Generating Lyndon brackets: a basis for the n-th homogeneous component of the free Lie algebra

J Sawada [*], C.R. Miers [†], F. Ruskey [‡]

May 9, 2001

It is well known that the Lyndon words of length n can be used to construct a basis for the n-th homogeneous component of the free Lie algebra. In this paper, we develop an algorithm which generates the standard bracketing for all Lyndon words of length n, thus constructing a basis for the n-th homogeneous component of the free Lie algebra. The algorithm runs in linear amortized time; i.e., $O(n)$ time per basis element.

1 Introduction

A *necklace* is the lexicographically smallest element in an equivalence class of strings under rotation. A *Lyndon word* is an aperiodic necklace. Let $\mathbf{L}_k(n)$ denote the set of k-ary Lyndon words of length n. The number of elements in $\mathbf{L}_k(n)$ is given by the Witt formula:

$$|\mathbf{L}_k(n)| = \frac{1}{n}\sum_{d|n} \mu(d) k^{n/d}.$$

[*]Department of Computer Science, University of Sydney, Australia. Research supported by NSERC and partial support of Czech Grant GAČR 201/99/0242 and ITI under project LN-00A 056.

[†]Department of Mathematics, University of Victoria, Canada. Research supported by NSERC.

[‡]Department of Computer Science, University of Victoria, Canada. Research supported by NSERC.

$$
\begin{array}{l}
[\,0\,,[\,0\,,[\,0\,,[\,0\,,[\,0\,,1\,]\,]\,]\,]\,]\\
[\,0\,,[\,0\,,[\,0\,,[\,[\,0\,,1\,]\,,1\,]\,]\,]\,]\\
[\,0\,,[\,[\,0\,,[\,0\,,1\,]\,]\,,[\,0\,,1\,]\,]\,]\\
[\,0\,,[\,0\,,[\,[\,[\,0\,,1\,]\,,1\,]\,,1\,]\,]\,]\\
[\,0\,,[\,[\,0\,,1\,]\,,[\,[\,0\,,1\,]\,,1\,]\,]\,]\\
[\,[\,0\,,[\,[\,0\,,1\,]\,,1\,]\,]\,,[\,0\,,1\,]\,]\\
[\,0\,,[\,[\,[\,[\,0\,,1\,]\,,1\,]\,,1\,]\,,1\,]\,]\\
[\,[\,0\,,1\,]\,,[\,[\,[\,0\,,1\,]\,,1\,]\,,1\,]\,]\\
[\,[\,[\,[\,[\,0\,,1\,]\,,1\,]\,,1\,]\,,1\,]\,,1\,]
\end{array}
$$

Figure 1: The Lyndon brackets of $\mathbf{L}_2(6)$

The *standard factorization* of a Lyndon word w ($|w| > 1$), denoted $\sigma(w)$, is the pair of Lyndon words (l, m) such that $w = lm$ where m has maximal length and l is non-empty. By Proposition 5.1.3 in Lothaire [4] such a factorization exists. Using this standard factorization, the Lyndon words can be recursively mapped into their standard bracketing using the following function:

$$
\gamma(w) = \begin{cases} w & \text{if } |w| = 1, \\ [\gamma(l), \gamma(m)] & \text{otherwise, where } \sigma(w) = (l, m). \end{cases}
$$

If w is a Lyndon word, then $\gamma(w)$ is called the *Lyndon bracket* of w. As an example, the bracketing that results when γ is applied to $\mathbf{L}_2(6)$ is illustrated in Figure 1.

Lothaire [4] and Reutenauer [6] both demonstrate that the set of standard bracketings of all Lyndon words in $\mathbf{L}_k(n)$ is a basis for the n-th homogeneous component of the free Lie algebra over an alphabet of size k. Knowledge of free Lie algebras is not required for the scope of this paper, although it helps to motivate the results. For completeness, we give a definition similar to the one given by Lothaire [4].

Let K be a commutative ring with unit. Then a Lie algebra $\mathcal{L}$ over K is a K-algebra whose product satisfies the two identities:

$$
\begin{aligned}
[x, x] &= 0, \\
[\,[x, y], z\,] + [\,[y, z], x\,] + [\,[z, x], y\,] &= 0.
\end{aligned}
$$

The latter property is known as the *Jacobi identity*. For any associative

K-algebra R the product

$$[x, y] = xy - yx, \quad \text{for } x, y \in R$$

turns the K-module R into a Lie algebra R_L. Let $K\langle \mathbf{A} \rangle$ be the free associative algebra generated by the alphabet $\mathbf{A}$. That is, $K\langle \mathbf{A} \rangle$ is the set of all finite K-linear combinations of words from $\mathbf{A}$. Multiplication and addition are defined naturally. It is a fact that

$$K\langle \mathbf{A} \rangle = \bigoplus_{n \geq 0} W_n,$$

where W_n is the set of K-linear combinations of words of length n. W_n is called the n-th homogeneous component of $K\langle \mathbf{A} \rangle$. The smallest subalgebra of $K\langle \mathbf{A} \rangle_L$ containing $\mathbf{A}$ is called the free Lie algebra over $\mathbf{A}$ which is denoted $\mathcal{L}(\mathbf{A})$. We denote the nth homogeneous component of $\mathcal{L}(\mathbf{A})$, $(n \geq 0)$, by $\mathcal{L}_n(\mathbf{A})$ and thus

$$\mathcal{L}(\mathbf{A}) = \bigoplus_{n \geq 1} \mathcal{L}_n(\mathbf{A}).$$

Muthe-Kaas and Owren [5] discuss applications in numerical algorithms which use computations in free Lie algebras. Using the Lyndon words, several finely homogeneous computations (the number of each alphabet symbol is fixed) in free Lie algebras are disussed by Andary [1], including the computation of a (non-standard) left-bracketing. However, we know of no algorithms for producing the standard Lyndon bracketing. A naïve implementation of the definitions gives an algorithm with running time $O(n^3)$ per bracketing; we improve this to $O(n)$ per bracketing by integrating the computation of a *bracketing table* into a fast algorithm for generating Lyndon words.

The algorithm of this paper has been incorporated into the "Combinatorial Object Server" at `www.theory.csc.uvic.ca/~cos` under the "necklace" section.

1.1 Generating Lyndon words

In this subsection we briefly describe an algorithm for generating Lyndon words that runs in constant amortized time. This algorithm will be used in the following section when we generate the Lyndon words with their standard bracketing. For a more comprehensive description of the algorithm and its analysis, see [2].

```
procedure Lyndon ( t, p : ℕ );
local j : ℕ;

    if t > n then
        if p = n then Print()
    else
        a_t := a_{t-p};    Lyndon( t + 1, p );
        for j ∈ {a_{t-p} + 1, ..., k - 2, k - 1} do
            a_t := j;    Lyndon( t + 1, t );
```

Figure 2: An algorithm for generating Lyndon words

A *pre-necklace* is a prefix of some necklace. The general approach of the algorithm we use for generating Lyndon words is to generate all length n pre-necklaces. The pre-necklace being generated is stored in the array a; one position for each character. We assume that $a_0 = 0$. Pseudocode for this algorithm is shown in Figure 2. The initial call is Lyndon(1,1) and each recursive call appends a character to the pre-necklace to get a new pre-necklace. At the beginning of each recursive call to Lyndon(t, p), the length of the current pre-necklace is $t - 1$ and the length of the longest Lyndon prefix of that pre-necklace is p. As long as the length of the current pre-necklace is less than n, each call to Lyndon(t, p) makes one recursive call for each value from a_{t-p} to $k - 1$, updating the values of both t and p in the process. Once t exceeds n, the resulting pre-necklace is a Lyndon word exactly when $p = n$.

2 Generating Lyndon brackets

In this section we develop an algorithm for generating the length n Lyndon brackets over an alphabet of size k (or equivalently, a basis for the n-th homogeneous component of the free Lie algebra). The Lyndon words can be generated in constant amortized time using the algorithm Lyndon(t, p), but the problem of generating these words with their respective bracketing previously had no fast solution.

A naïve approach to generating these strings is to generate a Lyndon word of length n and then test each proper right factor (starting with max-

```
procedure PrintBracket ( ℓ, r : ℕ );

    if ℓ = r then print(a_ℓ);
    else
        print(" [ ");
        PrintBracket( ℓ, split(ℓ,r) − 1 );
        print(" , ");
        PrintBracket( split(ℓ,r), r );
        print(" ] ");
```

Figure 3: A function to print the brackets of a Lyndon word

imal length) until a Lyndon word is found. This process is repeated recursively for each of the two factors until a factor of length 1 is reached. Verifying whether or not each factor is a Lyndon word can be computed in linear time using Duval's algorithm for factoring a string into Lyndon words [3] or using Theorem 2.1 from [2]. In the worst case, this test will have to be performed for each proper right factor. Since this must be done recursively for each factor, the total running time to generate each Lyndon bracket from the Lyndon word will be $O(n^3)$.

A significant improvement can be made to this naïve algorithm from the following observation. If the longest proper right factor is known for each Lyndon factor, then the bracketing for each Lyndon word can be determined in linear time using the recursive function $\mathsf{PrintBracket}(\ell,r)$ displayed in Figure 3. In this function, the Lyndon word being generated $a_1 \ldots a_n$ is stored in the array a. If $a_i \ldots a_j$ is a Lyndon word then the value of $\mathit{split}(i,j)$ is the starting position of its longest proper right factor. Observe that $\mathit{split}(i,j)$ is defined only for $i < j$. As an example the $\mathit{split}(i,j)$ values for the Lyndon word 001001011 are displayed in Figure 4. In this figure the value i represents the row number and the value j represents the column number, where each value ranges from 1 to n. The entry $\mathit{split}(1,n)$ determines the starting point of the longest proper right factor in the original Lyndon word, and thus the standard factorization of 001001011 is (001,001011).

Observe that the value $\mathit{split}(i,j)$, where $i < j$, can be defined recursively

$$\begin{bmatrix} - & 2 & 2 & 4 & 5 & 4 & 7 & 4 & 4 \\ - & - & 3 & 4 & 5 & 4 & 7 & 4 & 4 \\ - & - & - & 4 & 5 & 4 & 7 & 4 & 4 \\ - & - & - & - & 5 & 5 & 7 & 7 & 5 \\ - & - & - & - & - & 6 & 7 & 7 & 7 \\ - & - & - & - & - & - & 7 & 7 & 7 \\ - & - & - & - & - & - & - & 8 & 9 \\ - & - & - & - & - & - & - & - & 9 \\ - & - & - & - & - & - & - & - & - \end{bmatrix}$$

Figure 4: The values $split(i, j)$ for the Lyndon word 001001011

by the following recurrence relation.

$$split(i,j) = \begin{cases} i+1 & \text{if } a_{i+1}\cdots a_j \text{ is a Lyndon word,} \\ split(i+1,j) & \text{otherwise.} \end{cases}$$

Thus, the key to obtaining each value $split(i, j)$ is to determine all the Lyndon factors embedded in the Lyndon word $a_1 \cdots a_n$. This can be done by modifying the algorithm $\mathsf{Lyndon}(t, p)$ of Figure 2. In this algorithm, the parameter p maintains the length of the longest Lyndon prefix of the pre-necklace $a_1 \cdots a_{t-1}$. However, to obtain all the Lyndon factors, the longest Lyndon prefix must be maintained for all strings $a_i \cdots a_{t-1}$ where $1 \leq i < t$. If this value is stored in p_i, then effectively, the parameter p is replaced with the global string of values $p_1 \cdots p_n$. If the string starting at a_i is not a pre-necklace, then p_i is assigned the value 0; otherwise, it maintains the length of the longest Lyndon prefix starting at a_i. Note that the value p_1 maintains the value of the old parameter p. Using the values $p_1 \cdots p_t$, the values for $split(i, t)$ can be determined using its associated recurrence; the string $a_{i+1} \cdots a_t$ is a Lyndon word when $p_{i+1} = t - i$. The function $\mathsf{LyndonBracket}(t)$ shown in Figure 5 is the result of applying these modifications to the algorithm $\mathsf{Lyndon}(t, p)$. The initial call is $\mathsf{LyndonBracket}(1)$ and the values p_i are initialized to 1.

Updating the values for $p_1 \cdots p_t$ and $split(i, t)$ where $1 \leq i < t$ takes linear time. The function $\mathsf{PrintBracket}(\ell, r)$ also takes linear time. Thus, since the the algorithm $\mathsf{Lyndon}(t, p)$ for generating Lyndon words runs in constant amortized time, we obtain the following theorem.

```
procedure LyndonBracket ( t : ℕ );
local i, j : ℕ;   q : array of ℕ;

    if t > n then
        if n = p_1 then PrintBracket( 1, n ); println();
    else
        q := p;
        for j from a_{t-p_1} to k - 1 do
            a_t := j;
            for i from 1 to t - 1 do
                if a_t < a_{t-p_i} then p_i := 0;
                if a_t > a_{t-p_i} then p_i := t - i + 1;
            for i from t - 1 downto 1 do
                if p_{i+1} = t - i then split(i, t) := i + 1;
                else split(i, t) := split(i + 1, t);
            LyndonBracket( t + 1 );
            p := q;
```

Figure 5: An algorithm for generating Lyndon brackets

THEOREM 1 *The algorithm LyndonBracket(t) for generating k-ary Lyndon brackets of length n runs in $O(n)$ amortized time.*

The ideas developed in the algorithm can be used determine the Lyndon bracket for a given length n Lyndon word in $O(n^2)$ time. This is done by computing the *split*(i, j) values for the Lyndon word and using the PrintBracket(ℓ, r) function.

References

[1] P. Andary, Finely homogeneous computations in free Lie algebras, Discrete Mathematics and Theoretical Computer Science, 1 (1997) 101-114.

[2] K. Cattell, F. Ruskey, J. Sawada, C.R. Miers, M. Serra, Fast algorithms to generate necklaces, unlabeled necklaces and irreducible polynomials over GF(2), Journal of Algorithms, Vol 37. No 2. (2000) 267-282.

[3] J-P. Duval, Factoring words over an ordered alphabet, Journal of Algorithms, 4 (1983) 363-381.

[4] M. Lothaire, *Combinatorics on Words*, Cambridge University Press, 1983.

[5] H. Munhte-Kass, B. Owren, Computations in a free Lie algebra, Philosophical Transactions of the Royal Society of London: Series A, 357 (1999) 957-981.

[6] C. Reutenauer, *Free Lie Algebras*, Clarendon Press, Oxford, 1993.