

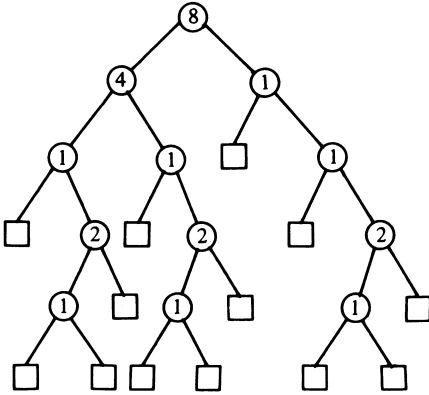


**Proof**

By induction on  $n$  by using Lemma 1.

The weight sequence of a binary tree can be obtained by an inorder traversal (traverse the left subtree, visit the root, traverse the right subtree) after labelling each internal node with the weight of its left subtree.

If  $T =$



then  $w_T = (1, 1, 2, 4, 1, 1, 2, 8, 1, 1, 1, 2)$

The following algorithm enables us, starting from a weight sequence, to compute the next one in the lexicographic order. In order to obtain all the  $n$ -weight sequences, we need only begin with  $(1, 1, \dots, 1)$ .

**Algorithm TREE GENERATION**

*Begin with*  $(w_1, w_2, \dots, w_n) = (1, 1, \dots, 1)$

**While**  $i = \max \{k | w_k < k\}$  **exists do**:

$j \leftarrow i - w_i$     $w_i \leftarrow i - j + w_j$

**For**  $m = i + 1$  **to**  $n$  **do**  $w_m \leftarrow 1$  **enddo.**

List of the weight sequences of the 42 binary trees with five internal nodes in lexicographic order:

11111	11211	12111
11112	11212	12112
11113	11214	12113
11114	11215	12115
11115	11231	12121
11121	11234	12123
11123	11235	12125
11124	11241	12141
11125	11245	12145
11131	11311	12311
11134	11312	12312
11135	11315	12315
11141	11341	12341
11145	11345	12345

In order to generate this list, the algorithm had to use 64 comparisons of integers, i.e. an average number of comparisons equal to  $64/42 = 1.52$ .

The work made by the previous algorithm is proportional to the number of comparisons made with a view to finding  $i = \max \{k | w_k < k\}$ . Let  $t_{n,j}$  denote the number of  $n$ -weight sequences that require  $j$  comparisons in the generating algorithm, i.e. the number of  $n$ -weight sequences  $w$  such that  $w = (w_1, \dots, w_{n-j+1}, n-j+2, \dots, n)$  and  $w_{n-j+1} < n-j+1$ . Therefore  $t_{n,j}$  is also the number of  $(n-j+1)$  weight sequences which do not end in  $n-j+1$ . The number of  $(n-j+1)$  weight sequences

ending in  $n-j+1$  being equal to  $b_{n-j}$ , we deduce that  $t_{n,j} = b_{n-j+1} - b_{n-j}$ . The total number of comparisons in order to generate the  $n$ -weight sequences is

$$\sum_{j=1}^n j t_{n,j} = \sum_{j=1}^n j (b_{n-j+1} - b_{n-j}) = \sum_{k=1}^n b_k.$$

The average number of comparisons per sequence is equal to

$$\frac{1}{b_n} \sum_{k=1}^n b_k, \text{ which tends to } 4/3 = 1.33 \text{ as } n \text{ increases.}$$


This algorithm is the most efficient one to generate lexicographically all the  $n$ -node binary trees by making use of  $n$ -integer sequences such that  $1 \leq s_i \leq n$  for all  $i$ . Indeed, the average number of comparisons per  $n$ -feasible sequences<sup>12</sup> and per  $n$ -z sequences<sup>14</sup> tends respectively to 3 and  $4/3$  as  $n$  increases. However,  $n$ -z sequences verify  $1 \leq z_i \leq 2n$  for all  $i$ .

**3. RANKING AND UNRANKING****Definition**

The height  $h$  ( $1 \leq h \leq n$ ) of an  $n$ -weight sequence  $w$  is the integer  $h = \max \{i | w_i = 1\}$ . Let  $\sigma_{n,h}$  denote the number of  $n$ -weight sequences of height  $h$ .

**Lemma 2**

To obtain the trees corresponding to the  $n$ -weight

sequences of height  $h$ , we just have to substitute  for the external node  $h$  of trees corresponding to the  $(n-1)$  weight sequences of height  $\geq h-1$ .

**Corollary**

We have

$$\sigma_{n,1} = b_{n-1}, \sigma_{n,n} = 1, \sigma_{n,h} = \sum_{j=h-1}^{n-1} \sigma_{n-1,j} \quad \text{for } 2 \leq h \leq n-1.$$

These recurrence relations allow us to build the table:

	1	2	3	4	5	6	7	8
1	1							
2	1	1						
3	2	2	1					
4	5	5	3	1				
5	14	14	9	4	1			
6	42	42	28	14	5	1		
7	132	132	90	48	20	6	1	
8	429	429	297	165	75	27	7	1

Explicitly, we have

$$\sigma_{n,h} = (h/2n-h) \binom{2n-h}{n-h} \quad \text{for } h \geq 2.$$

Let  $\text{rank}(w_1, w_2, \dots, w_n)$  denote the position of the  $n$ -weight sequence  $(w_1, w_2, \dots, w_n)$  in the lexicographic ordering of all these  $n$ -weight sequences.

**Theorem**

Given an  $n$ -weight sequence  $w$ , we have the relation:

$$\text{rank}(w_1, \dots, w_n) = \sum_{j=h+1}^n \sigma_{n,j} + \text{rank}(\bar{w}_1, \dots, \bar{w}_{n-1}),$$

where  $h$  is the height of  $w$



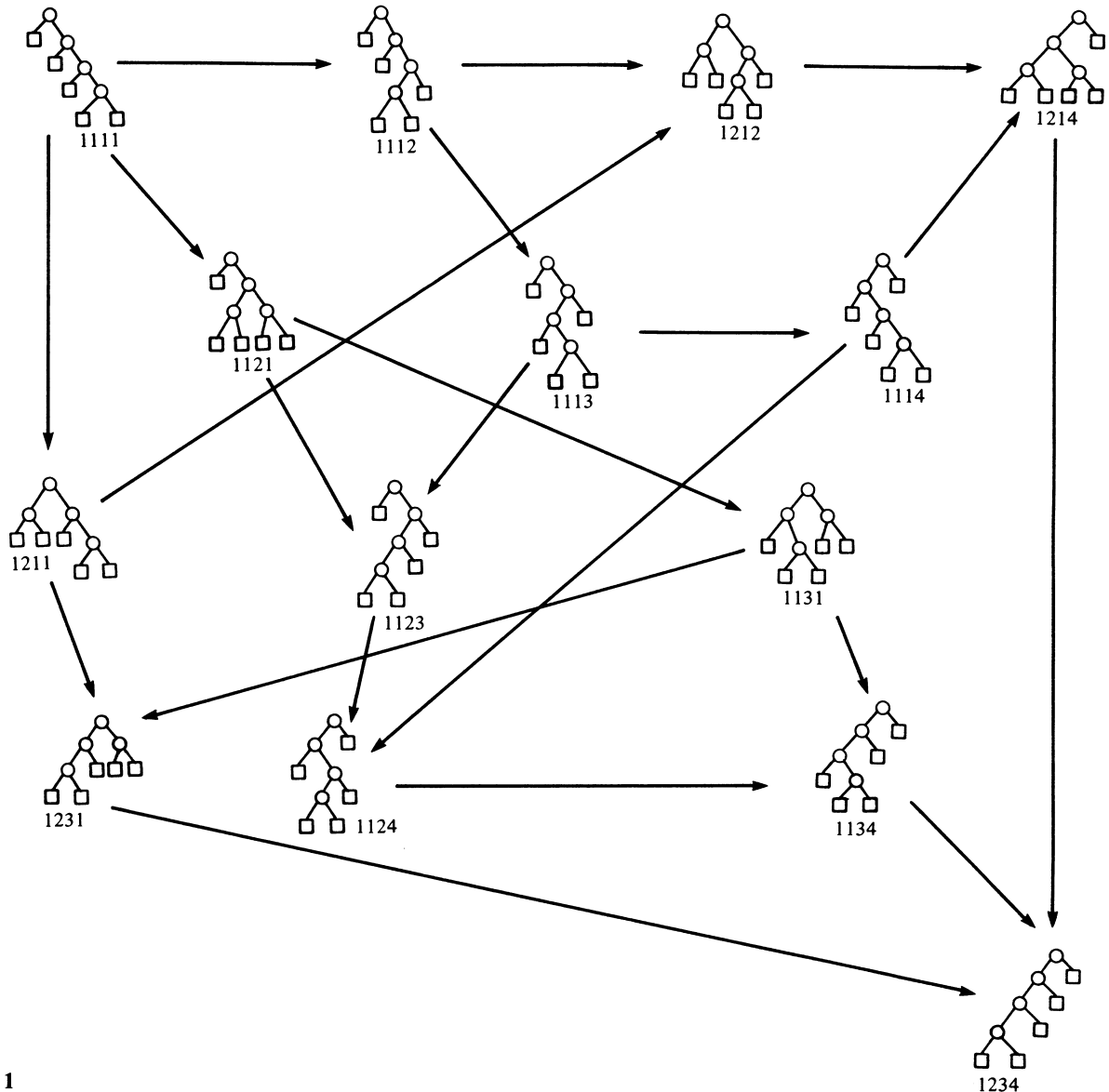


Figure 1

*Proof*

If  $T \rightarrow T'$  and  $T' \rightarrow T$ , then  $w_T \leq w_{T'}$  and  $w_{T'} \leq w_T$  by theorem 2, thus  $w_T = w_{T'}$ . Therefore  $T = T'$  and  $\rightarrow$  is antisymmetric. Q.E.D.

**Corollary**

$(B_n, \rightarrow)$  is a lattice.

*Proof*

Let us prove that any pair of  $n$ -node binary trees  $T$  and  $T'$  has an infimum  $T \wedge T'$ . If  $T'' \rightarrow T$  and  $T'' \rightarrow T'$ , then  $w_{T''}(i) \leq \inf(w_T(i), w_{T'}(i))$  for all  $i$ . We just have to show that  $w_i = \inf(w_T(i), w_{T'}(i))$  is a weight sequence, i.e. verifies conditions of Theorem 1. It is obvious that  $1 \leq w_i \leq i$ . If  $i' \in [i - w_i + 1, i]$  then  $i - w_{T'}(i) \leq i' - w_{T'}(i')$  and  $i - w_T(i) \leq i' - w_T(i')$  from which we deduce  $i - w_i \leq i' - w_i$ . Therefore  $w_{T \wedge T'}(i) = \inf(w_T(i), w_{T'}(i))$  for all  $i$ .

Now let us define recursively the dual tree  $\bar{T}$  of a tree

$$T \text{ by } \overline{\square} = \square \text{ and } \overline{T'} = \overline{T''} \text{ where } T' = \begin{array}{c} \square \\ \swarrow \quad \searrow \\ \square \quad \square \end{array} \text{ and } T'' = \begin{array}{c} \square \\ \swarrow \quad \searrow \\ \square \quad \square \end{array}$$

Since  $T \rightarrow T'$  iff  $\bar{T} \rightarrow \bar{T'}$ , any pair of  $n$ -node binary trees  $T$  and  $T'$  has a supremum  $T \vee T' = \overline{\bar{T} \wedge \bar{T'}}$ . Remark that  $w_{T \vee T'}(i) \neq \sup(w_T(i), w_{T'}(i))$  generally. Q.E.D.

**5. APPLICATIONS TO TERM REWRITING SYSTEMS**

The weight sequence of an arithmetic expression  $E$  in infix notation is obtained by scanning  $E$  from left to right and by associating to each operator the number of variable symbols of its left operand.

Let  $E = (((x+y) + (z+(y+t))) + ((u+(v+x)) + t))$ , an expression whose corresponding tree is the one given in Section 2. Then  $w_E = (1, 2, 1, 1, 5, 1, 1, 3)$ .

When associativity occurs in term rewriting systems, we orient the associativity axiom:  $x.(y.z) \rightarrow (x.y).z$ . Given two arithmetic expressions  $E$  and  $E'$  of same length, having the same and unique operator, and whose variables appear in the same order from left to right, Theorem 2 allows us to see immediately if  $E \rightarrow E'$  ( $\rightarrow$  is the rewriting relation), i.e. if one can go from  $E$  to  $E'$  by moving parentheses in the same direction. Furthermore, the proof of theorem 2 allows us to construct a path going from  $E$  to  $E'$  by the following algorithm.

**Algorithm PATH CONSTRUCTION**

**while**  $i = \min \{k \mid w_E(k) < w_{E'}(k)\}$  exists  
**do**  $j \leftarrow i - w_E(i) + 1$   $k \leftarrow j - w_{E'}(j - 1)$   
 $l = \max \{m \geq i \mid j = m - w_E(m) + 1\}$   
 $w_E(l) \leftarrow l - k + 1$   
**enddo.**

This path has a minimal length which is bounded above by  $|E|$ .

**Example**

Let  $E = (x + (y + ((z + s) + (t + (u + (v + q))))))$  and  
 $E' = (((((x + y) + (z + s)) + t) + u) + (v + q))$   
 $w_E = (1, 1, 1, 2, 1, 1, 1)$  and  $w_{E'} = (1, 2, 1, 4, 5, 6, 1)$ .  
 Consequently  $E \xrightarrow{*} E'$ .

We construct below a minimal path joining  $E$  and  $E'$ :

$E = (x + (y + ((z + s) + (t + (u + (v + q))))))$   
 $\begin{cases} s_E = (1, 1, 1, 2, 1, 1, 1) \\ s_{E'} = (1, 2, 1, 4, 5, 6, 1) \end{cases} \quad i = 2 \quad j = 2 \quad k = 1 \quad l = 2$   
 $\downarrow$   
 $E^1 = ((x + y) + ((z + s) + (t + (u + (v + q)))))$   
 $\begin{cases} s_{E^1} = (1, 2, 1, 2, 1, 1, 1) \\ s_{E'} = (1, 2, 1, 4, 5, 6, 1) \end{cases} \quad i = 4 \quad j = 3 \quad k = 1 \quad l = 4$   
 $\downarrow$   
 $E^2 = (((x + y) + (z + s)) + (t + (u + (v + q))))$   
 $\begin{cases} s_{E^2} = (1, 2, 1, 4, 1, 1, 1) \\ s_{E'} = (1, 2, 1, 4, 5, 6, 1) \end{cases} \quad i = 5 \quad j = 5 \quad k = 1 \quad l = 5$   
 $\downarrow$

$E^3 = (((((x + y) + (z + s)) + t) + u) + (v + q))$   
 $\begin{cases} s_{E^3} = (1, 2, 1, 4, 5, 1, 1) \\ s_{E'} = (1, 2, 1, 4, 5, 6, 1) \end{cases} \quad i = 6 \quad j = 6 \quad k = 1 \quad l = 6$

$\downarrow$   
 $E' = (((((x + y) + (z + s)) + t) + u) + (v + q))$

If an arithmetic expression  $E$  has  $x$  as variable  $k$  and  $x^{-1}$  as variable  $k + 1$  with  $w_E(k) = w_E(k + 1) = 1$ , and if we want to apply the rewrite rule  $x \cdot x^{-1} \rightarrow e$ , the following algorithm allows us to compute the weight sequence of an expression  $E'$  such that  $E \xrightarrow{*} E'$  for the associative rule and  $w_{E'}(k) = 1$ ,  $w_{E'}(k + 1) \geq 2$ .

**while**  $w_{k+1} = 1$   
**do**  $l = \max \{m \geq k + 1 \mid m - w_m = k\}$   
 $w_l \leftarrow w_l + 1$   
**enddo.**

The same method is applicable if we want to apply the rewrite rule  $(x \cdot y) \cdot y^{-1} \rightarrow x$  on an expression  $E$  which has  $x$  as variable  $k$ ,  $y$  as variable  $k + 1$ ,  $y^{-1}$  as variable  $k + 2$ ,  $w_E(k) = 1$  and  $w_E(k + 1) \leq 2$ . We obtain  $E'$  such that  $E \xrightarrow{*} E'$ ,  $w_{E'}(k) = 1$ ,  $w_{E'}(k + 1) = 2$ ,  $w_{E'}(k + 2) \geq 3$ .

Given an arithmetic expression  $E$  containing operators of different types, we define the sequence  $c_E$  in the following manner:  $c_E(i)$  is the operator adjacent to the set of parentheses adjacent to the variable  $i$  (except the first and the last variable).

For example, if  $E = (((x + y)/(z + t)) + x)$  then  $c_E = (/ , / , +)$ .

If we can apply several oriented associativity rules in expressions having different operators, then  $E \xrightarrow{*} E'$  iff the variables of  $E$  and  $E'$  appear in the same order from left to right,  $w_E \leq w_{E'}$  and  $c_E = c_{E'}$ .

**REFERENCES**

1. T. Beyer and S. M. Hedetniemi, Constant time generation of rooted trees. *SIAM Journal on Computing* **9** (4), 706–712 (1980).
2. A. Bonnin and J. M. Pallo, A-Transformation dans les arbres  $n$ -aires. *Discrete Mathematics* **45**, 153–163 (1983).
3. K. Culik and D. Wood, A note on some tree similarity measures. *Information Processing Letters* **15**, 39–42 (1982).
4. M. C. Er, A note on generating well-formed parenthesis strings lexicographically. *The Computer Journal* **26** (3), 205–207 (1983).
5. G. Huet and D. Oppen, Equations and rewrite rules: a survey. *Formal Languages Theory: Perspectives and Open Problems*, pp. 349–405. Academic Press, London (1980).
6. G. D. Knott, A numbering system for binary trees. *Comm. ACM* **20** (2), 113–115 (1977).
7. D. E. Knuth, *The Art of Computer Programming*, vol. 1: *Fundamental Algorithms*. Addison-Wesley, London (1968).
8. D. E. Knuth, *The Art of Computer Programming*, vol. 3: *Sorting and Searching*. Addison-Wesley, London (1973).
9. D. E. Knuth and P. Bendix, Simple word problems in universal algebras. In *Computational Problems in Abstract Algebras*, pp. 263–297. Pergamon Press, Oxford (1970).
10. W. R. Lalonde and J. Des Rivières, Handling operator precedence in arithmetic expressions with tree transformations. *ACM Transactions on Programming Languages and Systems* **3** (1), 83–103 (1981).
11. D. Rotem and Y. L. Varol, Generation of binary trees from ballot sequences. *Journal of the ACM* **25** (3), 396–404 (1978).
12. F. Ruskey and T. C. Hu, Generating binary trees lexicographically. *SIAM Journal on Computing* **7** (4), 492–509 (1978).
13. M. Solomon and R. A. Finkel, A note on enumerating binary trees. *Journal of the ACM* **27** (1), 3–5 (1980).
14. S. Zaks, Lexicographic generation of ordered trees. *Theoretical Computer Science* **10**, 63–82 (1980).