opening (resp. its corresponding closing) bracket is $[_a$ (resp. $]_b$) and $([_a,]_b) \in \mathcal{R}$, we have $w = w_1\, [_a\, w_2\,]_b\, w_3$ with $w_1\, w_3, w_2 \in D^{\mathcal{R}}$.
The opening bracket corresponding to a closing bracket in a Dyck word can be found in an analogous way.

Definition 2:
Let $\mathsf{close}^{\mathcal{R}}([_a) := \{\,]_b \mid ([_a,]_b) \in \mathcal{R}\}$ be the *set of all closing brackets corresponding to a given opening bracket* according to the relation $\mathcal{R}$. □

Definition 3:
Let $<_{\text{lex}} \subseteq T \times T$ be an irreflexive linear ordering on T. The *lexicographical order* $\prec_{\text{lex}}$ *over* T^+ is defined as the extension of $<_{\text{lex}}$ to $\prec_{\text{lex}} \subseteq T^+ \times T^+$ by $x \prec_{\text{lex}} y :\Leftrightarrow$ $(\exists z \in T^+)\,(xz = y) \vee \big(\exists\,(u, \widetilde{x}, \widetilde{y}, v, \widetilde{v}) \in T^{\star 3} \times T^2\big)\,(x = u\,v\,\widetilde{x} \wedge y = u\,\widetilde{v}\,\widetilde{y} \wedge v <_{\text{lex}} \widetilde{v})$ [Ke98]. □

Note that in this paper we consider the lexicographical order on Dyck words of length $2n$ only.

We use the following ordering on T:

$$[_{|T_{[}|} <_{\text{lex}} \dots <_{\text{lex}} [_1 <_{\text{lex}}]_1 <_{\text{lex}} \dots <_{\text{lex}}]_{|T_{]}|} \ .$$

If all Dyck words of $D_{2n}^{\mathcal{R}}$ are arranged according to the lexicographical order, then *ranking* means to determine the *rank*, i. e. the position of a Dyck word $w \in D_{2n}^{\mathcal{R}}$. The rank of w is the number of lexicographically smaller Dyck words in the generalized Dyck language, i. e. $\operatorname{rank}(w) := \left|\{\widetilde{w} \mid \widetilde{w} \in D_{2n}^{\mathcal{R}} \ \wedge \ \widetilde{w} \prec_{\text{lex}} w\}\right|$. Thus, we immediately get $0 \leq \operatorname{rank}(w) < |D_{2n}^{\mathcal{R}}|$. Given the rank of $w \in D_{2n}^{\mathcal{R}}$, *unranking* computes w.

Let us demonstrate the above definitions by a simple example.

Example 1:
Let $T := \{[_1, [_2,]_1,]_2\}$, $\mathcal{R} := \{([_1,]_1), ([_1,]_2), ([_2,]_2)\}$ and $n := 2$. The relation $\mathcal{R}$ implies $\mathsf{close}^{\mathcal{R}}([_1) = \{\,]_1,]_2\}$ and $\mathsf{close}^{\mathcal{R}}([_2) = \{\,]_2\}$. We obtain the ordering on the alphabet $[_2 <_{\text{lex}} [_1 <_{\text{lex}}]_1 <_{\text{lex}}]_2$. In Figure 1 all Dyck words of this example are arranged lexicographically. Considering the rank of the Dyck words, we find $\operatorname{rank}([_2 [_2]_2]_2) = 0$, $\operatorname{rank}([_1 [_1]_2]_1) = 10$ and $\operatorname{rank}([_1]_2 [_1]_2) = 17 = |D_4^{\mathcal{R}}| - 1$, for example.

$[_2 [_2]_2]_2 \prec_{\text{lex}} [_2 [_1]_1]_2 \prec_{\text{lex}} [_2 [_1]_2]_2 \prec_{\text{lex}} [_2]_2 [_2]_2 \prec_{\text{lex}} [_2]_2 [_1]_1 \prec_{\text{lex}} [_2]_2 [_1]_2 \prec_{\text{lex}}$
$[_1 [_2]_2]_1 \prec_{\text{lex}} [_1 [_2]_2]_2 \prec_{\text{lex}} [_1 [_1]_1]_1 \prec_{\text{lex}} [_1 [_1]_1]_2 \prec_{\text{lex}} [_1 [_1]_2]_1 \prec_{\text{lex}} [_1 [_1]_2]_2 \prec_{\text{lex}}$
$[_1]_1 [_2]_2 \prec_{\text{lex}} [_1]_1 [_1]_1 \prec_{\text{lex}} [_1]_1 [_1]_2 \prec_{\text{lex}} [_1]_2 [_2]_2 \prec_{\text{lex}} [_1]_2 [_1]_1 \prec_{\text{lex}} [_1]_2 [_1]_2$

Figure 1: All Dyck words of Example 1 arranged according to the lexicographical order $\prec_{\text{lex}}$.

In the introduction we mentioned that it is often possible to create a list of lexicographically ordered words from a list of longer words by deleting words and performing some operation on the remaining words. We will now explain, how to modify the lexicographically ordered list of all Dyck words in $D_{2n}^{\mathcal{R}}$ to obtain all words in $D_{2n-2}^{\mathcal{R}}$ arranged according to the lexicographical order (for $n \geq 1$). In the first step, we delete all Dyck words that do not end with a pair of brackets. Then, we discard the rightmost pair of brackets from the remaining words. In the final step, all duplicates have to be cancelled. If any duplicates exist, then they are consecutive words in the list, thus it is simple to find duplicates. Applying these three steps to the list in Figure 1, we obtain all Dyck words in $D_2^{\mathcal{R}}$ with $\mathcal{R} := \{([_1,]_1), ([_1,]_2), ([_2,]_2)\}$: $[_2]_2 \prec_{\text{lex}} [_1]_1 \prec_{\text{lex}} [_1]_2$.

2 Ranking and Unranking Algorithm

In this section we will present a ranking algorithm for the generalized Dyck language $D_{2n}^{\mathcal{R}}$ as well as an unranking algorithm that creates the Dyck word by reversing the steps carried out by the ranking algorithm.

Since a Dyck word's rank is equal to the number of lexicographically smaller Dyck words, the ranking algorithm needs to determine that number. It follows the familiar practice and reads the Dyck word from left to right. As shown in the general approach to ranking and unranking [Li97], the algorithm has to go on reading until there is exactly one continuation to a word of the language. In that case the symbols not read so far are uniquely determined by the symbols read.

Given a Dyck word $w = w_0 \dots w_{2n-1}$, we compute the number of lexicographically smaller words by

$$\operatorname{rank}(w) := \sum_{i=0}^{2n-1} A_i \ ,$$

where $A_i := \left|\left\{\widetilde{w}_0 \dots \widetilde{w}_{2n-1} \in D_{2n}^{\mathcal{R}} \mid \widetilde{w}_0 \dots \widetilde{w}_{i-1} = w_0 \dots w_{i-1} \ \wedge \ \widetilde{w}_i <_{\text{lex}} w_i\right\}\right|$. Note that A_i is the number of Dyck words with the same prefix of length i as w and a smaller symbol at the right of that common prefix than in w. Furthermore, each A_i, $0 \leq i \leq 2n-1$, can be computed by reading the first i symbols of w, so that the algorithm can compute the rank by reading the Dyck word from left to right. If the shortest prefix with a unique continuation is read, the algorithm presented here does not read another symbol. So, it is optimal with respect to the number of symbols to be read from left to right.
Let j be the length of that shortest prefix with a unique continuation. We immediately find $0 \leq j \leq 2n$. Obviously, our algorithm is not able to compute A_i, $j \leq i \leq 2n-1$, because it does not read the suffix $w_j \dots w_{2n-1}$ of w. The computation of A_i, $j \leq i \leq 2n-1$, is not needed, because that suffix $w_j \dots w_{2n-1}$ is a unique continuation of the prefix read to a Dyck word in $D_{2n}^{\mathcal{R}}$, thus we get $A_i = 0$, $j \leq i \leq 2n-1$, by the definition of A_i.
Now, we have the fundamentals for the computation of the rank of a Dyck word w. The algorithm proceeds as follows: It reads w from left to right. For each position i, $0 \leq i \leq 2n-1$, is checked, whether exactly one continuation to a Dyck word of length $2n$ exists. That is the case, iff either all n opening brackets have been read and each of these corresponds to one closing bracket only or $(n-1)$ opening and $(n-1)$ closing brackets have been read and the relation contains only one pair of brackets. If such a unique continuation exists, then no more symbols have to be read, as we have seen before. If not, then the algorithm has to compute A_i, whereby it is necessary to distinguish between opening and closing brackets:

(i) For an opening bracket, the number of Dyck words with the same prefix as w and a smaller opening bracket at the current position than in w have to be added to the rank.

(ii) For a closing bracket, the number of Dyck words with the same prefix as w and an opening bracket at the current position have to be added to the rank as well as the number of Dyck words with a smaller closing bracket than in w – in consideration of the corresponding opening bracket.

Thus, it is necessary to know the opening bracket corresponding to each closing bracket. For that purpose the algorithms make use of a stack with the operations push and pop.
Furthermore, the algorithms need to compute the number of continuations of a Dyck word's prefix to a Dyck word of length $2n$. Therefore, let us recall the well-known one-to-one correspondence between Dyck words and paths in the lattice given in Figure 2.
Thereby, an *up-segment* $\nearrow$ (resp. *down-segment* $\searrow$) corresponds to an opening (resp. closing) bracket. Since we have a correspondence between opening and closing brackets (see Remark 1), there must be a correspondence between up-segments and down-segments of a path in the lattice. To each up-segment of any path there is a corresponding down-segment and vice versa: The down-segment (resp. up-segment) corresponding to an up-segment (resp. down-segment) is the next segment on the right (resp. left) side in the same row.
Thus, each Dyck word of length $2n$ corresponds to a path from $(0,0)$ to $(2n,0)$. The segments are labelled by the symbols of the alphabet; the labels of up-segments (resp. down-segments) are opening (resp. closing) brackets. Obviously, an up-segment and its corresponding down-segment must be labelled by some $[_a$ and $]_b$ with $\left([_a,]_b\right) \in \mathcal{R}$.

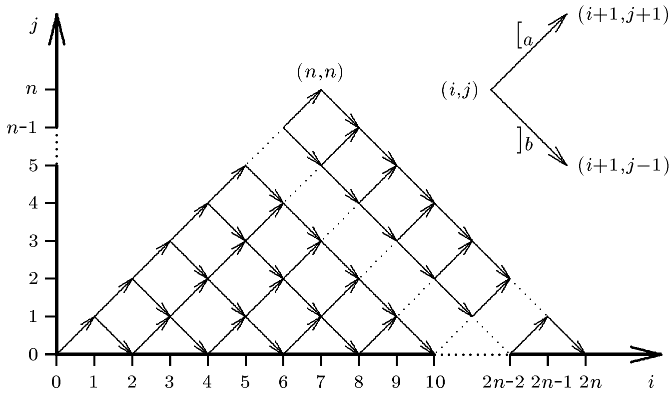

Figure 2: One-to-one correspondence between Dyck words of length $2n$ and the paths from $(0,0)$ to $(2n,0)$.

Note that the number of paths from $(0,0)$ to (i,j) in the lattice is given by the ballot number [CRS71]

$$\mathsf{p}(i,j) = \frac{j+1}{i+1}\binom{i+1}{\frac{1}{2}(i+j)+1} = \binom{i}{\frac{1}{2}(i-j)} - \binom{i}{\frac{1}{2}(i-j)-1}. \tag{1}$$

Now, the number of continuations can be computed by p and two other factors, one for the number of possibilities for the closing brackets not read so far and one for the pairs of brackets not read so far. The number of paths from (i,j) to $(2n,0)$ multiplied by those factors described above is given by:

$$\mathsf{q}(i,j) := \mathsf{p}(2n-i,j)\ \mathit{close}\ |\mathcal{R}|^{\frac{2n-i-j}{2}} ,$$

where *close* is a variable in the algorithms.
The variables used in the ranking algorithm will be explained now.

- *up* : Number of opening brackets (and up-segments).
- *down* : Number of closing brackets (and down-segments).
- $\mathit{close} := \prod_{j=0}^{k-1} \left|\mathsf{close}^{\mathcal{R}}(\widetilde{w}_j)\right|$, if $w_0 \dots w_i \equiv \widetilde{w}_0 \dots \widetilde{w}_{k-1} \bmod \delta \wedge w_0 \dots w_i$ is read in the algorithm $\wedge\ \widetilde{w}_0 \dots \widetilde{w}_{k-1} \in \left(T_{[}\right)^k$.
  Note that *close* is the product of the numbers of closing brackets in $\mathcal{R}$ corresponding to each opening bracket read without its corresponding closing bracket read so far, i. e. the number of possibilities to complete the prefix read to a Dyck word in $D^{\mathcal{R}}$ by appending closing brackets only. This variable is needed for the function q and will be recomputed for each symbol being read in both algorithms.
- $\mathit{corr\_cl\_br}[a-1] := \left|\mathsf{close}^{\mathcal{R}}([_a)\right|$, $1 \le a \le |T_{[}|$.
- $\mathit{corr\_cl\_br\_sm\_br}[a-1] := \sum_{l=1}^{a-1} \left|\mathsf{close}^{\mathcal{R}}([_l)\right|$, $1 \le a \le |T_{[}|$.
- $\mathit{sm\_corr\_cl\_br}[a-1][b-1] := \left|\{x \mid x \in \mathsf{close}^{\mathcal{R}}([_a) \wedge x <_{\text{lex}}]_b\}\right|$, $1 \le a \le |T_{[}|$, $1 \le b \le |T_{]}|$.

Note that the arrays *corr_cl_br* and *corr_cl_br_sm_br* and the matrix *sm_corr_cl_br* can be precomputed in $\mathcal{O}(|T_[||T_]|)$. This amount of time is constant with respect to the length of the words $2n$, because the variables depend on the relation $\mathcal{R}$ only. We code the brackets in integers in the ranking and unranking algorithms; the opening bracket of type a (resp. closing bracket of type b) is coded in the integer $-a$ (resp. b). Note that $w = w[0] \dots w[2n-1]$ is an array of integers in the algorithm. The formal ranking algorithm is given as follows.

```
function rank (w : dyck word)
  begin
    rank := 0
    up := 0
    down := 0
    close := 1
    i := 0
    while i < 2n do
      begin
        /* check if a unique continuation exists */
        if (up = n and close = 1) or
          (|R| = 1 and up = n - 1 and down = n - 1) then
          /* do not continue reading the Dyck word */
          i := 2n
        else
          begin
            if w[i] < 0 then
              begin
                /* w[i] is an opening bracket */
                rank := rank + corr_cl_br_sm_br[-w[i] - 1] ·
                                 q(i + 1, i + 1 - (2 · down))
                push(-w[i])
                close := close · corr_cl_br[-w[i] - 1]
                up := up + 1
              end
            else
              begin
                /* w[i] is a closing bracket */
                rank := rank + |R| · q(i + 1, i + 1 - (2 · down))
                j := pop()
                close := close / corr_cl_br[j-1]
                rank := rank + sm_corr_cl_br[j - 1][w[i] - 1] ·
                                 q(i + 1, i - 1 - (2 · down))
                down := down + 1
              end
            i := i + 1
          end
      end
  end;
```

Algorithm 1: Ranking algorithm for the generalized Dyck language $D_{2n}^{\mathcal{R}}$.

The unranking algorithm reverses the steps made by the ranking algorithm. It creates the word successively from left to right. The algorithm has to determine for each position, whether the current symbol is either an opening or a closing bracket.

For that purpose, we compute the number of Dyck words with the actual prefix and an opening bracket as the current symbol ($= |\mathcal{R}| \cdot k$, where $k = \mathsf{q}(i+1, i+1-(2 \cdot down))$). If the current value of $rank$ of the Dyck word in question is smaller than this number ($rank < |\mathcal{R}| \cdot k$), then the actual symbol is an opening bracket, otherwise it is a closing bracket.
Afterwards, the type of that bracket has to be computed. Further, a recomputation of the rank is necessary; this corresponds to the stepwise computation of the rank (A_i, $0 \leq i \leq 2n-1$) in the ranking algorithm.
To compute the type of the opening or closing bracket, two more variables are needed by the unranking algorithm:

- $op\_br[l-1]$: opening bracket of the l-th pair of brackets of the relation $\mathcal{R}$ according to the lexicographical order on the pairs of brackets, $1 \leq l \leq |\mathcal{R}|$.
- $cl\_br[a-1][l-1]$: l-th closing bracket that corresponds to $[_a$ in the relation $\mathcal{R}$ according to the ordering on the alphabet, $1 \leq a \leq |T_[|$, $1 \leq l \leq \left|\mathsf{close}^{\mathcal{R}}([_a)\right|$.

Again, the array op_br and the matrix cl_br can be precomputed in $\mathcal{O}(|T_[|\,|T_]|)$.
Now, let us have a look at the formal unranking algorithm.

function unrank ($rank$: **integer**)
  begin
    $down := 0$
    $close := 1$
    $i := 0$
    while $i < 2n$ **do**
      begin
        /⋆ $|\mathcal{R}| \cdot k$ = number of Dyck words with the actual prefix ⋆/
        /⋆ and an opening bracket as the current symbol ⋆/
        $k := \mathsf{q}(i+1, i+1-(2 \cdot down))$
        if $rank < |\mathcal{R}| \cdot k$ **then**
          begin
            /⋆ $unrank[i]$ is an opening bracket ⋆/
            $unrank[i] := op\_br[\lfloor \frac{rank}{k} \rfloor]$
            $rank := rank - corr\_cl\_br\_sm\_br[-unrank[i]-1] \cdot k$
            push$(-unrank[i])$
            $close := close \cdot corr\_cl\_br[-unrank[i]-1]$
          end
        else
          begin
            /⋆ $unrank[i]$ is a closing bracket ⋆/
            $rank := rank - |\mathcal{R}| \cdot k$
            $j :=$ pop()
            $close := \frac{close}{corr\_cl\_br[j-1]}$
            $k := \mathsf{q}(i+1, i-1-(2 \cdot down))$
            $unrank[i] := cl\_br[j-1][\lfloor \frac{rank}{k} \rfloor]$
            $rank := rank - sm\_corr\_cl\_br[j-1][unrank[i]-1] \cdot k$
            $down := down + 1$
          end
        $i := i + 1$
      end
  end;

Algorithm 2: Unranking algorithm for the generalized Dyck language $D_{2n}^{\mathcal{R}}$.

Let us have a look at the costs per symbol in the algorithms now. The critical part in both algorithms is the computation of the values $\mathsf{q}(i,j)$ for $0 \le i \le 2n$ and $0 \le j \le \min(i, 2n-i)$. In order to compute the costs for $\mathsf{q}(i,j)$, we need to compute the costs for the computation of the binomial coefficient in $\mathsf{p}(2n-i,j)$ and the costs for the computation of the power of $|\mathcal{R}|$.
There are two possibilities to obtain a constant amount of time for each symbol read or created, respectively:

(i) The binomial coefficients can be precomputed in $\mathcal{O}(n^2)$ by computing Pascal's triangle and the numbers $|\mathcal{R}|^i$, $0 \le i \le n$, can be precomputed in $\mathcal{O}(n)$ by successive multiplications of $|\mathcal{R}|$. Having precomputed these values once (in $\mathcal{O}(n^2)$), q can be computed in $\mathcal{O}(1)$. This is the method preferred for a frequent use of the ranking and unranking algorithms.

(ii) If one pass of the loop requires the computation of $\mathsf{q}(i,j)$, then the next pass requires the computation of one or two values of $\mathsf{q}(i+1,j+1)$, $\mathsf{q}(i+1,j-1)$ and $\mathsf{q}(i+1,j-3)$. With the formulae

$$\begin{aligned}
\mathsf{q}(i+1,j+1) &= \mathsf{q}(i,j)\,\frac{1}{2|\mathcal{R}|}\,\frac{j+2}{j+1}\,\frac{2n-i-j}{2n-i}\,,\\
\mathsf{q}(i+1,j-1) &= \mathsf{q}(i,j)\,\frac{1}{2}\,\frac{j}{j+1}\,\frac{2n-i+j+2}{2n-i}\,,\\
\mathsf{q}(i+1,j-3) &= \mathsf{q}(i,j)\,\frac{|\mathcal{R}|}{2}\,\frac{j-2}{j+1}\,\frac{2n-i+j}{2n-i}\,\frac{2n-i+j+2}{2n-i-j+2}\,,
\end{aligned}$$

we are able to compute q in $\mathcal{O}(1)$, too. This method is preferred in the case of few executions of the algorithms. The implementation of this method requires some modifications of the algorithms. These modifications are omitted here for the sake of a clear presentation.

Thus, the use of one of the strategies mentioned above leads to a constant amount of time per symbol read or created, respectively.

3 Analysis of the Algorithms

In this section we analyze the average number of symbols to be read by our ranking algorithm and the average number of symbols to be created by the unranking algorithm.
The analysis of the unranking algorithm does not require any computation, because the algorithm always has to create the whole Dyck word, so the number of symbols to create is equal to $2n$ for every $w \in D_{2n}^{\mathcal{R}}$. Thus, the mean value is equal to $2n$ and the variance is equal to 0.
Since the ranking algorithm reads the shortest prefix, that is long enough to compute the rank of the Dyck word, we can analyze the number of symbols to be read using the general approach to ranking and unranking [Li97]. Let $X_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$ be the *random variable describing the length of the shortest prefix to be read in order to rank a Dyck word.*
The mean value $\mu_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$ and the variance $\sigma^2_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$ of $X_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$ are given by:

$$\begin{aligned}
\mu_{\mathsf{pref}}(D_{2n}^{\mathcal{R}}) &= \mathbb{E}[X^1_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})] \text{ and}\\
\sigma^2_{\mathsf{pref}}(D_{2n}^{\mathcal{R}}) &= \mathbb{E}[X^2_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})] - \mathbb{E}[X^1_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})]^2\,.
\end{aligned}$$

We obtain the *s-th moments*, $s \geq 1$, of $X_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$ by [Li97]:

$$\begin{aligned} &\mathbb{E}[X_{\mathsf{pref}}^{s}(D_{2n}^{\mathcal{R}})] \\ :=\quad & (2n)^s - \left|D_{2n}^{\mathcal{R}}\right|^{-1} \left(\sum_{k=1}^{2n-1} \left((k+1)^s - k^s\right) \left|\mathsf{INIT}_k^1(D_{2n}^{\mathcal{R}})\right| + \delta_{|D_{2n}^{\mathcal{R}}|,1} \right) , \end{aligned} \tag{2}$$

where

$$\mathsf{INIT}_k^1(D_{2n}^{\mathcal{R}}) := \left\{ u \mid u\,v \in D_{2n}^{\mathcal{R}} \ \wedge\ u \in T^k \ \wedge\ \left|\left\{ u\,\widetilde{v} \in D_{2n}^{\mathcal{R}} \mid \widetilde{v} \in T^{2n-k} \right\}\right| = 1 \right\}$$

is the *set of prefixes of length k with a unique continuation to a word belonging to* $D_{2n}^{\mathcal{R}}$ and δ is *Kronecker's delta.*
Again, we need a criterion for a unique continuation of a Dyck word's prefix. Let us formalize the condition for a unique continuation that has been given in Section 2. Regarding the lattice in Figure 2, we see that a unique continuation to $(2n, 0)$ exists from all points of $\{(n+l, n-l) \mid 0 \leq l \leq n\}$, if the labels of all down-segments are known; in other words, all opening brackets are read and the types of the closing brackets not read so far can be determined with knowledge of the prefix read. Moreover, a unique continuation exists from $(2n-2, 0)$, if the labels of the last two segments are known, i. e. if only one pair of brackets is not read so far and $|\mathcal{R}| = 1$. We can compute the cardinality of $\mathsf{INIT}_k^1(D_{2n}^{\mathcal{R}})$, $1 \leq k < 2n$, now:

$$\begin{aligned} &\left|\mathsf{INIT}_k^1(D_{2n}^{\mathcal{R}})\right| \\ =\quad & \begin{cases} 0 & \text{if} \quad 1 \leq k \leq n-1 \\ \mathsf{p}(k, 2n-k)\ \left|T_{[}^{1}\right|^{2n-k}\ |\mathcal{R}|^{k-n} & \text{if} \quad n \leq k \leq 2n-3 \\ \mathsf{p}(2n-2, 2) + \mathsf{p}(2n-2, 0) & \text{if} \quad k = 2n-2 \wedge |\mathcal{R}| = 1 \\ \mathsf{p}(2n-2, 2)\ \left|T_{[}^{1}\right|^{2}\ |\mathcal{R}|^{n-2} & \text{if} \quad k = 2n-2 \wedge |\mathcal{R}| \geq 2 \\ \mathsf{p}(2n-1, 1)\ \left|T_{[}^{1}\right|\ |\mathcal{R}|^{n-1} & \text{if} \quad k = 2n-1 \end{cases} . \end{aligned} \tag{3}$$

Here, $T_{[}^{1} := \{x \mid x \in T_{[} \ \wedge\ |\mathsf{close}^{\mathcal{R}}(x)| = 1\}$ is the *set of opening brackets with only one corresponding closing bracket according to* $\mathcal{R}$ and p is given in (1).
Before computing the s-th moments of $X_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$, we prove the following technical lemma which gives us the asymptotic behaviour for two types of sums.
Lemma 1:
The asymptotics for both sums are valid for $n \to \infty$.

(i) The following asymptotic holds for $s \in \mathbb{N}_0$ and $\alpha > \frac{1}{2}$:

$$\begin{aligned} & \sum_{k=0}^{n-1} (n+k+1)^s\, \alpha^k \left[\binom{n+k}{n} - \binom{n+k}{n+1} \right] \\ =\quad & (4\alpha)^n\, n^{-\frac{3}{2}}\, \pi^{-\frac{1}{2}}\, (2n)^s\, \frac{4\alpha - 1}{(2\alpha-1)^2}\, \left(1 + \mathcal{O}(n^{-1})\right) . \end{aligned} \tag{4}$$

(ii) The following asymptotic holds for $\alpha > \frac{1}{2}$:

$$\sum_{k=0}^{n-1} (n-k)\, \alpha^k \left[\binom{n+k}{n} - \binom{n+k}{n+1} \right] \tag{5}$$

$$= \quad (4\alpha)^n\, n^{-\frac{3}{2}}\, \pi^{-\frac{1}{2}}\, \frac{8\alpha^2}{(2\alpha-1)^3}\, \left(1 + \mathcal{O}(n^{-1})\right) . \tag{6}$$

Proof:
The asymptotic for the first sum was computed in [Ke96a].

To compute the asymptotic equivalent for the second sum we first consider the following generating function for $c \in \mathbb{N}_0$

$$G(z,c,\alpha) := \sum_{n\geq 0} z^n \sum_{k=0}^{n} (n-k)\,\alpha^k \binom{n+k}{n+c} = \sum_{n\geq 0} z^n \sum_{k=0}^{n} k\,\alpha^{n-k} \binom{2n-k}{n+c} \; .$$

After rearranging the sums and shifting the index of summation, we get:

$$G(z,c,\alpha) = \sum_{\lambda\geq 0} \lambda\, z^\lambda \sum_{\mu\geq 0} (\alpha z)^{\mu+c} \binom{2\mu+\lambda+2c}{\mu} \; . \qquad (7)$$

With the identities $\sum_{\lambda\geq 0} \lambda\, x^\lambda = \frac{x}{(1-x)^2}$, e. g. [GKP94, p. 335], and

$$\sum_{l\geq 0} \binom{2l+\beta}{l} (\alpha z)^l = \frac{1}{\sqrt{1-4\alpha z}} \left(\frac{1-\sqrt{1-4\alpha z}}{2\alpha z}\right)^\beta , \; \beta \in \mathbb{N}_0 \; ,$$

e. g. [GKP94, p. 203], we can rewrite (7):

$$\begin{aligned} G(z,c,\alpha) &= \frac{1}{\sqrt{1-4\alpha z}} (\alpha z)^c \left(\frac{1-\sqrt{1-4\alpha z}}{2\alpha z}\right)^{2c} \sum_{\lambda\geq 0} \lambda \left(\frac{1-\sqrt{1-4\alpha z}}{2\alpha}\right)^\lambda \\ &= \frac{1}{\sqrt{1-4\alpha z}} \frac{1}{\alpha} \left(\frac{1}{\alpha z}\right)^c \left(\frac{1-\sqrt{1-4\alpha z}}{2}\right)^{2c+1} \frac{1}{\left(1-\frac{1-\sqrt{1-4\alpha z}}{2\alpha}\right)^2} \; . \qquad (8) \end{aligned}$$

Now, with (8) for $c=0$ and $c=1$ we find the generating function of the sum (5). Note that the term for $k=n$ in the sum with index of summation k is equal to 0. We obtain the following simple closed-form expression of the generating function $H(z,\alpha)$ of the sum (5) by a straight-forward computation.

$$H(z,\alpha) := G(z,0,\alpha) - G(z,1,\alpha) = \frac{2\left(1-2\alpha z-\sqrt{1-4\alpha z}\right)}{z\left(1-2\alpha-\sqrt{1-4\alpha z}\right)^2} \; .$$

Now, we are looking for the singularity of smallest modulus that is not equal to 0. We find possible singularities at $z=\frac{1}{4\alpha}$ and $z=1-\alpha$. Obviously, $z=1-\alpha$ is no singularity for $\alpha > \frac{1}{2}$, because

$$H(1-\alpha,\alpha) = \frac{2\left(1-2\alpha(1-\alpha)-\sqrt{1-4\alpha(1-\alpha)}\right)}{(1-\alpha)\left(1-2\alpha-\sqrt{1-4\alpha(1-\alpha)}\right)^2} = \frac{1-\alpha}{(2\alpha-1)^2} \; , \; \alpha > \frac{1}{2} \; .$$

Thus, the dominant singularity is at $z=\frac{1}{4\alpha}$. Now, applying Darboux's method, e. g. [Ke84], we get the asymptotic behaviour of the sum stated in the lemma. ■

Remark 2:
The following computation will show that it is not possible to deduce the asymptotical behaviour of (5) from (4).

$$\begin{aligned} &\sum_{k=0}^{n-1} (n-k)\,\alpha^k \left[\binom{n+k}{n} - \binom{n+k}{n+1}\right] \\ &= \sum_{k=0}^{n-1} [(2n+1)-(n+k+1)]\,\alpha^k \left[\binom{n+k}{n} - \binom{n+k}{n+1}\right] \\ &= \frac{(4\alpha)^n}{n^{\frac{3}{2}}\,\pi^{\frac{1}{2}}}\,\mathcal{O}(1) \; . \end{aligned}$$

With the asymptotics for the two sums we are able to analyze the random variable $X_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$.

Theorem 1:
Assuming that all words in $D_{2n}^{\mathcal{R}}$ are equally likely, the s-th moments, $s \geq 1$, of the random variable $X_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$ are given by:

$$\mathbb{E}[X_{\mathsf{pref}}^{s}(D_{2n}^{\mathcal{R}})] = (2n)^s + \mathcal{O}(n^{s-1}) \ .$$

Proof:
With the abbreviation $\alpha := \frac{|\mathcal{R}|}{|T_{[}^{1}|} \geq 1$ we get by (2) using (3) and (1) and by shifting the index of summation

$$\begin{aligned}
&\mathbb{E}[X_{\mathsf{pref}}^{s}(D_{2n}^{\mathcal{R}})] = \\
&\quad (2n)^s - \left|D_{2n}^{\mathcal{R}}\right|^{-1} \left|T_{[}^{1}\right|^{n} \times \left\{ \sum_{k=0}^{n-1} (n+k+1)^s \,\alpha^k \left[\binom{n+k}{n} - \binom{n+k}{n+1}\right] \right. \\
&\qquad\qquad \left. - \underbrace{\sum_{k=0}^{n-1} (n+k)^s \,\alpha^k \left[\binom{n+k}{n} - \binom{n+k}{n+1}\right]}_{=: f(n,\alpha)} \right\} .
\end{aligned}$$

The asymptotic behaviour of the first sum is given by (4). To gain an asymptotic equivalent for the second sum $f(n,\alpha)$, we first apply the addition formula for binomial coefficients $\binom{a}{b} = \binom{a-1}{b} + \binom{a-1}{b-1}$. We find

$$f(n,\alpha) = g(n,\alpha) + h(n,\alpha) \ , \tag{9}$$

where

$$\begin{aligned}
g(n,\alpha) &:= \sum_{k=0}^{n-1} (n+k)^s \,\alpha^k \left[\binom{n+k-1}{n} - \binom{n+k-1}{n+1}\right] , \\
h(n,\alpha) &:= \sum_{k=0}^{n-1} (n+k)^s \,\alpha^k \left[\binom{n+k-1}{n-1} - \binom{n+k-1}{n}\right] .
\end{aligned}$$

By shifting the index of summation and applying $\binom{2n-1}{n} - \binom{2n-1}{n+1} = \frac{1}{n+1}\binom{2n}{n}$, we obtain

$$g(n,\alpha) = \sum_{k=0}^{n-1} (n+k+1)^s \,\alpha^{k+1} \left[\binom{n+k}{n} - \binom{n+k}{n+1}\right] - (2n)^s \,\alpha^n \,\frac{1}{n+1}\binom{2n}{n} \ .$$

By (4) and the asymptotic behaviour of the *Catalan numbers* computed by Stirling's formula [GKP94]

$$\frac{1}{n+1}\binom{2n}{n} = 4^n \, n^{-\frac{3}{2}} \, \pi^{-\frac{1}{2}} \,(1 + \mathcal{O}(n^{-1})) \tag{10}$$

we immediately get:

$$g(n,\alpha) = (4\alpha)^n \, n^{-\frac{3}{2}} \, \pi^{-\frac{1}{2}} \,(2n)^s \, \frac{3\alpha - 1}{(2\alpha-1)^2} \,(1 + \mathcal{O}(n^{-1})) \ .$$

A similar computation yields

$$\begin{aligned}
&h(n+1,\alpha) \\
&\quad = \sum_{k=0}^{n-1} (n+k+1)^s \,\alpha^k \left[\binom{n+k}{n} - \binom{n+k}{n+1}\right] + (2n+1)^s \,\alpha^n \,\frac{1}{n+1}\binom{2n}{n} \\
&\quad = (4\alpha)^n \, n^{-\frac{3}{2}} \, \pi^{-\frac{1}{2}} \left[(2n)^s \frac{4\alpha-1}{(2\alpha-1)^2} + (2n+1)^s\right] (1 + \mathcal{O}(n^{-1})) \ .
\end{aligned}$$

Now, using the asymptotics for $g(n,\alpha)$ and $h(n,\alpha)$ in (9) and applying the formula for the number of Dyck words $|D_{2n}^{\mathcal{R}}| = \frac{1}{n+1}\binom{2n}{n}|\mathcal{R}|^n$, cf. [Li98], a simple computation leads to the equation $\mathbb{E}[X_{\mathsf{pref}}^s(D_{2n}^{\mathcal{R}})] = (2n)^s\,(1+\mathcal{O}(n^{-1}))$. ∎

With $\mathbb{E}[X_{\mathsf{pref}}^s(D_{2n}^{\mathcal{R}})]$ we are not able to compute $\mu_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$ and $\sigma_{\mathsf{pref}}^2(D_{2n}^{\mathcal{R}})$ with precision $\mathcal{O}(n^{-1})$. This will be done in Theorem 2 and Theorem 3.

Obviously, (3) implies that we must distinguish between the cases $|\mathcal{R}| = 1$ and $|\mathcal{R}| \geq 2$. The case $|\mathcal{R}| = 1$ has been investigated in [Li97]. There we find the following theorem.

Theorem 2:
Assuming that all words in $D_{2n}^{\mathcal{R}}$ are equally likely, the mean value and the variance of the random variable $X_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$ for $|\mathcal{R}| = 1$ and $n \geq 2$ are given by:

(i) $\mu_{\mathsf{pref}}(D_{2n}^{\mathcal{R}}) = 2n - \dfrac{13}{4} + \mathcal{O}(n^{-1})$,

(ii) $\sigma_{\mathsf{pref}}^2(D_{2n}^{\mathcal{R}}) = \dfrac{51}{16} + \mathcal{O}(n^{-1})$.

∎

For the case $|\mathcal{R}| \geq 2$ we get the mean value and the variance of the number of symbols to be read in the following theorem.

Theorem 3:
Let $\alpha := \frac{|\mathcal{R}|}{|T_{[}^{1}|} \geq 1$. Assuming that all words in $D_{2n}^{\mathcal{R}}$ are equally likely, the mean value and the variance of the random variable $X_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$ for $|\mathcal{R}| \geq 2$ are given by:

Case 1: $|T_{[}^{1}| = 0$, $n \geq 1$.

(i) $\mu_{\mathsf{pref}}(D_{2n}^{\mathcal{R}}) = 2n$,

(ii) $\sigma_{\mathsf{pref}}^2(D_{2n}^{\mathcal{R}}) = 0$.

Case 2: $|T_{[}^{1}| \geq 1$, $n \to \infty$.

(i) $\mu_{\mathsf{pref}}(D_{2n}^{\mathcal{R}}) = 2n - \dfrac{4\alpha - 1}{(2\alpha-1)^2} + \mathcal{O}(n^{-1})$,

(ii) $\sigma_{\mathsf{pref}}^2(D_{2n}^{\mathcal{R}}) = \dfrac{4\alpha^2\,(4\alpha-3)}{(2\alpha-1)^4} + \mathcal{O}(n^{-1})$.

Proof:
The proof of Case 1 is trivial. We insert (3) in (2) and obtain $\mathbb{E}[X_{\mathsf{pref}}^1(D_{2n}^{\mathcal{R}})] = 2n$ and $\mathbb{E}[X_{\mathsf{pref}}^2(D_{2n}^{\mathcal{R}})] = (2n)^2$.

For the mean value in Case 2 we get:

$$\begin{aligned}\mu_{\mathsf{pref}}(D_{2n}^{\mathcal{R}}) = \mathbb{E}[X_{\mathsf{pref}}^1(D_{2n}^{\mathcal{R}})] &= 2n - |D_{2n}^{\mathcal{R}}|^{-1}\sum_{k=n}^{2n-1}\mathsf{p}(k,2n-k)\;|T_{[}^{1}|^{2n-k}\;|\mathcal{R}|^{k-n}\\ &= 2n - \frac{|T_{[}^{1}|^n}{|D_{2n}^{\mathcal{R}}|}\sum_{k=0}^{n-1}\alpha^k\mathsf{p}(n+k,n-k)\ .\end{aligned}$$

With (1) and $|D_{2n}^{\mathcal{R}}| = \frac{1}{n+1}\binom{2n}{n}|\mathcal{R}|^n$, cf. [Li98], we obtain

$$\mu_{\mathsf{pref}}(D_{2n}^{\mathcal{R}}) = 2n - \frac{n+1}{\binom{2n}{n}}\alpha^{-n}\sum_{k=0}^{n-1}\alpha^k\left[\binom{n+k}{n} - \binom{n+k}{n+1}\right]\ .$$

By (4) and (10) we immediately get the asymptotics for $\mu_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})$.

For the variance, we obtain with (2), (3) and (1)

$$\begin{aligned}\sigma^2_{\mathsf{pref}}(D_{2n}^{\mathcal{R}}) &= \mathbb{E}[X^2_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})] - \mathbb{E}[X^1_{\mathsf{pref}}(D_{2n}^{\mathcal{R}})]^2 \\ &= \left((2n)^2 - \frac{\left|T_{[}^{1}\right|^n}{|D_{2n}^{\mathcal{R}}|}\sum_{k=0}^{n-1}(2n+2k+1)\,\alpha^k\left[\binom{n+k}{n}-\binom{n+k}{n+1}\right]\right) \\ &\quad - \left(2n - \frac{\left|T_{[}^{1}\right|^n}{|D_{2n}^{\mathcal{R}}|}\sum_{k=0}^{n-1}\alpha^k\left[\binom{n+k}{n}-\binom{n+k}{n+1}\right]\right)^2 .\end{aligned}$$

Simplifying this expression yields

$$\begin{aligned}\sigma^2_{\mathsf{pref}}(D_{2n}^{\mathcal{R}}) &= \frac{\left|T_{[}^{1}\right|^n}{|D_{2n}^{\mathcal{R}}|}\,2\sum_{k=0}^{n-1}(n-k)\,\alpha^k\left[\binom{n+k}{n}-\binom{n+k}{n+1}\right] \\ &\quad - \frac{\left|T_{[}^{1}\right|^n}{|D_{2n}^{\mathcal{R}}|}\sum_{k=0}^{n-1}\alpha^k\left[\binom{n+k}{n}-\binom{n+k}{n+1}\right] \\ &\quad - \left(\frac{\left|T_{[}^{1}\right|^n}{|D_{2n}^{\mathcal{R}}|}\sum_{k=0}^{n-1}\alpha^k\left[\binom{n+k}{n}-\binom{n+k}{n+1}\right]\right)^2 .\end{aligned}$$

By applying (4), (6) and (10) and performing simplifications, we directly get the variance stated in the theorem. ■

Remark 3:
We have analyzed the number of symbols to be read in the ranking algorithm and have pointed out the number of symbols to be created by the unranking algorithm. So far, we did not talk about the running-time of the algorithms. In the last section we have seen that the amount of time per symbol is constant in both algorithms, so the running-time is proportional to the number of symbols to be read or created, respectively.

Remark 4:
In [Li97] the ranking and unranking of the *Dyck language with t types of brackets* D^t [Ha78, p. 313] was analyzed. Following our definitions, $D^t = D^{\mathcal{R}}$ with the relation $\mathcal{R} = \{([_1,]_1),([_2,]_2),\dots,([_t,]_t)\}$. Further, $D_{2n}^t = D_{2n}^{\mathcal{R}}$.
The mean value and the variance of the random variable $X_{\mathsf{pref}}(D_{2n}^t)$ describing the number of symbols to be read for computing the rank for $t \geq 2 \wedge n \geq 1$ was found to be:

(i) $\mu_{\mathsf{pref}}(D_{2n}^t) = 2n - 3 + \mathcal{O}(n^{-1})$,

(ii) $\sigma^2_{\mathsf{pref}}(D_{2n}^t) = 4 + \mathcal{O}(n^{-1})$.

With $\mathcal{R} = \{([_1,]_1),\dots,([_t,]_t)\}$, $t \geq 2$, we find $|\mathcal{R}| = \left|T_{[}^{1}\right| = t$. This implies $\alpha = \frac{|\mathcal{R}|}{\left|T_{[}^{1}\right|} = 1$. Thus, we obtain by Theorem 3, Case 2, the results stated in [Li97].

4 Generation of Random Trees

An interesting application for the unranking algorithm will be presented now: The generation of random labelled trees.
Studying a class of combinatorial objects, it turns out to be an advantage to the researcher to have a procedure at hand, that gives an idea of the behaviour of the parameter in question. This procedure performs an exhaustive search for small objects and a random sampling for large objects [NW78, p. 3].

Another application for random labelled trees is in the area of testing computer programs, in particular the implementation of tree-based algorithms and data structures.
For a small size of the trees, one can generate all trees [Li98] and compute the parameter in question. Unfortunately, this is not possible for trees with a larger number of nodes, since the number of trees is exponential in the number of nodes. Although the parameter can be computed exactly for small trees, an conclusion to the asymptotical behaviour of the parameter (for large trees) cannot be made. So, it is necessary to find another method to get an idea of the parameter's behaviour. A practicable method is the generation of a sufficiently large set of random trees.
Two methods to generate random objects were presented in [Wi77]; these methods were also applied in [NW78].
One method is to generate a random number and compute the corresponding object (or its linear coding) with an unranking algorithm. In this paper, we generate a random tree by applying the unranking algorithm for the generalized Dyck language to a random number.
The other method constructs the object step by step. In each single step, the next symbol must be chosen from all possible symbols according to their probabilities depending on the actual position; these probabilities result from the number of continuations to an object of the desired size. This method requires the application of the random number generator for each symbol to be generated. It is used in [BLP99], for instance.
It is well-known that there are several one-to-one correspondences between the normal Dyck language and other combinatorial objects. For example, the normal Dyck language is a linear coding for the shape of an *ordered tree* (arbitrary degree of the nodes) or an *extended ordered binary tree* (two types of nodes: internal nodes with two sons and leaves), cf. [AS95, p. 32] and [Za80], but the labels of the nodes or of the edges cannot be coded in the normal Dyck language. In [FS96, p. 266] a coding of ordered trees with labelled nodes by another generalization of the normal Dyck language was mentioned. With the generalization discussed here, we can also code the labels of the edges. Additionally, connections between certain labels can be considered, for example connections between the labels of the root of a subtree and the edge to its father (in an ordered tree) and connections between the labels of the brothers and/or the edges to their father (in an extended ordered binary tree). So, several classes of labelled trees can be coded by the generalized Dyck language. We point out three of them:

(a) Ordered trees with labelled nodes and labelled edges:
In this class the relation $\mathcal{R}$ implies a connection between the labels of the nodes and the labels of the edges. The label of the node (resp. the label of the edge to its father) is coded in the type of the opening (resp. closing) bracket.

(b) Extended ordered binary trees with labelled nodes:
Considering an internal node in an extended ordered binary tree, the label of the left (resp. right) son is coded in the type of the opening (resp. closing) bracket. So, a connection between the labels of the brothers is established.

(c) Extended ordered binary trees with labelled nodes and labelled edges:
Here, the labels of the left (resp. right) son and the edge to its father are coded in the type of the opening (resp. closing) bracket. In that case we have a relation between the labels of the brothers and the labels of the edges to their father.

Remark 5:
Note that in each case the label of the root of the tree is not coded in the Dyck word. It can be coded – if needed – in a further symbol to the left or right of the Dyck word.

Remark 6:
Consider a class of trees without any connection between the labels of the nodes and/or edges, coded in $T_{[}$ and $T_{]}$; then each combination of opening and closing brackets is possible, i. e. the relation is given by: $\mathcal{R} := T_{[} \times T_{]}$.

The one-to-one correspondences between the trees and the Dyck words are defined recursively. All trees in Figure 3 correspond to the Dyck word $[_a\, X\,]_b\, Y$, where $[_a$ and $]_b$ are corresponding brackets (following Remark 1) and $X, Y \in D^{\mathcal{R}}$. Note that $[_a\, X\,]_b\, Y$ is a unique factorization of a Dyck word in $D^{\mathcal{R}} \setminus \{\varepsilon\}$.

The left picture shows an ordered tree (see (a)), whereas the other pictures represent extended ordered binary trees. In the middle, we find a tree with labelled nodes, corresponding to (b), and in the right picture, a tree with labels of the nodes and the edges is given (see (c)). In the latter case, the label of the left son (a_2) and the label of the edge to its father (a_1) must be coded in the type of the opening bracket (a). The same holds for b_2, b_1 and b.

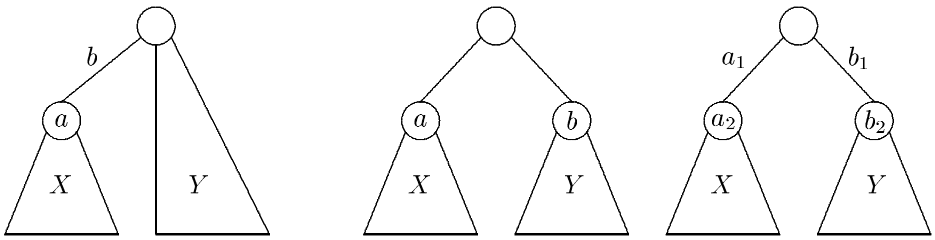

Figure 3: Labelled trees, corresponding to the Dyck word $[_a\, X\,]_b\, Y$.

In these three cases, we find *local* connections between the labels only, i. e. connections between two or four labels at a time. These local connections are implied by the fact, that there is a unique corresponding closing to each opening bracket in every Dyck word and vice versa (see Remark 1). With the generalized Dyck language we cannot code trees with *global* connections, such as connections between a label and a property of the tree (or a subtree). The reason is obvious: In the generalized Dyck language, the labels of one pair of brackets are independent of the labels of the other pairs of brackets; they are independent of the structure of the Dyck word as well. For example, trees with labelled nodes, where the label describes the number of nodes in the subtree or its height, cannot be coded in the generalized Dyck language $D_{2n}^{\mathcal{R}}$.

Regarding Figure 3, we see that the shape of the tree and the labels can be determined in a very simple way from its coding. Obviously, the reason is the existence of the unique factorization of Dyck words mentioned above. Again, consider the Dyck word $[_a\, X\,]_b\, Y$, where $[_a$ and $]_b$ are corresponding brackets. If this Dyck word is a coding for an ordered tree, a and b are the labels of the node and the edge to its father and X is the coding of the subtree with the node, that is labelled by a, as its root. Y is the coding of all further subtrees on the right of the node with label a. In the case of an extended ordered binary tree, a (resp. b) codes the label of the left (resp. right) son and/or the edge to its father. The left and right subtrees can be determined from X and Y in the same manner. So, the recursive structure of the Dyck words reflects the recursive structure of the corresponding tree. Note that no recomputation of Dyck words X and Y has to be performed to find the shape and the labels of the subtrees.

Now, we will present two examples. In the first one, we find all words of a generalized Dyck language with their corresponding labelled ordered and extended ordered binary trees. One can see that only local connections between the labels exist, because we obtain a correct Dyck word, if we substitute one pair of corresponding brackets in a Dyck word by another one without further changes of the Dyck word. Considering the corresponding tree, this means to change some label(s) without changing the shape of the tree or its other labels.

Example 2:
Consider the generalized Dyck language $D_{2n}^{\mathcal{R}}$ with $\mathcal{R} := \{([_1,]_1), ([_1,]_2)\}$.
In Figure 4 we find all Dyck words $w \in D_4^{\mathcal{R}}$ arranged according to their lexicographical order with the corresponding ordered trees (class (a)). From the relation follows immediately that the trees have two different types of edges, namely type 1 and type 2. Further, all nodes (except the root) are labelled by 1, thus this class of trees has only one type of nodes. Therefore, the labels of the nodes are suppressed in the picture.

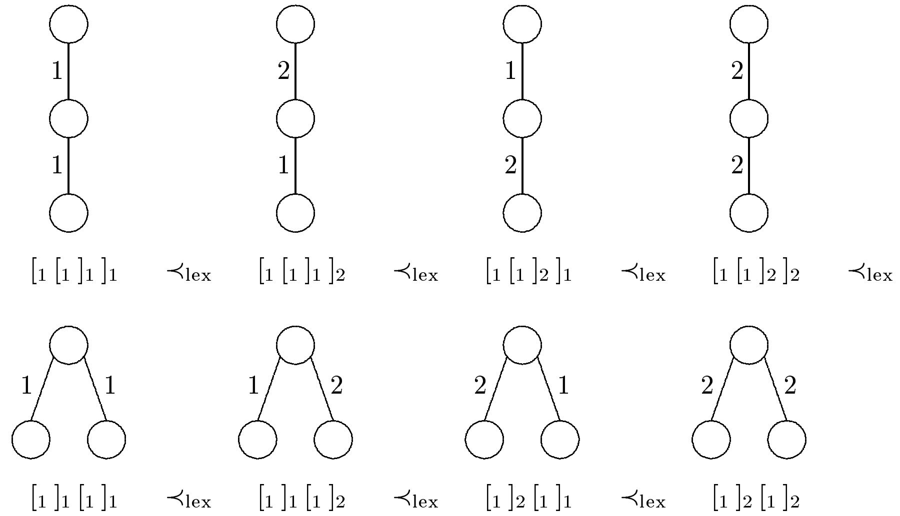

Figure 4: All Dyck words in $D_4^{\mathcal{R}}$ and their corresponding ordered trees.

In Figure 5 we present all extended ordered binary trees (class (b)) corresponding to the Dyck words $w \in D_4^{\mathcal{R}}$. Again, they are arranged according to their lexicographical order. In this case, the relation implies that there is one type of nodes that are left sons of its father and two types of nodes that are right sons.

With the unranking algorithm for the generalized Dyck language we have a procedure at hand that generates the coding of a random tree. We use a random number generator and compute the Dyck word and the corresponding labelled tree, whose rank is equal to the random number. Note the following fact: If the random number generator produces "good" random numbers, then the resulting Dyck words are codings of "good" random trees.

Example 3:
Let us have a look at two random trees according to the classes (a) and (b).
First (class (a)), we consider a class of ordered trees with labelled edges and unlabelled nodes. As in the previous example, there are two different types of edges, say type 1 and type 2. Thus, the relation is given by $\mathcal{R} := \{([_1,]_1), ([_1,]_2)\}$ again. Now, we want to generate an ordered tree with nine nodes, so $n := 16$. The random number generator computes $220609 \in [0 : D_{16}^{\mathcal{R}} - 1]$. In the left picture in Figure 6 we

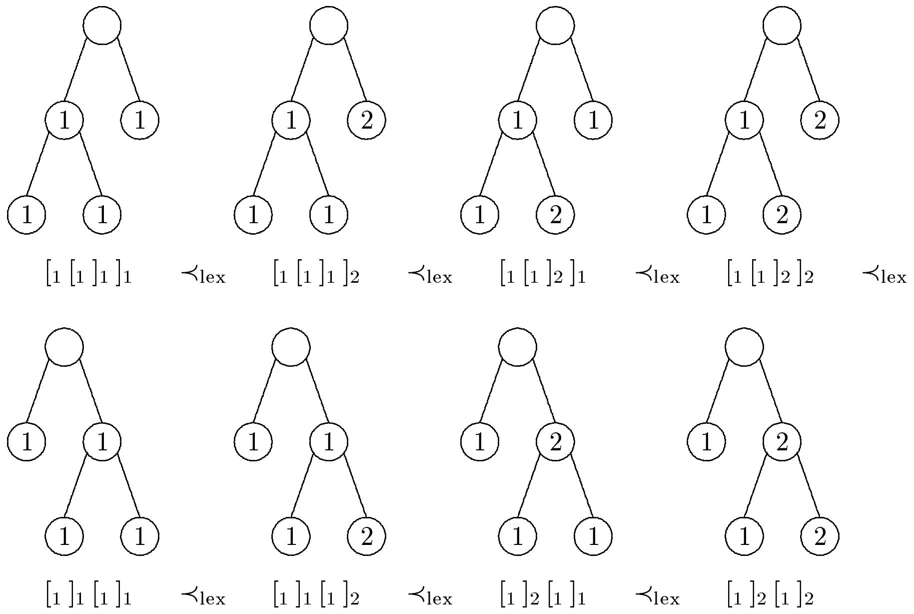

Figure 5: All Dyck words in $D_4^{\mathcal{R}}$ and their corresponding extended ordered binary trees.

find the tree that corresponds to the Dyck word $[_1 [_1]_2 [_1 [_1]_2]_1 [_1]_2]_1 [_1 [_1]_2 [_1]_1]_2$ with rank 220609. As all nodes (except the root) are labelled by 1, the labels are suppressed in the picture.
Second (class (b)), we want to generate an extended ordered binary tree with labelled nodes; there are three different types of nodes (1, 2 and 3) and the label of the left son of any node must not be smaller than the label of the right son. Obviously, we obtain $\mathcal{R} := \{([_1,]_1), ([_2,]_1), ([_2,]_2), ([_3,]_1), ([_3,]_2), ([_3,]_3)\}$. The root is unlabelled, because it does not have a brother (see Remark 5). Let $n := 4$ and let $12657 \in [0 : D_8^{\mathcal{R}}]$ be the random number. The Dyck word with rank 12657 is given by $[_2 [_1]_1 [_3]_1]_2 [_2]_1$ and the resulting tree can be found in the right picture of Figure 6.

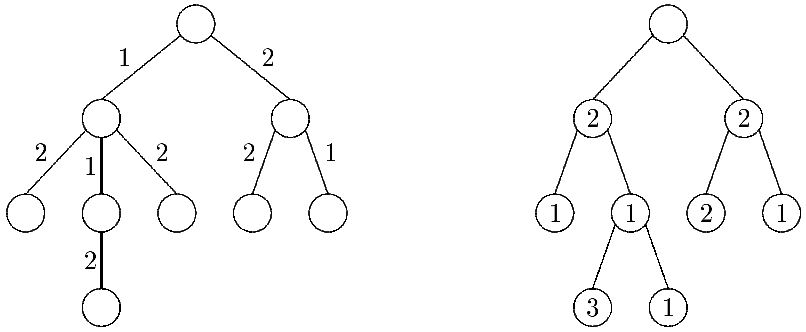

Figure 6: Two random trees (ordered and extended ordered binary).

5 Concluding Remarks

In this paper we have presented a ranking and an unranking algorithm for the generalized Dyck language. It is defined by a relation $\mathcal{R}$ which describes the pairs of brackets that can be used. The ranking (resp. unranking) algorithm reads (resp. creates) the Dyck word from left to right.

Following a general approach to ranking and unranking [Li97], we computed the s-th moments of the random variable describing the length of the prefix to be read in order to determine the rank of a Dyck word. Especially, we pointed out the mean value and the variance of the length of that prefix with precision $\mathcal{O}(n^{-1})$ for the generalized Dyck language $D_{2n}^{\mathcal{R}}$. We showed that the number of symbols to be read is linear in the length of the words with a constant variance. The unranking algorithm always has to create the whole word, so the mean value of the number of symbols to be created is equal to the length of the words and the variance is equal to 0. Both algorithms presented here are optimal with respect to the number of symbols to be read (resp. created) from left to right. As the amount of time for each symbol is $\mathcal{O}(1)$, the running-time is proportional to the number of symbols to be read or created.

Further, we mentioned one-to-one correspondences between the generalized Dyck language and ordered as well as extended ordered binary trees. With the generalization of the normal Dyck language we are able to code not only the shape of a tree but also the information of the nodes and/or the edges. Thus, with the algorithms presented here, we have ranking and unranking algorithms for a linear coding of various kinds of labelled trees at hand.

An application for the unranking algorithm is the generation of random trees, which is useful in many areas. The unranking algorithm computes the Dyck word and the corresponding labelled tree, whose rank is equal to a random number.

Acknowledgement

The author takes the opportunity to thank the two anonymous referees for their detailed remarks and their suggestions that have helped to improve the paper.

References

[AS95] L. Alonso and R. Schott, *Random Generation of Trees*, Kluwer Academic Publishers, 1995.

[BLP99] E. Barcucci, A. Del Lungo, E. Pergola, Random Generation of Trees and Other Combinatorial Objects, *Theoretical Computer Science* **218**, 1999, 219-232.

[CRS71] L. Carlitz, D. P. Roselle and R. A. Scoville, Some Remarks on Ballot-Type Sequences of Positive Integers, *Journal of Combinatorial Theory* **11**, 1971, 258-271.

[FS96] P. Flajolet and R. Sedgewick, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996.

[GKP94] R. L. Graham, D. E. Knuth and O. Patashnik, *Concrete Mathematics, 2nd ed.*, Addison-Wesley, 1994.

[Ha78] M. A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, 1978.

[Ke84] R. Kemp, *Fundamentals of the Average Case Analysis of Particular Algorithms*, Wiley-Teubner, 1984.

[Ke96a] R. Kemp, On Prefixes of Formal Languages and their Relation to the Average-Case Complexity of the Membership Problem, *Journal of Automata, Languages and Combinatorics* **1**, 1997, 4, 259-303.

[Ke96b] R. Kemp, On the Average Minimal Prefix-Length of the Generalized Semi-Dycklanguage, *RAIRO Theoretical Informatics and Applications* **30**, 1996, 6, 545-561.

[Ke98] R. Kemp, Generating Words Lexicographically: An Average-Case Analysis, *Acta Informatica* **35**, 1998, 17-89.

[Li97] J. Liebehenschel, Ranking and Unranking of Lexicographically Ordered Words: An Average-Case Analysis, *Journal of Automata, Languages and Combinatorics* **2**, 1998, 4, 227-268.

[Li98] J. Liebehenschel, Lexicographical Generation of a Generalized Dyck Language, Technical Report: *Interner Bericht 5/98*, Fachbereich Informatik, Johann Wolfgang Goethe-Universität, Frankfurt am Main, Germany, submitted.

[NW78] A. Nijenhuis, H. S. Wilf, *Combinatorial Algorithms for computers and calculators, 2nd ed.*, Academic Press, 1978.

[Wi77] H. S. Wilf, A Unified Setting for Sequencing, Ranking and Selection Algorithms for Combinatorial Objects, *Advances in Mathematics* **24**, 1977, 281-291.

[Za80] S. Zaks, Lexicographic Generation of Ordered Trees, *Theoretical Computer Science* **10**, 1980, 63-82.