JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **34,** 1–13 (1996)
ARTICLE NO. 0041

A Log (N) Distributed Mutual Exclusion Algorithm Based on Path Reversal[1]

MOHAMED NAIMI,* MICHEL TREHEL,* AND ANDRÉ ARNOLD†

*LIB, *Faculté des sciences, Route de Gray 25030 Besancon Cedex, France; and* †LaBRI, *Université Bordeaux* I, *33405 Talence Cedex, France*

In this paper, we present a distributed algorithm for mutual exclusion based on path reversal. The algorithm does not use logical clocks to serialize the concurrent events, and all the variables are bounded. When a process invokes a critical section, it sends a request to the tail of a queue. A dynamical rooted tree gives the path to this tail. The algorithm requires only $O(\log(n))$ messages on average, where n is the number of processes in the network. The performance analysis of the algorithm is based on generating formal power series. © 1996 Academic Press, Inc.

1. INTRODUCTION

Algorithms for mutual exclusion may vary from centralized system to distributed system. In a centralized system, one process is designated as the control process; this controls access to a shared object. Whenever a process wishes to gain access to the shared object, it sends a request message to the control process, which returns a reply (permission) message when the shared object becomes available. A centralized mutual exclusion algorithm is characterized by the following properties:

—only the central (control) process makes a decision
—all necessary information is concentrated in the central process.

In a distributed system, the design of a mutual exclusion algorithm consists of defining the protocols used to coordinate access to a shared object. A distributed algorithm for mutual exclusion is characterized by these properties:

—all processes have an equal amount of information
—all processes make a decision based on local information. Many distributed algorithms for mutual exclusion have been proposed.

In Lamport's algorithm [17], each process has a queue. A process which wants to enter the critical section broadcasts a request message with a time-stamp. The return of a time-stamped acknowledgment allows it to check whether a process has invoked the critical section earlier than it has. The number of exchanged messages is $3(N - 1)$, where N is the number of processes in the network.

[1] This work was financed in part by the CNRS: Unité associée 040822 and the C^3 coordinated research program.

Several algorithms reducing the number of messages were presented later (see Ricart and Agrawala [21] and Carvalho and Roucairol [4]). The number of messages was proportional to N. The algorithm presented by Chandy and Misra [6] (in which permission is in the form of a fork) is the most efficient of the three in terms of message complexity per resource. Maekawa [19] introduced the notion of arbitraring process. A process wishing resource access must obtain permission from a fixed set of $\sqrt{N}$ arbitraring processes. Each arbitraring process gives permission on behalf of itself and $(\sqrt{N} - 1)$ other processes. The amortized message complexity of this algorithm is $3(\sqrt{N} - 1)$.

In this paper, we present an algorithm which relies on a new approach. Every process i maintains two pointers, "father" and "next," at any time; the pointer "father" indicates the process to which requests for critical section access should be forwarded, and the pointer "next" indicates the node to which access permission should be forwarded after process i leaves its critical section. The key point of the algorithm is a very simple dynamic scheme for updating the pointers while processing a sequence of requests.

This paper is organized as follows: In Section 2, we introduce the algorithm; in Section 3, we prove its correctness; and in Section 4, we compute its complexity.

2. MODEL DESCRIPTION AND SPECIFICATION OF THE ALGORITHM

2.1. General Model

Each process has a local memory and can send messages to any other by using a unique identifier as an address. The communication between processes is assured to be perfect (no losses, duplications, or modifications of messages).

2.2. Description

The algorithm we present here is based on the fact that a process sends its request to only one other process and awaits its permission.

It would be a centralized algorithm if the receiving process were always the same, but the receiving process will change at every request.

2.2.1. Simplified Description

The permission to enter a critical section is materialized by a token; one and only one process has a token. The process which owns the token may enter the critical section.

There are two data structures:

The first one is a queue. Each process knows the next process in the queue only if this "next" exists. The head is the process possessing the token. The tail is the last process which has requested the critical section (excepted when there is only one process in the queue). A path is organized so that, when a process requests entering the critical section, the request message is transmitted to the tail. When the request arrives at the tail, there are two possible situations.

—The tail has the token and is not in the critical section. It sends the token to the requesting process. That one is authorized to enter the critical section.

—The tail has the token and uses it (that means: it is in the critical section) or it waits for the token. In this case, the requesting process is linked to the tail.

—When the process at the head leaves the critical section, it gives the token to its next in the queue (if there is one "next").

The second data structure gives the path to go to the tail. It is a logical rooted tree. A process which invokes the critical section sends its request to its "father." From "father" to "father," a request is transmitted to the root (the process for which "father = nil"), which is also the tail of the queue. It is a distributed structure: every process knows only its "father."

Furthermore, if the requesting process is not the root, the rooted tree is transformed: the requesting process is the new root and the processes located between the requesting process and the root will have the new root as "father."

2.2.2. Taking Transfer Times into Account

The tree structure is modified as follows: the requester transmits its request to the "father" and regards itself as the new root. There are therefore two rooted trees: one associated with the old root, and one associated with the new root. More generally, if there are several requests in transit at the same time, then there are several rooted trees. When all the messages in transit have arrived, these rooted trees are grouped to form one single rooted tree.

The transformation of the tree structure has this consequence: if the request of i is sent in the direction of root j and if the request of k arrives at j before the request of i, the request of i will be transmitted to k. See Fig. 1.

2.3. Specification of the Algorithm

Local Variables.

token_present: boolean (token_present is true if the process owns the token, false otherwise)

requesting_c_s: boolean (requesting_c_s is true if the process has invoked the critical section and remains true until i releases the critical section.

```
Algorithm of every process i

Initialization
begin
    father :=1      {the initialization of father is the same for every
    process}
    next :=nil
    requesting_c_s := false
    token_present := (father = i)
    if (father = i)   then
        father := nil
    endif
end {Initialization}

Procedure Request_C_S
begin
   requesting_c_s := true
   if (father ≠ nil) then
   begin
        send Req(i) to father
        father := nil
   endif
end {Request_C_S}

Procedure Release_C_S
begin
   requesting_c_s:=false
   if next ≠ nil then
   begin
        send Token() to next
        token_present:= false
        next := nil
   endif
end {Release_C_S}

Upon receiving the message Req(k)
        {k is the requesting process}
do
    if (father=nil then
        if requesting_c_s then
            next:= k
        else token_present := false
            send Token() to k
        endif
    else
      send Req (k) to father
    endif
    father := k
od

Upon receiving the message Token()
do
    token_present := true
od
```

FIG. 1. A log(n) distributed mutual exclusion algorithm.

next, father: $1..n \cup \{\text{nil}\}$ (n is the number of processes; nil means indefinite)

Messages Used.

Req(k): request sent by process k to the root.

Token ( ): transmission of the token.

2.4. Example

The possible transformations of the initial rooted tree are shown in Fig. 2.

Initial state of the distributed algorithm.
Process 1 has the token (fig. 2. a)

Process 2 invokes the mutual exclusion. It sends a request to process 1, and becomes a new root.
Process 1 receives the request from process 2 and sends the token to it. The father of process 1 becomes 2.
Process 2 receives the token and enters the critical section (fig. 2. b)

Process 3 invokes the mutual exclusion.
It sends a request to process 1 which transmits it to process 2.
Process 2 receives the request, and considers 3 as its next (thin line). The father of processes 1 and 2 becomes 3 (fig. 2. c)

Process 4 invokes the mutual exclusion.
It sends a request to process 1 which transmits it to process 3.
Process 3 receives the request sent by process 1, its next becomes the process 4.
The father of process 1 becomes 4 (fig. 2. d)

Process 2 releases the critical section, and sends the token to its next, the process 3 (fig. 2. e).

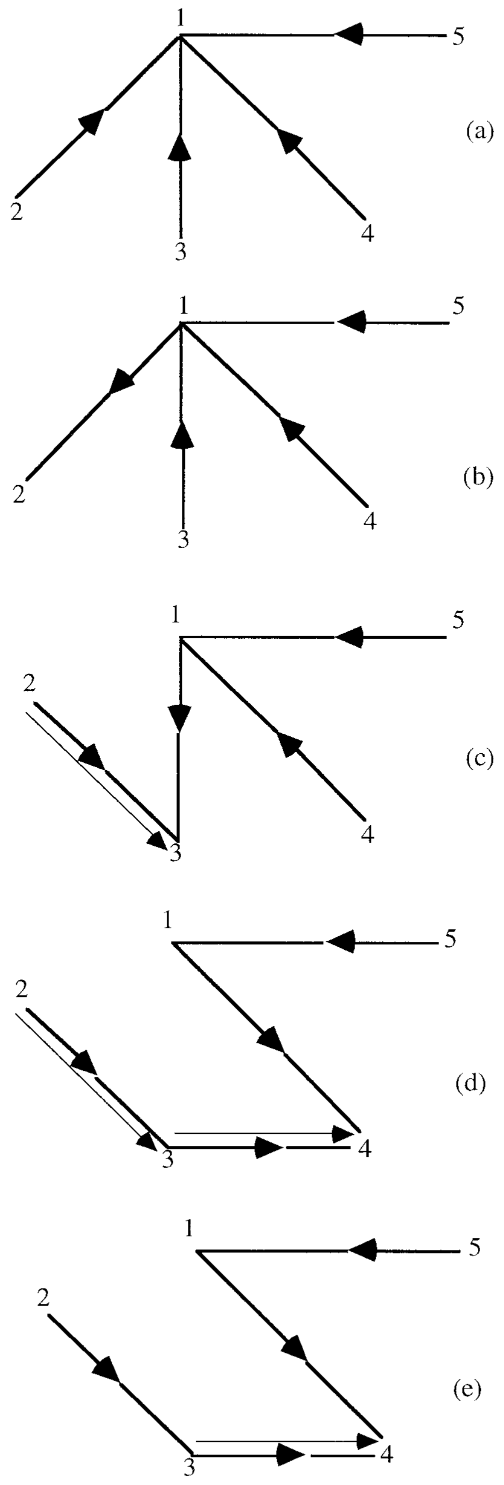

FIG. 2. Possible transformations of the initial rooted tree.

3. INFORMAL PROOF

A formal proof of this algorithm is developed by Ginat and Shankar [13] and Bouabdallah [5].

3.1. Mutual Exclusion

In the initial state of the algorithm, only one process has the token: the root. There are two situations in which a process will transmit the token to another process:

(a) when it releases the critical section and its queue is not empty;
(b) when it receives a request at a time when it holds the token and it is not in the critical section.

In these two situations, it sends the token to only one process, and loses it for itself. Consequently, there is always at most one process which has the token. No process will enter the critical section until it has the token. Mutual exclusion is therefore guaranteed.

3.2. Preliminary Lemmas for Absence of Deadlock and of Starvation

Deadlock means that, while no process is in the critical section, no requesting site can ever enter the critical section.

Starvation occurs when one process must wait indefinitely to enter the critical section even though other processes are entering and leaving the critical section.

We are going to establish a theorem showing that the process requesting entry to the critical section will have its request satisfied within a finite delay. This shows that the algorithm avoids both deadlock and starvation.

Consider process i requesting entry to the critical section and let us examine the procedures **Request_C_S** and the event of receiving a request "**Req**(k)."

If i is the root, it waits for the token. Otherwise, the request of i is transmitted, by the arcs corresponding to "father," to a process j for which ("father = nil"). If j has invoked the critical section, i will be the "next" of process j; otherwise, j sends the token to process i.

We have to check that:

(a) The request of process i is transmitted to a process j for which "father= nil," within a finite delay. This will be proved in Lemma 2 in which we use the fact there are no circuits (lemma 1).
(b) If j has not requested entry to a critical section, j holds the token (Lemma 3).
(c) If j has requested the critical section, i becomes the "next" of j, and that fact will allow i to obtain the token within a finite delay. That will be proved in Lemma 6 and we shall use the structure of the queue (Lemmas 4 and 5).

The variables "father" and "next" are local to each process. But if we take the overall viewpoint for the whole distributed system, they can be seen as mappings of X into $X \cup \{\text{nil}\}$ (X is the set of processes).

LEMMA 1. *The following properties are satisfied:*

(1) *The mapping "father" constitutes a forest (set of rooted trees).*
(2) *This set is reduced to a single rooted tree if no request message is in transit between two processes.*

Proof. Points (1) and (2) are true in the initial situation. Let us suppose they are true at some instant.

We assume that i invokes the critical section and let us consider the use of the procedure **Request_C_S**. If i is a root, there is no modification made to the forest; otherwise, a rooted tree (or a forest) is deconnected from the rest. The number of rooted trees is increased by 1. Let us examine the event of receiving "**Req**(k)."

When a message arrives at a root process j, the new value of "father" is the requesting process. j is cut off from the rooted tree in which it was and is attached (with its children) to the rooted tree of the requesting process. We have a new forest. The number of rooted trees is unchanged.

When a message reaches a process j for which ("father= nil"), j loses its quality of root and is attached (with its children) to i. The number of rooted trees is decreased by 1.

LEMMA 2. *A request message is transmitted to a process for which "father=nil" within a finite delay.*

Proof. Let us consider that process i requests the critical section without having the token. A request is therefore sent from i toward the root of a tree.

Consider an instant during the transmission of this request, when it is in transit between k_1 and k_2. The arc from k_1 to k_2 has been deleted and the forest is cut into two parts: part A which the message comes from, and part B which the message goes to. No other request message can pass between A and B because no path can be created before the request message has arrived at k_2.

When the request message has arrived at k_2, if k_2 is not the root, the request message is sent from k_2 to k_3. Part A is increased and part B is decreased and there is always a cut between A and B.

Therefore, the request message can never again reach a node of A. We have proved a request message can never pass twice times by the same node; i.e., the number of nodes which the message passes by is less than n.

Otherwise, the delays of transmission are finite. We have proved the request message will reach a root within a finite delay.

LEMMA 3. *The following property is invariant:*

$$(\text{father} = \text{nil} \wedge \neg\ \text{requesting\_c\_s}) \Rightarrow \text{token\_present}.$$

Proof. This assertion is true initially, and it is preserved by every action of the algorithm.

LEMMA 4. *The following property is invariant:*

$$(\text{requesting\_c\_s} \wedge \text{father} \neq \text{nil}) \Leftrightarrow (\text{next} \neq \text{nil}).$$

Proof. This assertion is true initially, and it is preserved by every action of the algorithm.

DEFINITION. Let *WQ* be the set of the processes for which "requesting_c_s" or "token_present" is true.

LEMMA 5. The following properties are satisfied

(1) The mapping "next" structures *WQ* into a set of disjoint queues without repetition (some of them might be reduced to a single element).

(2) Processes at the head of a queue will be in one of the following situations:

—One has the token or the message "token ( )" is in transit toward it.

—The other heads have a request message concerning them in transit.

(3) The tails of the queues are roots.

Proof. There properties are true *in the initial situation*: there is only one queue reduced to a single process which holds the token.

Suppose they are *true at a given time*

In Fig. 3, the thick lines express a node is the next of another one. The thin lines express a request is in transit.

(a) When process *i* receives the message "**Req**(*k*)" and it cannot give the token to process *k*, a new "next" arc is created from *i* to *k* by the instruction "next:= *k*," the origin *i* of this edge is a root, since requests are transmitted to a root, and the extremity *k* is a requesting process. By (3) of Lemma 5, before such arc creation, the origin was an extremity of queue and under Lemma 4, we had "next= nil." No part of a queue is therefore ever deleted. A new queue is created or an arc is concatened to an old queue. The new process cannot already be in the queue. Then the new structure is a set of disjoint queues without repetition; (1) is satisfied.

If receiving the message "**Req**(*k*)" produces a concatenation of two queues, the head of the global queue is an head of one of the initial queues, (2) is satisfied. Otherwise, *i* was a queue constituted with a single process. It was a head and it was satisfying (2). It remains a head and it continues to satisfy (2).

In the first case, *k* is not a tail but it has lost its quality of root; (3) is satisfied. In the second case, the new tail is *k* and is a root.

(b) When a "next" arc is cancelled by the procedure **Release_C_S**, the new head of the queue is the process which has obtained or is going to obtain the token.

So, we have proved the lemma is satisfied after every modification of the queues.

LEMMA 6. If *i* is the "next" of another process, *i* will obtain the token within a finite delay.

Proof. The queue containing the process owning the token or about to obtain it is called privileged queue. If *i* is in the privileged queue, the procedure **Release_C_S** will give the token to it (the queue is FIFO).

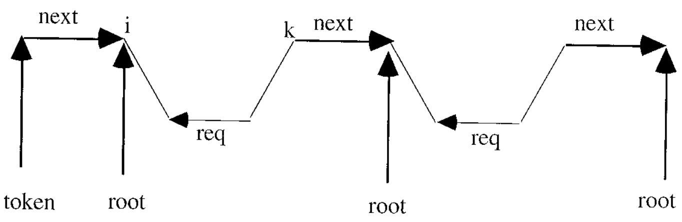

FIG. 3. Routing of requests and token.

Suppose *i* is in another queue and that the head of this queue is *j*. By part (2) of Lemma 5, the request message concerning *j* is in transit. By Lemma 2, this request message will arrive at a root *k* within a finite delay. *j* obtains the token from *k* or becomes the "next" of *k*. Now, *i* is perhaps in the privileged queue. If it is not the case, we are assured that the number of processes which are before *i* in its queue is increased.

We have proved that every time the request message concerning the head of the queue has arrived at a root, this number is increased or *i* finds itself in the privileged queue. Because the number of processes is finite and the queues are without repetition, *i* will necessarily find itself in the privileged queue within a finite delay.

THEOREM 1 (Absence of Deadlock and of Starvation). *If a process invokes the critical section, it will be able to enter it within a finite delay.*

Proof. Assume that a process *i* invokes entry to the critical section. If *i* is a root, by Lemma 3, it has the token and it may enter. Otherwise the request of *i* is transmitted to a root *j* within a finite delay (Lemma 2). Let us examine the two possible cases:

(a) *j* has not requested the critical section. By Lemma 3, *j* has the token and as a result of receiving of the message **Req**(*k*), it sends the token to *i*.

(b) *j* has requested the critical section. In this case, *i* becomes the "next" of *j*. By Lemma 6, *i* will enter the critical section.

4. PERFORMANCE

4.1. Average Number of Messages

4.1.1. Definitions

In this section, we shall set out to compute the average number of messages needed for a request to reach a root in a network of *n* processes.

This number can be measured on the rooted tree representing the mapping "father."

For instance, in Fig. 4:

If process 1 invokes the critical section, there are zero messages.

If process 2 invokes the critical section, there are two messages: one "req" and one "token."

If process 3 or process 4 invokes the critical section,

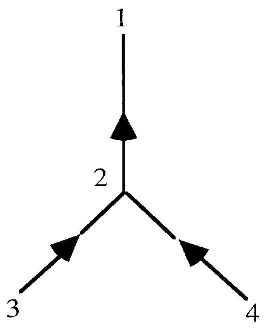

FIG. 4. Initial rooted tree for four processes.

there are three messages: two messages "req" and one message "token."

For root 1, the number of messages is 0. For the other processes, this number is the height of the requesting site (we consider the height is 1 at the root).

Let $N^i(a)$ be the number of elements of height i in the rooted tree a.

The average number of messages in a is

$$M(a) = \frac{1}{n}\sum_{i=2}^{n} i * N^i(a).$$

Suppose we have the rooted trees a, b, c, d.

For instance, in Fig. 5, we have

$$N^1(a) = 1 \quad N^2(a) = 3 \quad N^3(a) = 0 \quad M(a) = 1.5$$
$$N^1(b) = 1 \quad N^2(b) = 1 \quad N^3(b) = 2 \quad M(b) = 2.$$

Expression of the Efficiency. We have to compute the average of $M(a)$ over the different possible rooted trees. Let us specify what "average" means.

Let us denote by $T_i(a)$ the result of the transformation of tree a when the node i requests the critical section.

Example (Continued).

$$T_1(a) = a, \quad T_2(a) = b, \quad T_3(a) = c, \quad T_4(a) = d.$$

Let us define the probability π_x^j for the rooted tree x to appear at the step j:

—For the initial rooted tree r, $\pi_r^1 = 1$.
—otherwise $\pi_x^{j+1} = (1/n)\,\Sigma\,\{\pi_y^j / \exists\ i: x = T_i(y)\}$.

In the previous example,

$$\pi_a^1 = 1,$$
$$\pi_a^2 = \pi_b^2 = \pi_c^2 = \pi_d^2 = 0.25$$
$$\pi_a^3 = 0.25$$
$$\pi_b^3 = \pi_c^3 = \pi_d^3 = 0.125.$$

The average number of messages for n processes, at step j, is

$$M_n^j = \sum_x \pi_x^j * M(x)$$

The harmonic number H_i is defined as

$$\left(H_i = \sum_{j=1}^{i} \frac{1}{j};\ H_i \approxeq \log i + 0{,}577\right).$$

We shall prove that the average number of messages in a network of n processes is H_{n-1}; i.e., M_n^j converges to H_{n-1}.

4.1.2. Sketch of the Proof

• We shall express $M(a)$ as a function of the number of nodes of a height 2.

• We shall pass from the ordinary rooted trees to the oriented rooted trees and the Dyck words (Arnold *et al.* [1]).

• We shall prove the convergence of the M_n^j and compute its limit.

4.2. Number of Nodes of Height 2

Theorem 2.

$$M(a) = \frac{1}{n}\sum_{i=1}^{n} N^2(T_i(a)).$$

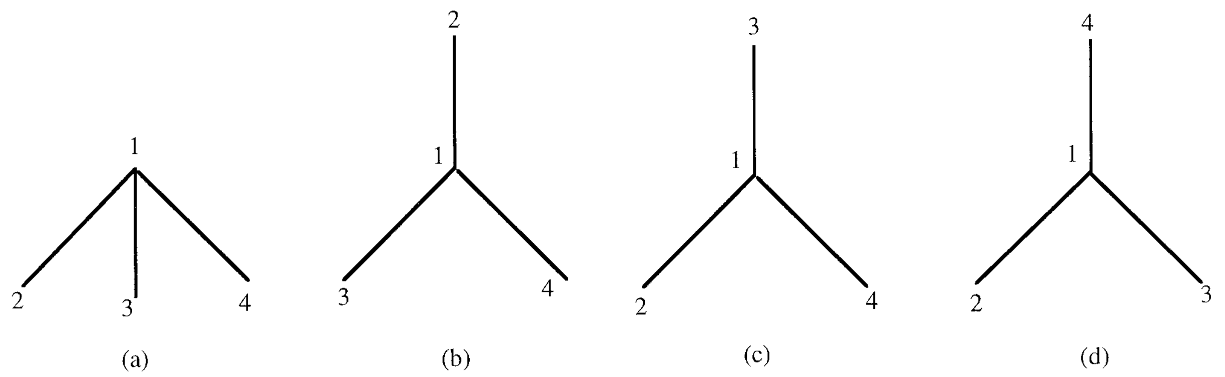

FIG. 5. Transformation of the rooted tree a.

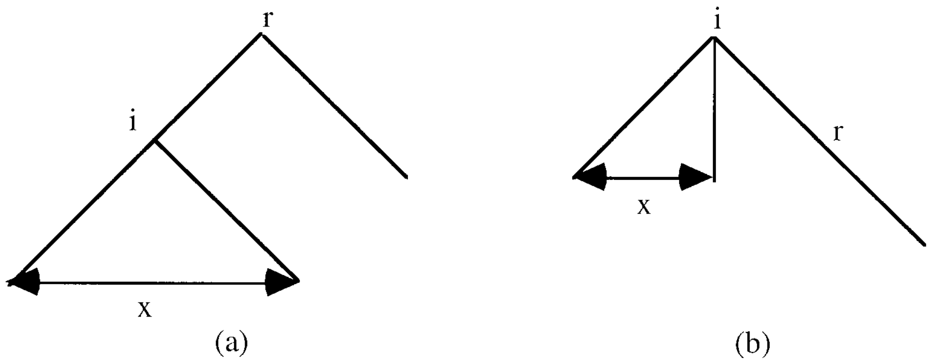

FIG. 6. Transformation when *i* is at height 2.

Proof. Consider a rooted tree a.

- If i is the root, $N^2(T_i(a)) = N^2(a)$.
- If i is at height 2, $N^2(T_i(a)) = x + 1$ where x is the number of sons of i

a can be schematized as the left part (Fig. 6a) and $T_i(a)$ can be schematized as the right part (Fig. 6b).

Thus

$$\sum_{i \text{ at height } 2} N^2(T_i(a)) = N^2(a) + N^3(a).$$

If i is at height 3 (see Fig. 7), $N^2(T_i(a)) = x + 2$, thus

$$\sum_{i \text{ at height } 3} N^2(T_i(a)) = 2N^3(a) + N^4(a).$$

More generally,

$$\sum_{i \text{ at height } p} N^2(T_i(a)) = (p-1)N^p(a) + N^{p+1}(a).$$

It becomes

$$\sum_{i=1}^{n} N^2(T_i(a)) = \sum_{i=2}^{n} i \cdot N^i(T_i(a)) = n \cdot M(a).$$

4.2.1. Transformation

Let us define the oriented rooted trees that we shall denote by ORT. An ORT is a rooted tree in which we do not take the labels of the nodes into account, and we take the order left to right in account. Thus, there is an exact correspondence between the ORT's and the Dyck words.

The transformation $T_i(x)$ has been defined on the labeled rooted tree x. We have to define it in the ORTs (the order left to right was not defined in $T_i(a)$ in the rooted trees). Let us take the transformation presented in Fig. 8.

4.2.2. Conservation of the Notion of M_n^j in the ORTs

Let us pass from the notion of rooted trees (RT) to the notion of unlabeled rooted trees denoted by URT, in which the order left to right is given up. If a is the URT corresponding to a RT a', we have $M(a) = M(a')$ because M does not depend on the labels of the nodes.

The probability π_a^j is the sum of the probabilities $\pi_{a'}^j$ for the RTs a' corresponding to a.

That means the value of the M_n^j is the same in the URTs and in the RTs.

Let us pass from the notion of URT to the notion of ORT. If a'' is an ORT corresponding to the URT a', M does not depend on the orientation left to right then

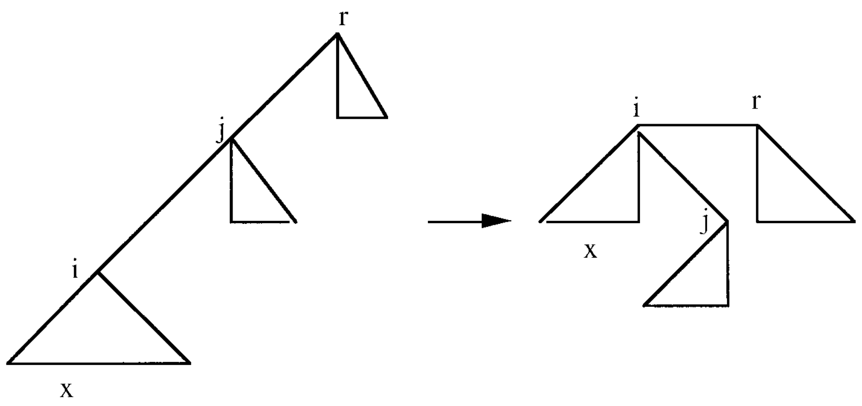

FIG. 7. Transformation when *i* is at height 3.

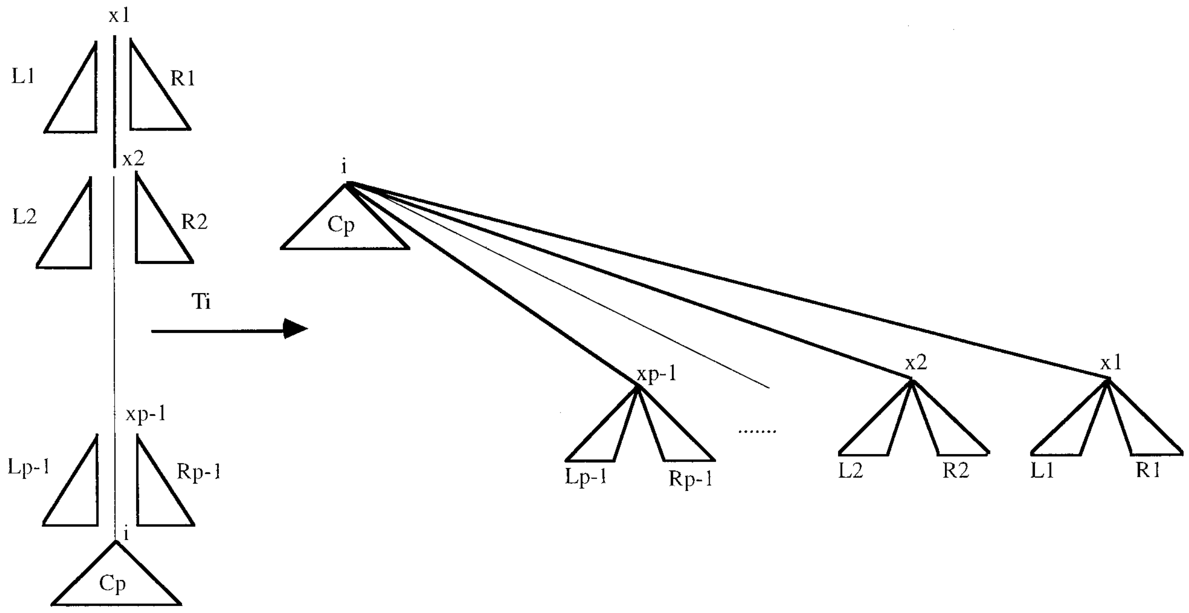

FIG. 8. Transformation of the oriented rooted tree.

$M(a) = M(a')$. The probability $\pi^j_{a'}$ is the sum of the probabilities $\pi^j_{a''}$ of the ORTs a'' corresponding to a'.

That means the value of the M^j_n is the same in the URTs and in the ORTs.

4.3. Convergence of the M^j_n

If there are m ORTs of n nodes, we can number them from 1 to m and consider the square matrix A, with m rows, m columns, where

$$A_{a,b} = \frac{1}{n}\operatorname{card}\{i/T_i(b) = a\}.$$

For example, in Fig. 9, $n = 4$ and we have five ORTs. Matrix A is the matrix of passage (see Fig. 10) between these ORTs. These matrices are Markov matrices.

Let us assume that the initial tree is the initial tree of our example. Consider the vector

$$I = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

A^jI will give the probabilities π^j_x of the ORTs x.

$M^j_n = M * A^jI$ where M is the line vector of the numbers of messages for every rooted tree.

We are interested in studying the convergence of $\lim_{i\to\infty} A^iI$.

Precisely, let us prove the convergence of $\lim_{i\to\infty} A^i$.

Varga [27] defines the graph of the matrix A by: the nodes are the indices of A, and there is an arc from i to j if $A_{i,j} \neq 0$. We will prove in Section 4.4.2 the strong connexity.

When p is great enough, there is none null element in

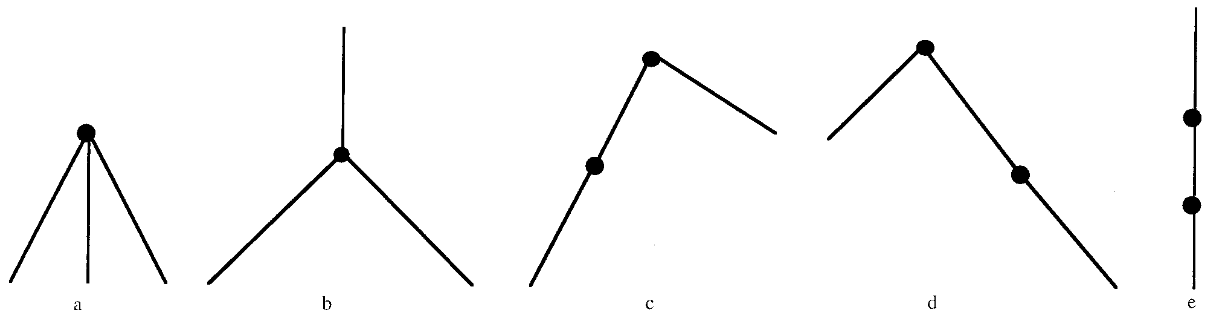

FIG. 9. All possible transformations of the trees of four nodes.

	a	b	c	d	e
a	$\frac{1}{4}$	$\frac{1}{4}$	0	0	$\frac{2}{4}$
b	$\frac{3}{4}$	$\frac{1}{4}$	0	0	0
c	0	$\frac{2}{4}$	$\frac{1}{4}$	0	$\frac{1}{4}$
d	0	0	$\frac{2}{4}$	$\frac{3}{4}$	0
e	0	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

FIG. 10. Matrix of passage.

A^p. If X is A^p it is easy to see that the difference between two elements of a same line is less in X^2 than in X. X^i tends to a matrix, all the columns of which are the same. Such a Markov matrix is a fixed point for the transformation $X \to X^2$. That assures the convergence of M_n^j.

4.4. Dyck Word and Oriented Rooted Trees

4.4.1. Definition

Let X be the alphabet $\{x, x'\}$ generated by the nonambiguous context-free grammar

$$D \to 1 \text{ (empty word)}$$
$$D \to DxDx'.$$

Every nonempty word w of this language is decomposed in a unique way $w = uxvx'$ where u and v are Dyck words.

If a is an ORT, we denote by $\mathcal{U}[a]$ the Dyck word corresponding to a, and by $\alpha[u]$ the ORT corresponding to u (see Fig. 11).

—If w is 1, $\alpha[w]$ is the ORT reduced to a single node.
—If $w = xx'$, $\alpha[w]$ is an ORT of two nodes.
—If $w = xx'xx'$, $\alpha[w]$ is an ORT of three nodes.

For the five trees presented in Fig. 5, we have

$$\mathcal{U}[a] = xx'xx'xx'$$

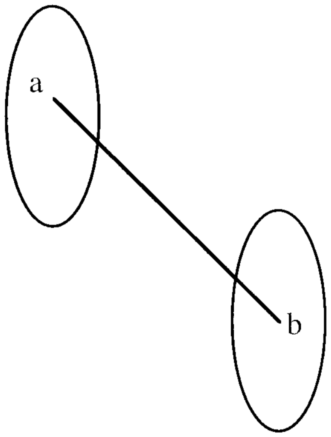

FIG. 11. Sketch of the rooted tree of Dyck word $\mathcal{U}[a]x\mathcal{U}[b]x'$.

$$\mathcal{U}[b] = xxx'xx'x'$$
$$\mathcal{U}[c] = xxx'x'xx'$$
$$\mathcal{U}[d] = xx'xxx'x'$$
$$\mathcal{U}[e] = xxxx'x'x'.$$

—If w is $uxvx'$, $\alpha[w]$ is the ORT composed with the ORTs $a = \alpha[u]$ and $b = \alpha[v]$, where the root of b is the right son of the root of a.

DEFINITION. Let D_n be the set of the Dyck words of length $2n$ (corresponding to the set $\mathcal{A}_n$ of the ORTs of $n + 1$ nodes).

4.4.2. Strong Connexity [15]

We will prove the strong connexity in the transposed matrix. Let us write $u \to v$, where $u, v \in D_n$ and $\alpha[v]$ is the result of a transformation of $\alpha[u]$.

We shall use the same arrow for a transformation and the transitive closure.

LEMMA 7. $v_1 \to v_2 \Rightarrow vxv_1x' \to v_2xvx'$.

The lemma is obvious in Fig. 12, with $l_1 = \alpha[v_1]$, $l_2 = \alpha[v_2]$, $l = \alpha[v]$.

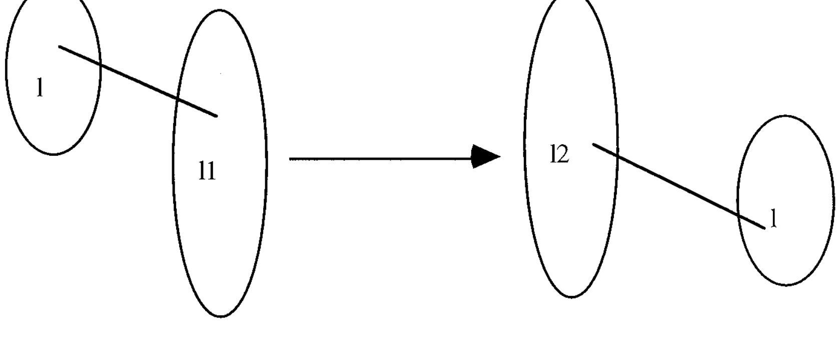

FIG. 12. Sketch of the transformation $vxv_1x' \to v_2xvx'$.

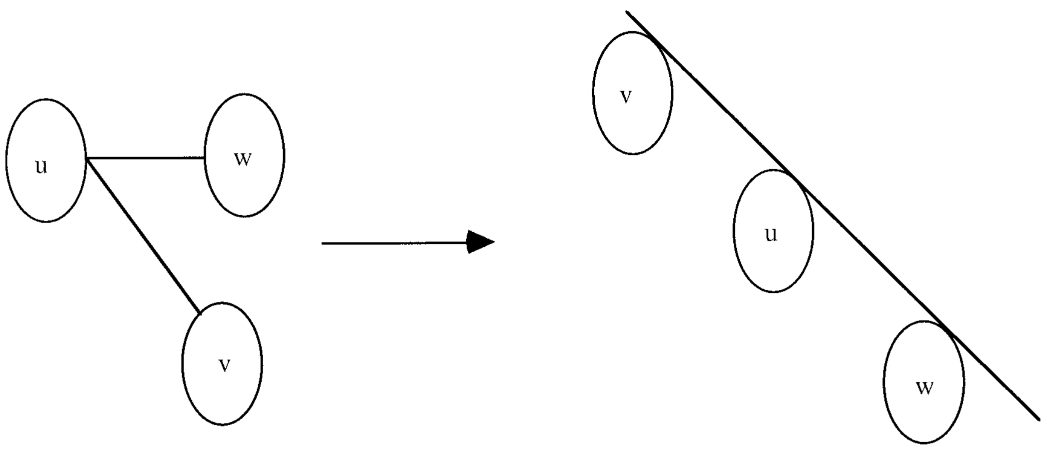

FIG. 13. Sketch of the transformation $uxvx'xwx' \to vxuxwx'x'$.

COROLLARY 1. $vxux' \to uxvx'$.

Proof. Apply Lemma 7 with $v_1 = v_2 = u$.

COROLLARY 2. $u_1 \to u_2$; $v_1 \to v_2 \Rightarrow u_1xv_1x' \to u_2xv_2x'$.

Proof. $v_1 \to v_2 \Rightarrow$ (by Lemma 7) $u_1xv_1x' \to v_2xu_1x'$. $u_1 \to u_2 \Rightarrow$ (by Lemma 7) $v_2xu_1x' \to u_2xv_2x'$.

LEMMA 8. $uxvx'xwx' \to vxuxwx'x'$.

Proof. This is immediate by the definition of the transformation (the new root is the father of v). The transformation can be observed in Fig. 13.

LEMMA 9. *Let $p, q < n + 1$. There exist $u \in D_p$, $v \in D_{n-p}$, $u' \in D_q$, $v' \in D_{n-q}$, such that $uxvx' \to u'xv'x'$.*

Proof. If $p = q$, then let us take $u = u'$, $v = v'$ and the lemma is true.

If $p > q$, then let us take $w \in D_{p-q-1}$, $u' \in D_q$, $v \in D_{n-p}$, and let $u = wxu'x' \in D_p$, $v' = wxvx' \in D_{n-q}$. By Lemma 8, we get $uxvx' = wxu'x'xvx' \to u'xwxvx'x' = u'xv'x'$

If $p < q$, then $n - p > n - q$, thus, as above, there exist $u \in D_p$, $v \in D_{n-p}$, $u' \in D_q$, $v' \in D_{n-q}$ such that $vxux' \to v'xu'x'$, and by Corollary 1, $uxvx' \to u'xv'x'$.

LEMMA 10. *D_{n+1} is strongly connected.*

Proof. We establish the proof by induction. D_0, D_1, and D_2 are strongly connected. Let us consider $n \geq 2$ and assume that D_i is strongly connected for each $i \leq n$. Let $uxvx'$, $u'xv'x' \in D_{n+1}$ with $u \in D_p$, $v \in D_{n-p}$, $u' \in D_q$, $v' \in D_{n-q}$. We want to prove $uxvx' \to u'xv'x'$.

By Lemma 9, there exist $w_1 \in D_p$, $w_2 \in D_{n-p}$, $w_3 \in D_q$, $w_4 \in D_{n-q}$, such that $w_1xw_2x' \to w_3xw_4x'$. By the induction hypothesis, $u \to w_1$, $v \to w_2$, $w_3 \to u'$, $w_4 \to v'$, and by Corollary 2, we get the result.

We are now assured that the vector of probabilities converges. Now the limit of M_n^j exists. The limit of π_a^i is denoted by π_a and the limit of M_n^j by M_n.

4.4.3. Average Number of Nodes of Height 2

THEOREM 3. *M_n is the average number of nodes of height 2.*

Proof. We have proved in Theorem 2 that

$$\frac{1}{n}\sum_{i=1}^{n} N^2(T_i(a)) = M(a).$$

At the step j, we have

$$\frac{1}{n}\sum_{a\in\mathcal{A}_{n-1}} \pi_a^j \sum_{i=1}^{n} N^2(T_i(a)) = \sum_{a\in\mathcal{A}_{n-1}} \pi_a^j M(a)$$

by definition of

$$\pi_b^{j+1} = \frac{1}{n}\sum \{\pi_a^j / \exists i \quad b = T_i(a)\}.$$

It becomes

$$\sum_{b\in\mathcal{A}_{n-1}} \pi_b^{j+1} N^2(b) = \sum_{a\in\mathcal{A}_{n-1}} \pi_a^j M(a).$$

The limit of the second member exists when $j \to \infty$; then the limit of the first member exists: it will be denoted N_n^2.

We have $M_n = N_n^2$.

The remainder of the proof consists of proving $N_n^2 = H_{n-1}$

4.4.4. Mappings in the Dyck Words

DEFINITIONS. We want to define a mapping μ so that $\mu(u)$ gives the sum of all the transformed words of u. That means we are going to work in the set F of the formal series, the support of which is the set D in Dyck words [7, 8, 10].

Consider the mappings μ, ν, and Id from F to F:

$$\mu = \nu + \mathrm{Id}$$
$$\forall d \in D, \quad \mathrm{Id}\,(d) = d$$
$$\nu(1) = 0 \quad (1 \text{ is the empty word})$$
$$\nu(uxvx') = \nu(u)\,[\,]\,(xvx') + \mu(v)xux'.$$

$[\,]$ is a mapping from $F * D$ to F so that:

$$0[\,]g = 0$$
$$f_1xf_2x'\,[\,]g = f_1xf_2gx'$$

Furthermore, μ, ν, and Id are linear mappings.

Intuitively, μ contains Id for the case when the requesting process is the root and ν for the other cases. The first term $\nu(u)\,[\,]\,(xvx')$ corresponds to the case when the requesting process is in u and the second one $\mu(v)\,xux'$ to the case when the requesting process is in v.

$[\,]$ corresponds to attaching g at the extreme right of height 2.

The following lemma proves that the definition of μ corresponds to its aim.

LEMMA 11. *For every word u of D_n, we have $\mu(u) = \Sigma_{i\in\alpha[u]}\,\mathscr{U}[T_i(\alpha[u])]$.*

Proof. We prove by induction that

$$\nu(u) = \sum_{i\in\alpha[u],i\neq\text{root}} \mathscr{U}[T_i(\alpha[u])].$$

It is obvious if the length of u is 0 (the ORT has one single node).

Let us assume that it is true if the length of u is less than $2n+2$ and let us consider that the length of u is $2n+2$. Then $u = vxwx'$ where the lengths of v and of w are at most equal to $2n$.

Let us define

$$A = \sum_{i\in\alpha[v],i\neq\text{root}} \mathscr{U}[T_i(\alpha[u])]$$
$$B = \sum_{i\in\alpha[w]} \mathscr{U}[T_i(\alpha[u])].$$

By the definitions of T_i and $[\,]$,

$$A = \sum_{i\in\alpha[v],i\neq\text{root}} \mathscr{U}[T_i(\alpha[v])]\,[\,]xwx'$$

and by induction

$$A = \nu(v)\,[\,]xwx'$$
$$B = \mathscr{U}[\alpha[w]]xvx' + \sum_{i\in\alpha[w],i\neq\text{root}} \mathscr{U}[T_i(\alpha[w])]xvx'$$
$$= (w + \nu(w))xvx' = \mu(w)xvx'$$

$$A + B = \nu(v)\,[\,]xwx' + \mu(w)xvx'$$
$$= \nu(vxwx') = \nu(u).$$

4.4.5. Family of Formal Series

DEFINITIONS. p_n is the family of the formal series defined by

$$p_0 = 1$$
$$p_{n+1} = \frac{1}{n+1}\sum_{k=0}^{n} p_kxp_{n-k}x'.$$

p_n is a formal series on the support of Dyck words. We will prove that p_n is a fixed point of the operator $\mu/(n+1)$.

THEOREM 4.

$$\frac{1}{n+1}\mu(p_n) = p_n.$$

Proof. We will prove by induction that $\nu(p_n) = np_n$.

It is true for $n = 0$

Suppose it is true for every value less than or equal to n. We have:

$$\nu(p_{n+1}) = A + B \text{ with}$$
$$A = \frac{1}{n+1}\sum_{k=0}^{n} \nu(p_k)[\,]xp_{n-k}x'$$
$$B = \frac{1}{n+1}\sum_{k=0}^{n} \mu(p_{n-k})xp_kx'$$

By the induction hypothesis we set

$$A = \frac{1}{n+1}\sum_{k=0}^{n} kp_k[\,]xp_{n-k}x'$$
$$= \frac{1}{n+1}\sum_{k=1}^{n}\sum_{i=0}^{k-1} p_ixp_{k-i-1}xp_{n-k}x'x'.$$

Let us permute the Σ's and take $j = k - i - 1$. We obtain

$$A = \frac{1}{n+1}\sum_{i=0}^{n-1} p_ix\left(\sum_{j=0}^{n-i-1} p_jxp_{n-j-i-1}x'\right)x'$$
$$= \frac{1}{n+1}\sum_{i=0}^{n-1} p_ix((n-i)p_{n-i})x'$$
$$= \frac{1}{n+1}\sum_{k=0}^{n} kp_{n-k}xp_kx' \text{ by taking } k = n - i,$$

$$B = \frac{1}{n+1}\sum_{k=0}^{n} (n-k+1)p_{n-k}xp_kx'$$

thus

$$\nu(p_{n+1}) = A + B = \sum_{k=0}^{n} p_{n-k}xp_x x' = (n+1)p_{n+1}.$$

COROLLARY 1. $p_n = \sum_{a\in\mathscr{A}_n} \pi_a \mathscr{U}[a]$ (ORT of $n+1$ nodes).

In fact, p_n is a formal series on the support of Dyck words. The array of the coefficients of the Dyck words in p_n is the array of the limit probabilities π because π is the unique fixed point of the transformation matrix and the operator $\mu/(n+1)$ corresponds to this transformation.

COROLLARY 2. *If w is a Dyck word of D_n, decomposed in $w = uxvx'$, the probability of the ORT associated is* $\pi_{\alpha[w]} = (1/n)\pi_{\alpha[u]}\pi_{\alpha[v]}$.

That comes by identifying the expression of p_n in its definition and the expression of p_n in Corollary 1.

DEFINITION. We define the series

$$r_0 = 1; \quad r_{n+1} = \frac{1}{n+1}\sum_{k=0}^{n} y r_k x p_{n-k} x'.$$

LEMMA 12.

$$r_n = \sum_{a\in\mathscr{A}_n} \pi_a y^{N^2(a)}\mathscr{U}[a]$$

Proof. It is true for $n = 0$

Suppose it is true for every value less than $n+1$,

$$r_{n+1} = \frac{1}{n+1}\sum_{k=0}^{n} y\left(\sum_{a\in\mathscr{A}_k}\pi_a y^{N^2(a)}\mathscr{U}[a]\right)x\left(\sum_{b\in\mathscr{A}_{n-k}}\pi_b\mathscr{U}[b]\right)x'$$

$$= \sum_{k=0}^{n}\sum_{a\in\mathscr{A}_k, b\in\mathscr{A}_{n-k}} y^{N^2(a)+1}\frac{\pi_a\pi_b}{n+1}\mathscr{U}[a]x\mathscr{U}[b]x'.$$

Let $c = \alpha(\mathscr{U}[a]x\mathscr{U}[b]x')$.

c has already been represented in Fig. 11. Then

$$\pi_c = \frac{\pi_a\pi_b}{n+1} \quad\text{and}\quad N^2(c) = 1 + N^2(a).$$

Moreover, $\mathscr{A}_{n+1} = \cup_k \{\alpha(\mathscr{U}[a]x\mathscr{U}[b]x')/a \in \mathscr{A}_k,\ b \in \mathscr{A}_{n-k}\}$. Therefore

$$r_{n+1} = \sum_{c\in\mathscr{A}_{n+1}} y^{N^2(c)}\pi_c\mathscr{U}[c].$$

THEOREM 5. $M_n = H_{n-1}$

Proof. After Theorem 3, it remains to prove

$$\sum_{a\in\mathscr{A}_n}\pi_a N^2(a) = H_{n-1}.$$

It is true for $n \leq 2$.

Suppose it is true for every value less than n.

Let us take the derivative $\partial r_n/\partial y$ of r_n: by Lemma 12, $M_{n+1} = (\partial r_n/\partial y)$ with $x = x' = y = 1$. Let us take the definition of r_n:

$$r_n = \frac{1}{n}\sum_{k=0}^{n-1} y r_k x p_{n-1-k}x';$$

$$\frac{\partial r_n}{\partial y} = \frac{1}{n}\sum_{k=0}^{n-1} r_k x p_{n-1-k}x' + \frac{1}{n}\sum_{k=0}^{n-1} y\frac{\partial r_k}{\partial y}xp_{n-1-k}x'.$$

If we set $x = x' = y = 1$, we obtain $r_k = p_k = 1$, then

$$M_{n+1} = 1 + \frac{1}{n}\sum_{k=0}^{n-1} M_{k+1}.$$

It is a known result that $\sum_{k=0}^{n-1} H_k = n(H_n - 1)$. The only solution to the recurrence equation is $M_n = H_{n-1}$

6. CONCLUSION

The algorithm we have described has the particular advantage of being very efficient. Furthermore, it uses distributed data structures instead of a control structure such as a logical clock.

It is possible to generalize it to an arbitrary network [11]. As long as the network is connected, a logical tree structure can always be defined. However, if it is not complete, it cannot be so easily deformed. The number of messages depends of the geometry: there cannot be a more suitable formula than H_{n-1}. We have already examined some geometries and studied the complexity.

ACKNOWLEDGMENTS

We are grateful to Mr. C. Lavault, M. Raynal, S. Dulucq, M. P. Delest, M. Maekawa, A. Agrawala, and the anonymous referees.

REFERENCES

1. A. Arnold, M. P. Delest, and S. Dulucq, Complexité moyenne de l'algorithme d'exclusion mutuelle de Naimi-Tréhel. Tech. Rep., UER de mathématique et informatique, Université de Bordeaux-1, 1987.
2. C. Berge, *Théorie des graphes et ses applications*, Dunod, Paris, 1967.
3. J. Berstel, *Transductions and Context-Free Languages*. Teubner, Stuttgart, 1979.
4. O. Carvalho and G. Roucairol, On mutual exclusion in computer networks, *Comm. ACM* **26,** 2 (Feb. 1983), 146–147.
5. A. Bouabdallah and M. Tréhel, A characterization of the dynamical ascending routing. *Proc. of the Workshop on the Future Trends of Distributed Computing Systems in the 1990's*, Hong Kong, Sept. 1988, pp. 237–244.
6. M. Chandy, J. Misra, The drinking philosopher problem. *ACM Trans. Parallel Languages and Systems* **6,** 4 (Oct. 1984), 632–646.
7. L. Chottin, Etude syntaxique de certains langages solutions d'équations avec opérateurs. *Theor. Comput. Sci.* **5** (1977), 51–84.
8. R. Cori, J. Richard, Enumération des graphes planaires à l'aide des séries formelles en variables non commutatives. *Discrete Math.* **2** (1972).

9. E. W. Dijkstra, Self-stabilizing systems in spite of distributed control. *Comm. ACM* **17,** 11 (Nov. 1974), 643–644.
10. S. Dulucq, Equations avec opérateurs: Un outil combinatoire. Thése de 3ème cycle, Bordeaux, 1981.
11. M. Fouzi, M. Trehel, A distributed algorithm for mutual exclusion using distributed structures in an arbitrary network. IFIP, Network Information Processing System, Sofia, May 1988.
12. H. Garcia-Molina, Elections in a distributed computing system. *IEEE Trans. Comm.* **C-31** (Jan. 1982), 48–59.
13. D. Ginat and A. U. Shankar, An assertional proof of correctness and serializability of a distributed mutual exclusion algorithm based on path reversal. UNIACS-TR-88-65 CS-TR-2104, (Sept. 1988).
14. D. Ginat, D. D. Sleator, R. E. Tarjan, A tight amortized bound for path reversal. *Inform. Process. Lett.* **31** (1989), 3–5.
15. J. Hardouin-Duparc, Maxima limites d'une caractéristique de certaines transformations d'arborescences. Tech. Rep., Lab. Inf., Univ. de Besançon, France, May 1987.
16. D. E. Knuth, *The Art of Computer Programming*, Vol. 1. Addison–Wesley, Reading, MA, 1973.
17. L. Lamport, Time, clocks and the ordering of events in a distributed system. *Comm. ACM* **21,** 7 (1978), 558–565.
18. G. Le Lann, Distributed systems, towards a formal approach. IFIP Congress, Toronto, Aug. 1977, pp. 155–160.
19. M. Maekawa, "A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Systems* **3,** 2 (May 1985), 145–159.
20. M. Naimi Une structure arborescente pour une classe d'algorithmes d'exclusion mutuelle. Ph.D. Thesis University of Besançon, France, Mar. 1987.
21. G. Ricart and A. Agrawala, An optimal algorithm for mutual exclusion in computer networks. *Comm. ACM* **24,** 1 (Jan. 1981), 9–17.
22. G. Ricart and A. Agrawala, Author's response to "On mutual exclusion in computer networks" by Carvalho and Roucairol. *Comm. ACM* **26,** 2 (Feb. 1983). 147–148.
23. I. Suzuki and T. Kasami, An optimality theory for mutual exclusion networks, *Proc. of the 3rd Int. Conf. on Distributed Computing Systems*. Miami, Oct. 1982, pp. 365–370.

Received April 18, 1992; accepted June 8, 1995

24. M. Tréhel, M. Naimi, Un algorithme distribué d'exclusion mutuelle en Log(n). *TSI* **6,** 2 (1987), 141–150.
25. M. Tréhel, M. Naimi, A distributed algorithm for mutual exclusion based on data structures and fault tolerance. *6th Annual International Phoenix Conference on Computers and Communications*, Scottsdale, AZ, Feb. 1987, pp. 35–39.
26. M. Tréhel, Propriétés du nombre harmonique H_n liées à des transformations d'arborescences. Tech. Rep., Lab. Inf., Univ. de Besançon, France, Sep. 1986.
27. R. S. Varga, *Matrix Iterative Analysis*. Prentice–Hall, Englewood Cliffs, NJ, 1962.

MOHAMED NAIMI is Maître de Conférences for computer science at the Université de Besançon. He received an Habilitation de Diriger la Recherche in computer science from the Université de Dijon in 1994 for work on distributed algorithms. He is working on distributed systems: mutual exclusion problems, ordering of communications, fault tolerance, multiagents, etc. He is the author of a book of methods of programming entitled *Introduction à la Programmation* (Hermes, Paris, 1992).

MICHEL TREHEL received a Doctorat d'Etat from Grenoble in 1972 for work on tree-structured files and their algebraic properties. He later worked on inverted multi-index files, on nonnumeric sequential and (later) parallel algorithms, mutual exclusion, and modeling of load balancing by queue theory. He has been the scientific advisor for 13 theses, and has been Chairman of two conferences. He is now a professor at the Université de Besançon.

ANDRÉ ARNOLD is Professor of Computer Science at the Université de Bordeaux I and is the Director of LaBRI, the research laboratory for computer science at this university. He received a Doctorat d'Etat from the Université de Lille in 1977. He is now working on temporal logics, mu-calculus, and transition systems as formal tools for modeling and verifying computerized systems. He is the author or coauthor of two books on this topic: *Finite Transition Systems* (Prentice–Hall, Englewood Cliffs, NJ, 1994) and *Construction and Analysis of Transition Systems with MEC* (World Scientific, Singapore, 1994).