

Prime numbers as a tool to design distributed algorithms

Michel Raynal

▶ **To cite this version:**

Michel Raynal. Prime numbers as a tool to design distributed algorithms. [Research Report] RR-1001, INRIA. 1989. inria-00075558

HAL Id: inria-00075558
https://inria.hal.science/inria-00075558

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1001

Programme 1

PRIME NUMBERS AS A TOOL TO DESIGN DISTRIBUTED ALGORITHMS

Michel RAYNAL

Mars 1989

RR_1001

PRIME NUMBERS AS A TOOL TO DESIGN DISTRIBUTED ALGORITHMS

Les nombres premiers : un outil pour la conception des algorithms répartis

Publication Interne n° 458 - 14 Pages - Février 1989

Michel RAYNAL

IRISA
Campus de Beaulieu
35042 Rennes Cedex
- FRANCE -
E-mail: raynal@irisa.fr

Abstract

Prime numbers are investigated as a tool to design distributed control algorithms. One of major drawbacks in designing such algorithms lies in the inability for one or several processes (or even for an external observer) to catch instantaneously some part of the global state of the system. It is shown in this paper how, in some cases, prime numbers can be used to make distributed observations allowing to make consistent decisions. Two very different distributed algorithms are produced as an illustration of the proposed tool (a mutual exclusion and a termination detection algorithms).

Résumé :

L'impossibilité d'obtenir un état global de manière instantanée constitue l'un des problèmes les plus délicats à résoudre lors de la conception d'un algorithme de contrôle réparti. Des outils algorithmiques particuliers tels que les horloges logiques ou les marqueurs ont été introduit afin de pallier cette incapacité. On propose ici un autre outil pour résoudre ces problèmes : les nombres premiers. Deux algorithms distribués (l'un réalisant une exclusion mutuelle, l'autre détectant la terminaison répartie) sont exhibés pour illustrer l'outil proposé.

PAPIER RÉCUPÉRÉ ET RECYCLÉ

PRIME NUMBERS AS A TOOL TO DESIGN DISTRIBUTED ALGORITHMS

Michel RAYNAL

IRISA
Campus de Beaulieu
35042 Rennes Cedex
- FRANCE -
E-mail: raynal@irisa.fr

Abstract

Prime numbers are investigated as a tool to design distributed control algorithms. One of major drawbacks in designing such algorithms lies in the inability for one or several processes (or even for an external observer) to catch instantaneously some part of the global state of the system. It is shown in this paper how, in some cases, prime numbers can be used to make distributed observations allowing to make consistent decisions. Two very different distributed algorithms are produced as an illustration of the proposed tool (a mutual exclusion and a termination detection algorithms).

1 INTRODUCTION

One of the more basic problems found in asynchronous distributed computations and distributed systems lies in the inability for one or several processes the system is made of (or even for an observer external to the system) to take instantaneously a consistent global snapshot[LEL77, FRA80, CL85, RAY88]. Among others causes, this inability comes from the not-bounded communication delays for messages. So when designing a distributed algorithm needing some kind of global knowledge, one has to introduce some devices allowing to get a consistent view of the part of the system he needs. Logical clocks introduced by Lamport [LAM78] to distribute

a state machine or markers introduced by Misra [MIS83] and used on fifo channels by Chandy and Lamport [CL85] are such devices.

This paper studies another such a device: prime numbers; each process of the system is endowed with a (different) prime number. As we will see the unicity of the decomposition of an integer in a product of prime numbers is a property which allows one or several processes (or an external observer) to take consistent decisions concerning some aspects of the global state of the system.

The §2 states the hypotheses. Then two simple and elegant distributed control algorithms are produced: a mutual exclusion algorithm (§3) and a termination detection algorithm (§4). Mutual exclusion and termination detection are two paradigms of distributed control. Our aim is not to obtain efficient algorithms from a number of messages point of view but to investigate and to show that prime numbers and their properties can be a useful tool when designing some kind of algorithms in a distributed context.

2 HYPOTHESES

We consider a distributed system made of n sites: $P_0, P_1, .., P_{n-1}$; one and only one process is associated to each site. These processes are linked by a connection network; there is no central memory and processes communicate solely by means of messages with unpredictable (and non-zero) transmission delays; communication channels are bidirectionnal.

Each process P_i is endowed with an attribute a_i; these attributes are natural integers, different from 1, and prime two by two.

3 A MUTUAL EXCLUSION ALGORITHM

3.1 The algorithm

The network is supposed totally connected. Each P_i owns a variable x_i initialized to a_i except for one x_k initialized to 1. Each process P_i can read any variable x_j. (This can be easily implemented by a request message from P_i to P_j followed by a reply message from P_j to P_i).

$$Let\ Q\ be\ \prod_{i=1}^{n} a_i$$

Protocols to enter and exit the critical section are the following ones for each P_i:

Wait $\prod_{m=1}^{n}(x_m = Q/a_i)$;
$<$ *critical section* $>$;
$x_i := x_i * a_i/a_j$;

In the exit part of the protocol the effect of the multiplication by a_i is the loss by P_i of the privilege to use the critical section; the effect of the division by a_j is to give P_j the privilege. If $j = (i+1)$ *mod* n the privilege to use the critical section turns around the logical ring: P_k, P_{k+1}, .., P_0, P_1, .., P_k, as in [LEL77, DIJ74].

Remark: The wait operation is implemented by having P_i to ask the $P_j s$ the values of their variables x_j from time to time until the condition becomes true.

3.2 Proof

The privilege to enter the critical section is expressed by some configuration of the state variables x_i. It is very easy to see that initially only the process P_k (the one with x_k initialized to 1) can enter the critical section (without loss of generality we suppose $k = 0$).

It is easy to show that in a centralized context (the x_i are stored in a central memory and the readings get always the last values of the variables) the privilege turns around the logical ring iff $a_i \neq a_j$, $\forall\, i \neq j$.

In a distributed context a problem comes from the arbitrary (but finite) delays of the request and reply messages. Two processes P_j and P_k can use at the same time for the value of some x_i, two different values the variable x_i had previously. Consequently we have to prove that, despite these potentially mutually inconsistent readings, mutual exclusion is always guaranted (safety property) and that the privilege turns around the logical ring (liveness property).

Safety

Let $x_k(i)$ be the last value obtained by P_i as value of x_k; it is possible to have $x_k(i) \neq x_k$ for $k \neq i$ (but $x_i(i) = x_i$ at any time).

Let us suppose that the privilege has completed t full turns ($t \geq 0$) and that in the turn $t+1$ the privilege is owned by P_i . The state variables have the following values ($\forall\, t \geq 0, \forall i$):

$$x_0 = \frac{a_0^{t+1}}{a_1^{t+1}},\ x_1 = \frac{a_1^{t+2}}{a_2^{t+1}},\ \ldots\ x_{i-1} = \frac{a_{i-1}^{t+2}}{a_i^{t+1}},\ x_i = \frac{a_i^{t+1}}{a_{i+1}^{t}},\ \ldots\ x_{n-1} = \frac{a_{n-1}^{t+1}}{a_0^{t}}$$

Let us consider P_m ($m \neq i$) and suppose P_m can obtain the privilege, ie:

$$\prod_{k=0}^{n-1} x_k(m) = Q/a_m \tag{1}$$

with:

$$1)\ x_0(m) = \frac{a_0^{\alpha_0}}{a_m^{\alpha_0}},\ x_1(m) = \frac{a_1^{\alpha_1+1}}{a_2^{\alpha_1}},\ \ldots\ x_{m-1}(m) = \frac{a_{m-1}^{\alpha_{m-1}+1}}{a_m^{\alpha_{m-1}}}$$

$$2)\ x_m(m) = x_m = \frac{a_m^{t+2}}{a_{m+1}^{t+1}} \quad if\ m < i,\ \frac{a_m^{t+1}}{a_{m+1}^{t}} \quad if\ m > i$$

$$3)\ x_{m+1}(m) = \frac{a_{m+1}^{\alpha_{m+1}+1}}{a_{m+2}^{\alpha_{m+1}}},\ \ldots\ x_{n-1}(m) = \frac{a_{n-1}^{\alpha_{n-1}+1}}{a_0^{\alpha_{n-1}}}$$

the exponents are such that:

$$\begin{gathered} 0 \leq \alpha_j \leq t+1\ for\ 0 \leq j \leq i-1 \\ 0 \leq \alpha_j \leq t\ for\ i \leq j \leq n-1 \\ \alpha_m = t+1\ if\ m < i,\ t\ if\ m > i \end{gathered} \tag{2}$$

Two cases have to be examined according to the case $m < i$ or $m > i$. We consider the first one.

$$(1) \wedge m < i \Rightarrow \frac{a_0^{\alpha_0}}{a_1^{\alpha_0}} \ldots \frac{a_{m-1}^{\alpha_{m-1}+1}}{a_m^{\alpha_{m-1}}} \ldots \frac{a_m^{t+2}}{a_{m+1}^{t+1}} \ldots \frac{a_{m+1}^{\alpha_{m+1}+1}}{a_{m+2}^{\alpha_1+1}} \ldots = a_0 \ldots a_{m-1} a_{m+1} .. a_{n-1}$$

that is to say:

$$a_0^{\alpha_0}\ a_1^{\alpha_1+1} \ldots a_{m-1}^{\alpha_{m-1}+1} a_m^{t+2} \ldots a_{n-1}^{\alpha_{n-1}+1} = a_0^{\alpha_{n-1}+1}\ a_1^{\alpha_0+1} \ldots a_m^{\alpha_{m-1}} \ldots a_{m+1}^{t+2} \ldots a_{n-1}^{\alpha_{n-2}+1} \tag{3}$$

Moreover:

$$m < i \wedge (2) \Rightarrow \alpha_{m-1} \leq t+1 \quad ie \quad \alpha_{m-1} < t+2 \tag{4}$$

$$(2) \wedge (3) \wedge (4) \Rightarrow a_m^{t+2-\alpha_{m-1}}\ divides\ a_0^{\alpha_{n-1}+1} \ldots a_{m-1}^{\alpha_{m-2}+1} a_{m+1}^{t+2} \ldots a_{n-1}^{\alpha_{n-2}+1}$$

A simular reasonning hold for the case $m > i$.

Summing up:

$$\forall\, m \neq i :\ \prod x_k(m) = Q/a_m \Rightarrow a_m\ divides\ a\ product\ of\ a_0 .. a_{m-1} a_{m+1} .. a_{n-1}$$

Consequently as the a_k are prime two by two P_m cannot enter the critical section if it is the P_i's turn.

Liveness

If it is the P_i 's turn to enter the critical section it will enter it as soon as it has received the last values of the variable x_m. (Recall the delays are finite).

4 A TERMINATION DETECTION ALGORITHM

4.1 Hypotheses

The termination detection problem is a paradigm of the distributed control [FRA80, CL85, MAT87, LY87]. A process P_i can become passive at any time; a passive process can only be made active by receiving a message; an active process can send messages to its neigbhours. We suppose without loss of generality that a process is passive when it receives a message and that a process does not send messages to itself. The communication network is totally connected.

Detect termination consists in detecting a state of the distributed application in which all the processes are passive and all the channels are empty.

4.2 The algorithm

Conceptually an observer OBS is in charge of the termination detection. It can be implemented in several ways: it can be a particular site, it can be duplicated on several sites, it can be implemented by a circulating memory like a token travelling along a ring, a tree, etc [MAT87, HJPR87, RAY88].

At a conceptual level the implementation of the observer does not matter. The only hypothese we assume is that control messages OBS receives from each process are received in their sending order.

When a process P_i becomes passive it sends to OBS a control message. Let us suppose that, when it was active for the last time, P_i sent p application messages to P_j and q to P_k. Becoming passive it sends OBS the following control message:

$$ctl(a_j^p * a_k^q / a_i)$$

The observer OBS in endowed with a variable t. It is initialized to the following value:

$$t \ init \ to \ \prod_{i \in I} a_i$$

(Where I is the set of initially active processes). When it receives a control message $ctl(n/d)$ the observer OBS updates t in the following way:

$$t := t * n/d$$

When $t = 1$ the observer claims the application is terminated: all the processes are passive and the channels are empty.

4.3 Proof

The proof consists in showing:

- that if the application is terminated then eventually the observer will detect it (liveness),
- and that if the termination is detected then the application is actually terminated (safety).

Liveness

When the application is terminated on the one hand each message m sent to a P_i has been received and processed, and on the other hand all the processes are passive. Consequently if q messages have been sent to P_i OBS will receive a_i^q in total (by control messages) as the senders of these messages are passive. When P_i became definitively passive (after having received these q messages) it has sent the observer in total $1/a_i^{q+1}$ if it was initially active (or $1/a_i^q$ if it was initialy passive). It follows that $t = 1$ when OBS has received all the control messages.

Safety

The value of t represents some global snapshot of the system from the point of view of its activity. The global snapshot S (global state) associated to t is the following one. The state of a process in t is [LY87]:

- its local state when it sent the last control message received by OBS,
- the sets of messages it has sent to each of its neigbhours and the sets of messages it has received from them up to the last control message it sent and which has been received by OBS.

So at any time (each "*" stands for any non negative integer):

$$\frac{a_0^* \; a_1^* \ldots a_{i-1}^* \; a_{i+1}^* \ldots a_{n-1}^*}{a_i^*}$$

represents the values sent by P_i and received (in fifo order) by OBS since the beginning; consequently the value of t has the following form:

$$t = \frac{a_1^* \ldots a_{n-1}^*}{a_0^*} \ldots \frac{a_0^* \, a_1^* \ldots a_{i-1}^* \, a_{i+1}^* \ldots a_{n-1}^*}{a_i^*} \ldots \frac{a_0^* \ldots a_{n-2}^*}{a_{n-1}^*}$$

namely:

$$t = \frac{a_0^{n_0} \, a_1^{n-1} \ldots a_{n-1}^{n_{n-1}}}{a_0^{d_0} \, a_1^{d_1} \ldots a_{n-1}^{d_{n-1}}}$$

1. if $t = 1$ and $\exists\ m$ such that $n_m \neq d_m$ then:
   if $n_m > d_m$ then $a_m^{n_m - d_m}$ divides $a_0^{d_o} \ldots a_{m-1}^{n_{m-1}} \; a_{m+1}^{d_{m+1}} \ldots a_{n-1}^{d_{n-1}}$
   if $n_m < d_m$ then $a_m^{d_m - n_m}$ divides $a_0^{n_o} \ldots a_{m-1}^{n_{m-1}} \; a_{m+1}^{n_{m+1}} \ldots a_{n-1}^{n_{n-1}}$!!

   both case are impossible as the a_i are prime. Consequently if $t = 1$ then $n_m = d_m$, $\forall m$. In others words, to OBS's knowledge, the number of messages sent to each process P_i equals the number of messages P_i received.

2. Let us consider $t = 1$, $\forall m : \; n_m = d_m$ and the application is not terminated. We consider two cases according to the snapshot S associated to t is consistent or not [LY87].

   (a) The snapshot S is consistent (namely there is no messages received and not sent in S). As for OBS t indicates the number of messages sent equals the number of messages received and as t is consistent it follows that there is no message sent and not received. Consequently as the last time a process sent OBS a control message it was passive the application is terminated.

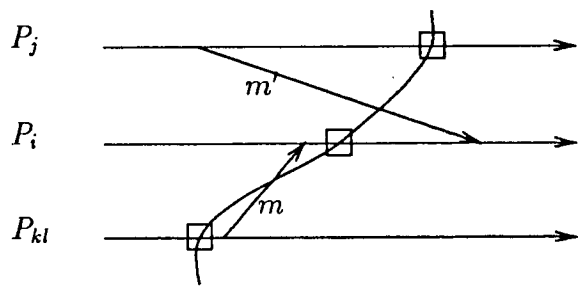


Figure 1: an inconsistent snapshot with $t = 1$

(b) The snapshot S associated to t is not consistent (there are messages in S which are perceived as received and not sent). As $t = 1$ and the snapshot associated is not consistent there is (at least) one message m whose reception by P_i is known by OBS but not its sending, and there is a corresponding message m' whose sending to P_i by some P_j is known by OBS but not its reception. (see fig. 1, $\square$ stands for the last sendings of control messages).

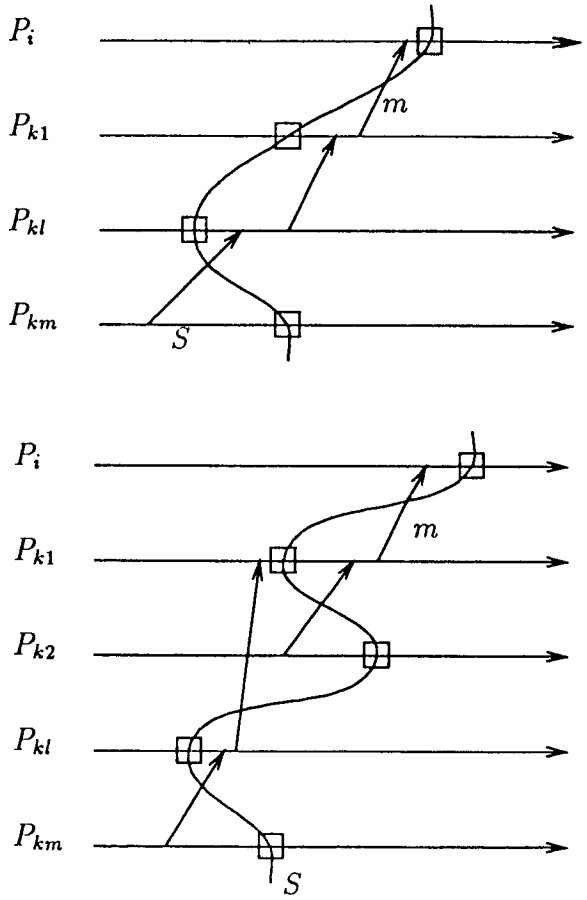


Figure 2: two inconsistent snapshots.

the message m has been sent by some P_{k1} after the snapshot associated to t. So there is a chain of activation: P_{k1} has been activated after S by P_{k2} which has been

activated by P_{k3} after S etc... such a chain is necessarily finite (the number of process is finite and an active process sends a finite number of messages): so P_{kl} has been activated after the snapshot by a message sent by P_{km} before the snapshot (figure 2). Consequently there is a process P_{kl} for which $n_{kl} \neq d_{kl}$ in t, which contradicts the hypotheses. Therefore it is not possible to have $t = 1$ when there is some activity in the system.

Remark: A similar reasonning in used by Lai and Yang in [LY87] to show that inconsistent snapshot can be used to detect deadlock.

5 ABOUT IMPLEMENTATIONS

An implementation using vectors of dimension n is obviously well-suited. For example in the termination detection case the j^{th} entry of a vector sent by P_i in a control message corresponds to the exponent of a_j. If we consider such a coding for the termination detection algorithm and an observer implemented by a token travelling along a ring (which visits periodically each process), we obtain the algorithm called "vector algorithm" proposed by Matterm in [MAT87]; if the coding uses matrices instead of vectors and if observers are put on an arbitrary number of sites we obtain an algorithm similar to the one proposed in [HJPR87].

6 CONCLUSION

A possible use of prime number to design distributed algorithms has been investigated. One of the major drawbacks in designing such algorithms lies in the inability for one or several processes (or for an external observer) to catch instantaneously a global state of a distributed system [CL85]. Properties of prime numbers (namely unicity of the decomposition of an integer) can allow to solve some of the problems such an inability has given rise to. Two very different distributed algorithms have been produced as an illustration of the proposed tool (one implementing a service - mutual exclusion -, the other implementing an observation - termination detection -).

Acknowledgements

This work has been partly supported by French National Project C^3 (CNRS Project on Parallelism). R. PEDRONO and A. COUVERT are acknowledged for their comments which improve on the proofs.

References

[CL85] CHANDY M.,LAMPORT L., *Distributed Snapshots: determining global states in distributed systems*, ACM TOCS, Vol. 3.1, (1985), pp. 63-75.

[DIJ74] DIJKSTRA E.W.D., *Self-stabilizing Systems in Spite of Distributed Control*, Comm. ACM, Vol. 17,11, (Nov. 1974), pp 643-644

[FRA80] FRANCEZ N., *Distributed Termination*, ACM Toplas, Vol. 2,1, (Jan. 1980), pp 42-55

[HJPR87] HELARY J.M., JARD C., PLOUZEAU N., RAYNAL M., *Detection of Stable Properties in Distributed Applications*, Proc. 6th ACM Symposium on PODC, Vancouver, (Aug. 1987), pp 125-136

[LEL77] LE LANN G., *Distributed Systems: Towards a Formal Approach*, IFIP Congress, Toronto, (Aug. 1977), pp 155-160

[LAM78] LAMPORT L, *Time, Clocks and the Ordering of Events in a Distributed system*, Comm. ACM, Vol. 21,7, (July 1978), pp 558-565

[LY87] LAI T.H., YANG T.H., *On Distributed Snapshots*, Inf. Processing Letters, Vol 25, (1987), pp 153-158

[MAT87] MATTERN F., *Algorithms for Distributed Termination*, Distributed Computing, Vol. 2, (1987), pp 161-175

[MIS83] MISRA J., *Detecting Termination of Distributed Computation using Markers*, Proc. 2d ACM Symposium on PODC, Montreal, (1983), pp 290-294

[RAY88] RAYNAL M., *Networks and Distributed Computation: Concepts, Tools and Algorithms*, The MIT Press, (1988), 166p

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 452 **LANCER DE RAYON : APPROCHES PARALLELES**
Didier BADOUEL, François BODIN, Thierry PRIOL
16 Pages, Janvier 1989.

PI 453 **UN COMPILATEUR ESTELLE MULTI-PROCESSEURS POUR L'EXPERIMENTATION D'ALGORITHMES DISTRIBUES SUR MACHINES PARALLELES**
Jean-Marc JEZEQUEL, Claude JARD
54 Pages, Janvier 1989.

PI 454 **REALISATION ET CALIBRATION D'UN SYSTEME EXPERIMENTAL DE VISION COMPOSE D'UNE CAMERA MOBILE EMBARQUEE SUR UN ROBOT-MANIPULATEUR**
François CHAUMETTE, Patrick RIVES
36 Pages, Février 1989.

PI 455 **ARCHITECTURE SYSTOLIQUE POUR LA CORRECTION AUTOMATIQUE DE LIBELLE D'ADRESSE**
Dominique LAVENIER, Jean-Luc SCHARBARG, Patrice FRISON
22 Pages, Février 1989.

PI 456 **DISTRIBUTION OF OPERATIONAL TIMES IN FAULT-TOLERANT SYSTEMS MODELED BY SEMI-MARKOV REWARD PROCESSES**
Gerardo RUBINO, Bruno SERICOLA
10 Pages, Février 1989.

PI 457 **TANDEM QUEUES WITH FEEDFORWARD FLOWS**
Kamel SISMAIL
16 Pages, Février 1989.

PI 458 **PRIME NUMBERS AS A TOOL TO DESIGN DISTRIBUTED ALGORITHMS**
Michel RAYNAL
14 Pages, Février 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique