On the synchronization of processes

M. Nivat

IRIA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105 78150 Le Chesnay
France
Tel. 954 90 20

Rapports de Recherche

N° 3

ON THE SYNCHRONIZATION OF PROCESSES

Maurice NIVAT

Janvier 1980

0 - INTRODUCTION

In the following pages we introduce a formalism to describe some classical problems related to the synchronisation of processes : these processes are described by the set of their infinite behaviours, each behaviour being an infinite word on the alphabet of actions including the empty action. Our approach is thus very close to the theory of path expressions [11].

The two main points are

- the introduction of delayable processes, that is processes which can be stopped either for a finite amount of time or indefinitely at any stage.

- the importance of the notion of closedness, borrowed from previous work of the author [1, 10]. We believe that the main difference in status of the two problems of deadlocks and starvation does come from the fact that the set of synchronised behaviours of a vector of processes is closed, when the set of fair behaviours is not closed.

Algorithms are given for detecting deadlocks and avoiding them, and also for detecting starvation phenomena in the case of closed rational processes which has been the most extensively studied in the litterature [2,3,4,5,6,7]. Some discussion follows of the notion of command of a vector of processes : it it is proved that in the restricted case we consider there is no finite complete fair command, that is no command using some finite state device which allow a vector of processes to take all possible fair behaviours and those only.

ON THE SYNCHRONISATION OF PROCESSES

Maurice NIVAT
Laboratoire d'Informatique Théorique
et Programmation
Université Paris VII

Résumé

Dans cet article nous introduisons un formalisme pour décrire des ensembles de comportements infinis de systèmes de processus concurrents : la condition de synchronisation est définie comme un sous ensemble de l'ensemble des actions que peuvent réaliser les processus composants du système à chaque instant. Des définitions correspondantes des "blocages" et "famines" sont également données et une notion de commande proposée pour donner un sens à l'action d'éviter blocages et famines.

Une version préliminaire, en français, de cet article doit paraître dans la Revue Technique Thomson-CSF.

Abstract

In this paper a formalism is introduced to describe infinite behaviours of a set of processes obeying some synchronisation condition which restricts the set of actions which can be performed by the processes at any given instant of time. The traditional problems of deadlocks and starvation are discussed, as well as the way to avoid both by using a command of the set of processes.

A preliminary version of this paper, in French, will appear in the Revue Technique Thomson-CSF.

June 1979

I - INFINITE WORDS

Let A be a finite alphabet. $\mathbb{N}_+$ is the set of strictly positive integers. We denote by A^* the set of finite words on A :

a finite word $f \in A^*$ is a partial mapping $f : \mathbb{N}_+ \to A$ whose domain dom(f) is of the form

$$dom(f)=[n]=\{m \in \mathbb{N}_+ \mid m \leq n\} \text{ for some } n \in \mathbb{N}$$

By definition the length of f, denoted $|f|$, is equal to n if $dom(f)=[n]$. The word with empty domain [0] is denoted ϵ and called the empty word.

We denote by A^ω the set of infinite words on A : an infinite word $u \in A^\omega$ is a total mapping $u : N_+ \to A$.

We denote by A^∞ the union $A^\infty = A^* \cup A^\omega$.

The main tool to deal with infinite words is a relation defined on A^∞ which entends the well known relation "is a left factor of" on A^*.

For all $f,g \in A^*$ we write $f \leq g$ and say that f is a left factor of g if and only if

$$|f| \leq |g| \text{ and } \forall n \in \mathbb{N}_+ \quad n \leq |f| \Longrightarrow f(n)=g(n).$$

This relation is clearly an order relation and we have $\epsilon \leq f$ for all $f \in A^*$.

For all $f \in A^*$, $u \in A^\omega$ we write $f \leq u$ and say that f is a left factor of u if and only if

$$\forall n \in \mathbb{N}_+ \quad n \leq |f| \Longrightarrow f(n)=g(n).$$

We also use the notation u[n] to denote the restriction of u to [n] which is a finite word of length n.

Thus $f \leq u \Leftrightarrow f=u[|f|]$.

If now $u,v \in A^\omega$ we define

$$u \leq v \Leftrightarrow u=v.$$

The relation $\leq$ thus defined is an order relation on A^∞ and we can state some properties [9,10].

Property 1 : Let $f_1, f_2, \ldots, f_n, \ldots$ be an infinite sequence of finite words increasing for $\leq$: there exists a unique infinite word $u \in A^\omega$ such that for all $i \in \mathbb{N}_+$ $f_i \leq u$.

For $f \in A^\omega$ denote by FG(f) the set of left factors of f :

$$FG(f) = \{g \in A^* \mid g \leq f\}$$

and for $u \in A^\omega$ denote by FG(u) the set of finite left factors of u :

$$FG(u) = \{f \in A^* \mid f \leq u\} = \{u[n] \mid n \in \mathbb{N}_+\}$$

If L is a subset of A^∞ we define FG(L) as

$$FG(L) = \bigcup \{FG(\alpha) \mid \alpha \in L\}$$

Property 2 : For all $u \in A^\omega$, $L \subset A^\infty$

$$\text{card}(FG(u) \cap L) = \infty \implies FG(u) \subset FG(L)$$

The relation $\leq$ is related with the monoïd structure of A^∞ which entends the well known monoïd structure of A^* in the following way

- for all $f, g \in A^*$ the product fg is the finite word of length $|f|+|g|$ defined by

$$\forall n \in \mathbb{N}_+ \qquad n \leq |f| \implies fg(n) = f(n)$$
$$|f| < n \leq |g| \implies fg(n) = g(n - |f|)$$

- for all $f \in A^*$, $u \in A^*$ the product fu is the infinite word given by

$$\forall n \in N_+ \qquad n \leq |f| \implies fu(n) = f(n)$$
$$|f| < n \implies fu(n) = u(n - |f|)$$

- for all $u \in A^\omega$ $\alpha \in A^\infty$ $u\alpha = u$.

It is easy to check that this product is associative and that ε is a neutral element. Its restriction to A^* is the ordinary concatenation. And we have

Property 3 : $\forall \alpha, \beta \in A^\omega \quad \alpha \leq \beta \iff \exists \gamma \in A^\omega : \alpha\gamma = \beta$

One should note here that γ is not unique since $u\alpha = u$ for all α. If we denote $\alpha < \beta$ the strict order relation

$$\alpha < \beta \iff \alpha \leq \beta \text{ and } \alpha \neq \beta.$$

We do have $\forall f, g \in A^* \quad f < g \iff \exists h \neq \varepsilon \quad fh = g$ but this is not true for infinite words.

We now introduce the concepts which will be essential in the sequel :

Let L be any ∞ language ie any subset of A^∞;

The adherence of L, denoted Adh(L), is the ω-language

$$Adh(L) = \{u \in A^\omega \mid FG(u) \subset FG(L)\}.$$

The centre of L, denoted L^c, is the language $L^c \subset A^*$ given by

$$L^c = FG(Adh(L)).$$

A number of properties can be immediately proved and are proved in [1,10].

Property 4 : For all $L \subset A^\infty \quad Adh(L) = Adh(L^c)$.

Property 5 : For all $L \subset A^*$

$$L^c = \{f \in A^* \mid \text{card}\{g \in A^* \mid fg \in L\} = \infty\}$$

Property 6 : For all $L \subset A^\omega$

$$L^c = FG(L)$$

(though L and Adh(L) may be different)

Property 7 : For all $L_1, L_2 \subset A^\infty$

$$Adh(L_1 \cup L_2) = Adh(L_1) \cup Adh(L_2)$$

Thanks to these properties one can compute L^c for $L \subset A^\infty$ knowing the finite part $L^{fin} = L \cap A^*$ and the infinite part $L^{inf} = L \cap A^\omega$.

The mapping $L \rightsquigarrow \overline{L} = L \cup Adh(L)$ is a closure in the topological sense ie satisfies

$$\overline{\emptyset} = \emptyset$$
$$L \subset \overline{L}$$
$$\overline{\overline{L}} = \overline{L}$$
$$\overline{L_1 \cup L_2} = \overline{L_1} \cup \overline{L_2}$$

Indeed the topology induced by this closure, topology whose family of open sets is the family of complements of the closed sets L such that $L = \overline{L}$, is also induced by a complete metric on A^∞ which is described in [1]. In the sequel it will be sufficient for us to use the definition of closed ∞-languages :

$L \subset A^\infty$ is a closed ∞-language if and only if $L \supset Adh(L)$.

Behaviour of processes

A process p is any mechanism able to perform actions taken in a finite set of possible actions A.

We suppose that each action in A can be performed in one unit of time.

Among these actions we distinguish the empty action $e \in A$ which denotes the fact that at a given instant p does nothing.

One behaviour of p is thus an infinite word $u \in A^\omega$ which describes the action u(n) performed by p at each instant $n \in \mathbb{N}_+$.

The process p is entirely described by its set of possible behaviours

$$C(p) \subset A^\omega.$$

We shall say that the process p is delayable if and only if the following condition holds :

$$\forall u \in C(p) \qquad \forall f < u \quad fe^{\omega} \in C(p) \text{ and } \forall k \in \mathbb{N}_+ \; : \; fe^{k}u' \in C(p)$$

where $u' \in A^{\omega}$ is such that $fu' = u$.

Intuitively this means that the process p can be stopped at any instant of time for any finite or infinite delay.

A process is said to be closed if and only if C(p) is a closed subset of A^{ω}. We can give an example of a typically not closed process : that is a process supposed to perform an action a_1 during a finite amount of time and to perform a_2 for ever afterwards. This process is described by $C(p) = a_1^{*} a_2^{\omega}$ and is not closed for

$$Adh(a_1^{*}a_2^{\omega}) = a_1^{*}a_2^{\omega} \cup a_1^{\omega} \notin C(p).$$

We shall be interested in the behaviours of a finite set of communicating processes $p_1, \ldots, p_k$. Rather than talking about a set of processes we shall talk about a vector of processes

$$\vec{p} = \langle p_1, \ldots, p_k \rangle.$$

One behaviour of $\vec{p}$, if we assume that the processes are independant (ie each process is able to perform any action at time n regardless of what the other processes are doing) is a vector $\vec{u} = \langle u_1, \ldots, u_k \rangle$ of infinite words, each u_i a behaviour of p_i.

We write $C(\vec{p}) = C(p_1) \times \ldots \times C(p_k)$ and we have $C(\vec{p}) \subset (A^{\omega})^k$

But indeed there is a clear isomorphism between $(A^{\omega})^k$ and $(A^k)^{\omega}$: the vector $\vec{u} \subset (A^{\omega})^k$ can be considered as an infinite word on the alphabet A^k if we define for all $n \in \mathbb{N}_+$

$$\vec{u}(n) = \langle u_1(n), \ldots, u_k(n) \rangle \in A^k.$$

Conversely any mapping $\vec{u} = \mathbb{N}_+ \to A^k$ can be considered as k-vector of mappings $\langle u_1,\ldots,u_k\rangle$ where $u_i : \mathbb{N}_+ \to A.$, is given by $u_i(n)=(\vec{u}(n))_i$.

Let us now introduce a constraint on the vector of actions which may be performed simultaneously by the processes $\langle p_1,\ldots,p_k\rangle$.

We define a synchronisation condition S as a subset of A^k which does not contain the k-vector of actions $\vec{e} = \langle e,e,\ldots,e\rangle$.

(This last conditon is to prevent the processes to stop at the same time : we shall be interested only in the behaviours of $\vec{p}$ for which at any instant of time at least one process in performing a non empty action).

We now define an S-synchronised behaviour of $\vec{p}$ as a behaviour $\vec{u}$ of $\vec{p}$ such that

$$\forall n\in \mathbb{N}_+ \qquad \vec{u}(n)\in S.$$

The set of S-synchronized behaviours of $\vec{p}$ is thus

$$C_S(\vec{p}) = C(\vec{p}) \cap S^\omega.$$

It is traditional to consider two problems in connection with the synchronized behaviours of a vector of processes

1) - <u>The deadlock problem</u>

The processes $\vec{p}$, synchronised by S reach a <u>deadlock</u> iff they have been acting in a synchronized way up to a certain instant n but cannot go further without violating the synchronisation contion.

Let us make precise the definition of a deadlock in our formalism.

An S-deadlock of $\vec{p} = \langle p_1,\ldots,p_k\rangle$ is a k-vector $\vec{f}$ of finite words all of whose components have the same length n satisfying

- $\vec{f} \in (A^k)^* \cap S^*$
- $\vec{f} \in FG(C(\vec{p}))$
  (this means $\forall i = 1,\ldots k \quad f_i \in FG(C(p_i))$)
- for all $\vec{s}\in S \quad \vec{f}\vec{s} = \langle f_1 s_1,\ldots,f_k s_k\rangle \notin FG(C(\vec{p}))$

The most important property regarding S-deadlocks is the following

Property 8 : If for $i = 1,\ldots,k$ p_i is a closed process, and S is a synchronisation condition, the set $C_S(\vec{p})$ of S-synchronized behaviours of $\vec{p}$ is precisely the set

$$C_S(\vec{p}) = \{u \in (A^k)^\omega \mid \forall n \in \mathbb{N}_+ : \vec{u}[n] \in S^* \cap C(\vec{p}) \quad \vec{u}[n] \text{ is not an S-deadlock}\}$$

Proof : Indeed $\vec{u} \in C_S(\vec{p})$ implies for all $n \in \mathbb{N}_+$: $\vec{u}[n] \in S$, $\vec{u}(n+1) \in S$ and $\vec{u}[n+1] = \vec{u}[n]\,\vec{u}(n+1) \in FG(C_S(\vec{p})) \subset FG(C(\vec{p}))$ proving that $\vec{u}[n]$ is not an S-deadlock.

Conversely if for all $n \in \mathbb{N}_+$ $\vec{u}[n] \in FG(C(\vec{p})) \cap S^*$ the closedness of p_i implies $\vec{u} \in C(\vec{p})$ and the (obvious) closedness of S^ω implies $\vec{u} \in S^\omega$. Whence if for no $n \in \mathbb{N}_+$, $\vec{u}[n]$ is a deadlock, $\vec{u} \in C_S(\vec{p})$. □

Remark : The property 8 shows the crucial rôle played by the closedness of the processes. Intuitively prop 8 says that the behaviours of $\vec{p}$ which avoid deadlocks are "good" synchronized behaviours. Or else that preventing $\vec{p}$ to reach an S-deadlock is sufficient to have it go on forever.

Obviously this would not be the case with a not closed process such as the process p whose behaviours are described by

$$C(p) = (a \cup e)^*(b \cup e)^\omega.$$

2 - THE STARVATION PROBLEM

The behaviour $\vec{u}$ of $\vec{p}$ induces starvation of p_i iff there exists $n \in \mathbb{N}$ such that for all $n' \geq n$: $u_i(n') = e$.

The set of synchronized behaviours $\vec{u}$ of $\vec{p}$ which induce no starvation is denoted $F_S(\vec{p})$, such a behaviour is called fair. A simple exemple shows that usually $F_S(\vec{p})$ is not closed.

Example : $\vec{p} = \langle p_1, p_2 \rangle$, $C(p_1) = (a_1 \cup e)^\omega$ $C(p_2) = (a_2 \cup e)^\omega$ $S = \{(a_1 \cup e), (a_2 \cup e)\} \setminus \{\langle e,e \rangle\}$.

Clearly $C_S(\vec{p}) = (\langle a_1, e \rangle \cup \langle e, a_2 \rangle \cup \langle a_1, a_2 \rangle)^\omega$.

$$F_S(p) = \{\langle u_1, u_2 \rangle \mid u_1 \in (a_1 \cup e)^\omega \; u_2 \in (a_2 \cup e)^\omega \text{ and } \forall n \in \mathbb{N} \; \exists n' > n, \; n'' > n$$
$$u_1(n') = a_1 \text{ and } u_2(n'') = a_2\}.$$

Thus $\{\langle e^n a_1^\omega, a_2^\omega \rangle \mid n \in \mathbb{N}\} \subset F_S(\vec{p})$ but $\vec{u} = \langle e^\omega, a_2^\omega \rangle$ is such that $FG(\vec{u}) \subset FG(F_S(\vec{p}))$ though $\vec{u} \notin F_S(\vec{p})$ showing that $F_S(\vec{p})$ is not closed.□

The starvation problem is then twofold

1 - determine whether some $\vec{u} \in C_S(\vec{p})$ induces some starvation, in other words decide whether $C_S(\vec{p}) = F_S(\vec{p})$.

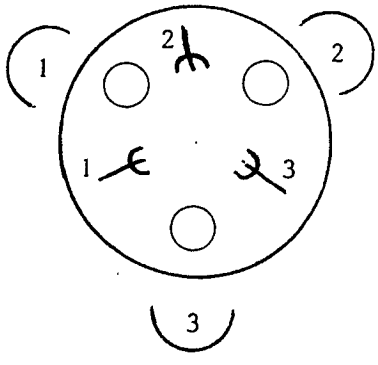
2 - Avoid starvation ie describe $F_S(\vec{p})$ as the set of all synchronized behaviours which obey some rule whose respect can be checked at every instant of time.

III - A WELL-KNOWN EXEMPLE

This is the celebrated problem of philosophers eating noodles [2].

We consider 3 philosophers p_1, p_2, p_3 :

a table is set for them with seats assigned, a plate in front of each seat, and a fork between two plates as shown on the drowing.



A philosopher p_i can only do the following sequence of actions

- seat in his place : a_i
- seize his left fork b_{i+1} (mod 3) and seize his right fork afterwords b_i

or

seize his right fork, b_i, and seize his left fork afterwords, b_{i+1} (mod 3)

- then eat c_i
- then replace the forks in either order $d_i d_{i+1}$ of $d_{i+1} d_i$ (we omit the mod 3 indication)
- then go away and think g_i

And repeat endlessly the same sequence.

The set of behaviours of the philosopher p_i is thus reprensented by

$$C(p_i) = (a_i(b_{i+1}b_i \cup b_i b_{i+1})\ c_i(d_{i+1}d_i \cup d_i d_{i+1})\ g_i)^\omega.$$

In order to make it a delayable process we introduce an arbitrary sequence of empty actions everywhere we thus define

$$R_i = (e^* a_i e^* (b_{i+1} e^* b_i \cup b_i e^* b_{i+1})\ e^* c_i e^* (d_{i+1} e^* d_i \cup d_i e^* d_{i+1})\ e^* g_i)^\omega$$

and take as the set of actions of the now delayable process p_i

$$C(p_i) = R_i \cup FG(R_i)\ e^\omega.$$

We note that p_i is a closed process.

The behaviour of forks can be represented in a similar way :

the fork numbered j denoted p_{j+3} can be seized and be replaced, seized again and replaced again endlessly.

If we introduce the possibility of arbitrary delays we are lead to define

$$R_{j+3} = (e^* \overline{b_j} e^* \overline{d_j})^\omega$$

and

$$C(p_{j+3}) = R_{j+3} \cup FG(R_{j+3})\ e^\omega.$$

The process p_{j+3} (that is the fork numbered j) is also closed.

The synchronisation condition is obvious :

a fork cannot be seized ($\overline{b_j}$) without a philosopher seizing it (b_j) at the same time, and a philosopher cannot seize a fork (b_i) without its being seized ($\overline{b_j}$) . Also a fork can be seized only once at each time.

The synchronisation condition S is thus the set $S \subset A^6$ given by :

S is the set of sextuple of actions $\langle s_1,\ldots,s_6\rangle$ such that

1 - they are not all empty

2 - if there exists $i\in\{1,2,3\}$, $j\in\{1,2,3\}$ such that $s_i=b_j$ then $s_{j+3} = \overline{b}_j$ and $i'\in\{1,2,3\}\setminus\{i\}$ implies $s_{i'}\neq b_j$ (same thing for d_j)

3 - is there exists $j\in\{1,2,3\}$ such that $s_{j+3}=\overline{b}_j$ then there exists a unique $i\in\{1,2,3\}$ such that $s_i=b_j$ (same thing for $\overline{d}_j$)

On this simple exemple we can easily see the deadlocks and starvation phenomena :

- a deadlock appears if the three philosophers seize then left fork at the same time, an event which is represented by the sextuple of words

$$\begin{aligned} f_1 &= a_1 b_2 \\ f_2 &= a_2 b_3 \\ f_3 &= a_3 b_1 \\ f_4 &= e\ \overline{b}_1 \\ f_5 &= e\ \overline{b}_2 \\ f_6 &= e\ \overline{b}_3 \end{aligned}$$

Now vector of actions $\vec{s} = \langle s_1,\ldots,s_6\rangle \in S$ can allow the six processes to go on.

- a starvation phenomena appears whenever at least one philosopher is , after a while, prevented to eat forever.
The following behaviour of $\vec{p}$ induces starvation of the philosophers numbered 2 and 3 :

$$\begin{cases} u_1 = (a_1 b_1 b_2 c_1 d_1 d_2 g_1)^\omega \\ u_2 = e^\omega \\ u_3 = e^\omega \\ u_4 = (e\ \overline{b}_1\ e\ e\ \overline{d}_1\ e\ e)^\omega \\ u_5 = (e\ e\ \overline{b}_2\ e\ e\ \overline{d}_2\ e)^\omega \\ u_6 = e^\omega \end{cases}$$

This is a synchronized behaviour which can also be represented as

$$n = (<a_1,e,e,e,e,e><b_1,e,e,\overline{b}_1,e,e><b_2,e,e,e,\overline{b}_2,e><c_1,e,e,e,e,e><d_1,e,e,\overline{d}_1,e,e> \\ <d_2,e,e,e,\overline{d}_2,e><g_1,e,e,e,e,e>)^{\omega}.$$

IV - CLOSED RATIONAL PROCESSES : THE DEADLOCK PROBLEM

A process p is closed and rational iff there exists a rational language $K \subset A^*$ such that $C(p) = Adh(K)$. If this is the case $C(p) = Adh(FG(C(p)))$ and $FG(C(p))$ is rational and conversely.

In case for all $i=1,\ldots,k$, p_i is a closed rational process we can solve entirely the deadlock problem, the solution goes through the definition and use of finite sink automata

Definition : a finite sink automation (abbreviated f.s.a.) on A, is given as a quadruple $A = <Q,q_0,q_s,\lambda>$ where

- Q is a finite set of states
- $q_0 \in Q$ is the initial state
- $q_s \in Q$ is the sink state
- $\lambda : Q \times A \to Q$ is the transition function satisfying $\forall a \in A \quad \lambda(q_s,a) = q_s$.

The function $\lambda : Q \times A \to Q$ is extended into a function $\lambda : Q \times A^* \to Q$ as for an ordinary automation.

If A is a f.s.a. (finite sink automation) we define the ω-language recognized by A as $V(A) = \{u \in A^{\omega} \mid \forall n \quad \lambda(q_0, u[n]) \neq q_s)\}$.

We can make a first remark :

suppose A is the above f.s.a. . Let us call a live state any state $q \in Q$ such that there exists $u \in V(A)$ and $n \in \mathbb{N}_+$ such that $\lambda(q_0,u[n]) = q$.

A necessary and sufficient condition for q to be a live state is that there exist arbitrary long words f such that $\lambda(q,f) \neq q_s$ and this is equivalent to the condition

$$\lambda(q,A^N) = \{\lambda(q_0,f) \mid f \in A^*,\ |f| = N\} \neq \{q_s\}$$

where N is the number of states.

Clearly thus, given A, one can determine the subset of live states and build a live automaton A' equivalent to A in the sense that

$$V(A') = V(A).$$

To build A' one takes as set of states $Q' = Q_\ell \cup \{q_s\}$ where Q_ℓ is the subset of live states of A. The initial state is q_0, the sink state q_s and the new transition function $\lambda' : Q' \times A \to Q'$ is given by

$$\lambda'(q,a) = \lambda(q,a) \text{ if } q \in Q_\ell \text{ and } \lambda(q,a) \in Q_\ell$$

$$\lambda'(q,a) = q_s \text{ if } q \notin Q_\ell \text{ or } q \in Q_\ell \text{ and } \lambda(q,a) \notin Q_\ell.$$

Verifying that $V(A') = V(A)$ is immediate : A' is said to be a live automaton for all the states but the sink state are live states. We may state that every f.s.a. is equivalent to a live f.s.a. .

The main property is then

Property 9 : The closed ω-language $L \subset A^\omega$ is rational if and only if $L = V(A)$ for some finite sink live automaton A.

Proof : Suppose $L = V(A)$ where A is a live f.s.a. .

Define $F(A) = \{f \in A^* \mid \lambda(q_0,f) \neq q_s\}$. Clearly $L = Adh(F(A))$ and $F(A) = FG(L)$. Thus L is closed and rational since $L \supset Adh(FG(L))$ and $FG(L) = FG(A)$ is rational being recognized by the ordinary finite automaton A with $Q_f = Q \setminus \{q_s\}$ as the subset of final states.

Conversely suppose L is closed, $L \subset A^\omega$ and FG(L) is rational.

We have indeed L = Adh(FG(L)).

Suppose FG(L) is recognized by the ordinary finite automatom

$$A = \langle Q, q_0, Q_f, \lambda \rangle$$

this meaning that $FG(L) = \{f \in A^* \mid \lambda(q_0, f) \in Q_f\}$.

Consider then the subset Q_ℓ of Q given by

$$Q_\ell = \{q \in Q \mid \exists f \in A^* \quad \lambda(q,f) \in Q_f\}.$$

One can build a finite sink automaton A' in the following way

- the set of states Q' is $Q_\ell \cup \{q_s\}$
- the initial state is q_0
- the sink state is q_s
- the transition function $\lambda' : Q' \times A \to Q'$ is defined by

$$\lambda'(q,a) = \lambda(q,a) \text{ if } q \in Q_\ell \text{ and } \lambda(q,a) \in Q_\ell$$
$$\lambda'(q,a) = q_s \text{ if } q \notin Q_\ell \text{ or } q \in Q_\ell \text{ and } \lambda(q,a) \notin Q_\ell \ .$$

It is immediate to verify that FG(L) = F(A') and that A' is a live automatom. Whence L = Adh(FG(L)) = V(A'). □

We can now give the main results of this section.

Theorem : if $\vec{p} = \langle p_1, \ldots, p_k \rangle$ is a k-vector of closed rational processes then $C(\vec{p})$ and $C_S(\vec{p})$ are closed rational ω-languages on A^k and one can build a finite sink automaton recognizing $C(\vec{p})$ and $C_S(\vec{p})$ for all $S \subset A^k$.

Proof : Let for all i = A,...,k $C(p_i) = V(A_i)$ for some live f.s.a. A_i.

We build an automaton $\vec{A}$ on A^k which recognizes $C(\vec{p})$ very easily : it is the cartesian product of the A_i with all sink states analgamated in a single one.

The set of states is thus $Q = Q_{1\ell} \times Q_{2\ell} \times \ldots \times Q_{k\ell} \cup \{q_s\}$.

The initial state is $\vec{q}_0$.

The transition function $\vec{\lambda} : \vec{Q} \times A^k \to \vec{Q}$ is given by

$$\vec{\lambda}(<q_1,\ldots,q_k>, <a_1,\ldots,a_k>) = <\lambda(q_1,a_1),\ldots,\lambda(q_k,a_k)>$$

if for all $i = 1,\ldots,k \quad \lambda(q_i,a_i) \in Q_{i\ell}$.

$$\vec{\lambda}(<q_1,\ldots,q_k>, <a_1,\ldots,a_k>) = q_s$$

if for some $i \in \{1,\ldots,k\} \quad \lambda(q_i,a_i) = q_s \qquad \vec{\lambda}(q_s,\vec{a}) = q_s$.

The set $V(\vec{A})$ is by definition the set of all $\vec{u} \in (A^k)$ such that $\forall n \in \mathbb{N}_+$ $\vec{\lambda}(\vec{q}_0,\vec{u}[n]) \neq q_s$. Thus $\vec{u} \in V(\vec{A}) \iff \forall i = 1,\ldots,k \quad \forall n \in \mathbb{N}_+ \quad \lambda(q_0,u_i[n]) \neq q_s$ in other words since $C(p_i) = V(A_i)$

$$\vec{u} \in V(\vec{A}) \iff \forall i = 1,\ldots,k \quad u_i \in C(p_i).$$

We have proved $V(\vec{A}) = C(\vec{p})$.

Suppose we are now given a synchronisation condition $S \subset A^k$. The S-deadlocks appear very clearly in the picture : an S-deadlock is just a word $\vec{f} \in S^*$ such that

$$\vec{\lambda}(\vec{q}_0,\vec{f}) = \vec{q} \neq q_s \text{ and } \quad \forall \vec{s} \in S \quad \vec{\lambda}(\vec{q}_0,\vec{f}\,\vec{s}) = q_s.$$

Obviously there exists an algorithm to decide whether there exists any S-deadlock since this amounts to decide whether there exists an S-accessible state $\vec{q} \neq q_s$ ($\vec{q}$ is S-accessible $\iff \exists \vec{f} \in S^* : \vec{\lambda}(\vec{q}_0,\vec{f}) = \vec{q}$) such that $\forall \vec{s} \in S$ $\vec{\lambda}(\vec{q},\vec{s}) = q_s$.

But we can do more ie build from $\vec{A}$ an automaton recognizing $C_S(\vec{p})$. To do this we first determine all the S-accessible states $\vec{q}$ which are not such that $\vec{\lambda}(\vec{q},S) = \{q_s\}$. Call $\vec{Q}_S$ this subset of $\vec{Q}$. The automaton $\vec{A}_S$ has

$\vec{Q}_S \cup \{q_s\}$ as set of states

$\vec{q}_0$ as initial state

q_s as sink state

the following transition function $\vec{\lambda}_S : (\vec{Q}_S \cup \{q_s\}) \times A^k \to \vec{Q}_S \cup \{q_s\}$ given by $\vec{\lambda}_S(\vec{q},\vec{a}) = \vec{\lambda}(\vec{q},\vec{a})$ if $\vec{q} \in \vec{Q}_S$, $\vec{a} \in S$ and $\vec{\lambda}(\vec{q},\vec{a}) \in \vec{Q}_S$ $\vec{\lambda}_S(\vec{q},\vec{a}) = q_s$ if $\vec{q} \notin Q_s$ or $\vec{a} \notin S$ or $\vec{\lambda}(\vec{q},\vec{a}) \notin \vec{Q}_S$ and $\vec{\lambda}_S(q_s,\vec{a}) = q_s$.

By definition the recognized ω-language $V(\vec{A}_S)$ is the set of all $\vec{u} \in (A^k)^\omega$ such that for all $n \in \mathbb{N}_+$ $\lambda_S(\vec{q}_0,\vec{u}[n]) \neq q_s$.

This is exactly the set of all $\vec{u} \in C(\vec{p})$ such that $\forall n \in \mathbb{N}_+$ $\vec{u}[n] \in S^* \cap FG(C(\vec{p}))$ and $\vec{u}[n]$ is not an S-deadlock. But property 8 we thus have $V(\vec{A}_S) = C_S(\vec{p})$. □

Commands to avoid deadlocks

The detection of deadlocks is one thing, avoiding them is another. If we wish to formalize the notion of a command we are lead to consider a command as an external mechanism which can register in some memory M an information on the behaviour of $\vec{p}$ up to instant $n \in N_+$ and tells which vectors of actions can be performed by $\vec{p}$ at instant n+1. Thus we define

Definition : A command D of $\vec{p}$ synchronized by S is given as a quadruple $D = \langle M, m_0, \phi, \psi \rangle$ where

- M is a set of states (finite or infinite : the command is said to be finite if and only if M is finite)
- an initial state $m_0 \in M$
- a mapping $\phi : M \to 2^S \setminus \{\emptyset\}$
- a mapping $\psi : M \times S \to 2^M \setminus \{\emptyset\}$

(the command D is said to be deterministic iff for all $(m,s) \in M \times S$ $\psi(m,s)$ is a singleton, non deterministic if this last condition is not true, finitary iff for all $(m,s) \in M \times S$ $\psi(m,s)$ is a finite subset of M)

A behaviour $\vec{u}$ of $\vec{p}$ obeys the command D iff there exists an infinite sequence of states $m_0, m_1, \ldots, m_n, \ldots$ such that $\forall n \in \mathbb{N}_+$ $\vec{u}(n) \in \phi(m_{n-1})$ and $m_n \in \psi(\vec{u}(n), m_{n-1})$.

Intuitively $\phi(m_{n-1})$ describes the subset of S which can be performed by $\vec{p}$ at time n, and $\psi(\vec{u}(n),m_{n-1})$ the subset of M in which D may move, depending on the special choice of $\vec{u}(n)$ which has been made by $\vec{p}$.

We denote by $C_D(\vec{p})$ the set of behaviours of $\vec{p}$ which obeys the command D. And we can thus define

Definition : The command D is adequate for S iff $C_D(\vec{p}) \subset C_S(\vec{p})$.

The command D is completely adequate for S iff $C_D(\vec{p}) \subset C_S(\vec{p})$.

The command D is fair for S iff $C_D(\vec{p}) \subset F_S(\vec{p})$.

The command D is completely fair for S iff $C_D(\vec{p}) = F_S(\vec{p})$.

What we have proved above is that if for all $i = 1,\ldots,k$, p_i is closed and rational, then there exists a finite deterministic command D which is completely adequate for S.

Indeed consider $\vec{A}_S$ and define the command D defined by $M = \vec{Q}_S$, $m_0 = \vec{q}_0$, $\phi(\vec{q}) = \{s \in S \mid \vec{\lambda}(\vec{q},\vec{s}) \neq q_s$ which is not empty since $\vec{q}$ is a live state $\psi(\vec{q},\vec{s}) = \vec{\lambda}(\vec{q},\vec{s}) \in M$ by construction.

Cheching that $V(\vec{A}_S) = C_D(\vec{p})$ is a straightforward and entirely formal matter. We can state

Theorem 2 : The exists a finite deterministic command which is completely adequate for S, whenever $\vec{p}$ is a closed rational vector of processes (that is each component p_i is closed rational).

V - CLOSED RATIONAL PROCESSES : THE STARVATION PROBLEM

First we remark that detecting possibilities of starvation is easy :

Let $\vec{p}$ be closed rational. There exists a possibility of starvation iff there exists $\vec{u} \in C_S(\vec{p})$ such that for all sufficiently large n's $u_i(n)=e$. But this implies the existense of u loop in $\vec{A}_S$ which is empty for i if we define

Definition = A loop of a f.s.a. A is a pair (q,f) such that $\lambda(q,f) = q$ and $q \neq q_s$.

A loop of $\vec{A}$ is empty for i iff $f_i \subset e^*$.

One sees immediately that the existence of a loop in $\vec{A}_S$ which is empty for i implies the existence of $\vec{u} \in V(\vec{A}_S) = C_S(\vec{p})$ which induces starvation of p.

These simple remarks are embodied in the following statement.

Theorem 3 : If $\vec{p}$ is a k-vector of closed rational processes, there exists an algorithm to decide whether some $\vec{u} \in C_S(\vec{p})$ induces starvation of some process p_i, $i \in \{1,\ldots,k\}$.

Proof it is entirely trivial : if there exists a loop of $\vec{A}_S$ which is empty for i then there exists such a loop which is elementary (ie if $(\vec{q},\vec{f})$ is the loop, $\vec{f} \in (A^k)^n$, all the states $\lambda(q,f(n))$, n' n are distinct) and an exhaustive search of elementary loops (whose length $|\vec{f}|$ is less than card $\vec{Q}$) is possible. □

But according to the definitions given above the main problems are to build commands which are fair for S (if any) and commands which are completely fair for S.

We shall first prove the

Theorem 4 : There exist vectors of closed rational processes for which no finitary completely fair commands exist.

Proof : We show that if D is a finitary command $C_D(\vec{p})$ is closed : since we know that there exist vectors of closed rational processes such that $F_S(\vec{p})$ is not closed (see exemple above) the result follows.

Let us introduce the alphabet Z

$$Z = \{(m,s,m') \mid m,m' \in M,\ s \in S,\ s \in \phi(m) \text{ and } m' \in \psi(m,s)\}$$

Define the projection $\pi_2 : Z^* \to S^*$ given by $\pi_2(m,s,m') = s$.

Clearly $C_D(\vec{p}) = \pi_2(L)$ where $L \subset Z^\omega$ is the ω-language formed of all the infinite words $v \in Z^\omega$ such that

$$\forall n \in \mathbb{N}_+ \quad \pi_3(v(n)) = \pi_1(v(n+1)) \text{ and } \pi_1(v(1)) = m_0.$$

The set L is obviously closed.

To show that if D is finitary $\pi_2(L)$ is closed we use Koenig's lemma : consider $u \in S^\omega$ such that $FG(u) \subset FG(\pi_2(L))$.

Since π_2 is alphabetical $FG(\pi_2(L)) = \pi_2(FG(L))$ whence for all $n \in \mathbb{N}_+$ there exists $v_n \in FG(L)$ such that $\pi_2(v_n) = u[n]$.

Define $E_n = \{v_n \in FG(L) \mid \pi_2(v_n) = u[n]\}$.

From the above remark it follows that $E_n \neq \emptyset$ for all n, and the finitary condition implies that E_n is finite for all n. Clearly for all n and $y \in E_{n+1}$ there exists $x \in E_n$ such that $x \leq y$ (it suffices to delete the last letter of y).

Whence by Koenig's lemma there exists an infinite sequence $v_1, v_2, \ldots, v_n, \ldots$ such that

$$\forall n \in \mathbb{N}_+ \quad v_n \in E_n \text{ and } v_n < v_{n+1}.$$

This sequence has obviously a limit $v \in Z^\omega$. The fact that L is closed implies that $v \in L$ since $\forall n : v_n \in FG(L)$.

We have $u = \pi_2(v)$ whence $u \in \pi_2(L) = C_D(\vec{p})$. □

Remark : We cannot escape the finitary condition is the statement of Thm 4 :

let us go back to the exemple given above

$$C(p_1) = (a_1 \cup e)^\omega \quad C(p_2) = (a_2 \cup e)^\omega$$

and

$$S = \{\langle a_1, e\rangle, \langle e, a_2\rangle, \langle a_1, a_2\rangle\}.$$

We can build a not finitary completely fair command for S even though $F_S(\vec{p})$ is not closed.

Take three copies of $\mathbb{N}$ denoted $\mathbb{N}$, $\overline{\mathbb{N}}$, $\overline{\overline{\mathbb{N}}}$ and a special element m_0. Define $M = \mathbb{N} \cup \overline{\mathbb{N}} \cup \overline{\overline{\mathbb{N}}} \cup \{m_0\}$.

Consider the following mappings

$$\phi(m_0) = S$$

$$\psi(m_0,s) = \mathbb{N} \cup \overline{\mathbb{N}} \cup \overline{\overline{\mathbb{N}}}$$

$$\phi(n) = \{<a_1,e>\} \text{ if } n \neq 0$$

$$\psi(n,s) = \{n-1\} \text{ if } n \neq 0$$

$$\phi(0) = S$$

$$\psi(0,s) = \overline{\mathbb{N}} \cup \overline{\overline{\mathbb{N}}}$$

$$\phi(\overline{n}) = \{<e,a_2>\} \text{ if } n \neq 0$$

$$\psi(\overline{n},s) = \{\overline{n-1}\} \text{ if } n \neq 0$$

$$\phi(\overline{0}) = S$$

$$\psi(\overline{0},s) = \mathbb{N} \cup \overline{\overline{\mathbb{N}}}$$

$$\phi(\overline{\overline{n}}) = \{<a_1,a_2>\} \text{ if } n \neq 0$$

$$\psi(\overline{\overline{n}},s) = \{\overline{\overline{n-1}}\} \text{ if } n \neq 0$$

$$\phi(\overline{\overline{0}}) = S$$

$$\psi(\overline{\overline{0}},s) = \mathbb{N} \cup \overline{\mathbb{N}}$$

It is clear that $F_S(\vec{p}) = C_D(\vec{p})$ since every $\vec{u} \in F_S(\vec{p})$ can be factorized in

$$\alpha_1^{n_1} \alpha_2^{n_2} \alpha_3^{n_3} \ \ldots \ \alpha_p^{n_p} \ \ldots$$

where $\alpha_i \in \{<a_1,e>, <e,a_2>, <a_1,a_2>\}$ $n_i \in \mathbb{N}$ and for all i : $\alpha_{i+1} \neq \alpha_i$.

But a non finitary command looks very much like an <u>oracle</u> which chooses from times to times an integer at random : no finitary process can simulate such an oracle. □

On the other hand it is possible to build a finitary fair command for S whenever $F_S(\vec{p})$ is not empty.

We sketch below such a construction.

D is fair for S iff $C_D(\vec{p}) \subset F_S(\vec{p})$ that is the command D has only to avoid entering loops which induce starvation of some p_i. Now $F_S(\vec{p})$ is not empty if and only if $\vec{A}_S$ contains a fair loop : the loop $(q,\vec{f})$ is fair iff $\forall i \quad f_i \in A^*(A\backslash\{e\})\ A^* = A^*\backslash e^*$.

Suppose there exists a fair behaviour $\vec{u}$ of $\vec{p}$ and consider one state $q \in Q$ such that $\lambda(q_0, \vec{u}[n]) = q$ for an infinite number of n's. we are sure that such a state exists and we can order the sequence $n_1 < n_2 < \ldots < n_\ell < \ldots$ of the n's such that $\lambda(q_0, \vec{u}[n_\ell]) = q$.

Clearly $(q, \vec{u}\ (n_\ell+1) \ \ldots \ \vec{u}(n_{\ell+1}))$ is a loop for all $\ell \in N_+$, denote it $(q,\vec{v}_\ell)$ with $\vec{v}_\ell = \vec{u}(n_\ell+1) \ \ldots \ \vec{u}(n_{\ell+1})$.

By the fairness of $\vec{u}$ there exists ℓ_1 such that $(v_{\ell_1})_1 \notin e^*$, then $\ell_2 > \ell_1$ such that $(v_{\ell_2})_2 \notin e^*$ and so on ...

Eventually $\vec{v}_{\ell_1}\ \vec{v}_{\ell_2} \ldots \vec{v}_{\ell_k}$ is such that for all $i = 1,\ldots,k$: $(v_{\ell_1})_i \ldots (v_{\ell_k})_i \notin e^*$ since $(v_{\ell_i})_i \notin e^*$. And $(q,\vec{v}_{\ell_1} \ldots \vec{v}_{\ell_k})$ is a fair loop.

The converse obvious : if $(q,\vec{f})$ is a fair loop and q is accessible, ie there exists a $\vec{g}$ such that $\lambda(q_0,\vec{g}) = q$, then $\vec{g}(f)^\omega$ is a fair behaviour of p.

Building a fair command amounts to find a fair loop and this is not extremely difficult though there need not exist an elementary fair loop : indeed if there exists $\vec{v}_{\ell_1}$ such that the $|\vec{v}_{\ell_1}| < 2N + 1$ because $\vec{v}_{\ell_1}$ can be factorized then in

$$\vec{w}_1\ \vec{s}\ \vec{w}_2 \text{ such that } \lambda(q,\vec{w}_1) = q_1, \quad \lambda(q_1,\vec{s}) = q_2 \text{ and } \lambda(q_2,\vec{w}_2) = q.$$

And we can always assume that $|\vec{w}_1|$ and $|\vec{w}_2|$ are less than N. All this allows us to state that if there is a fair q-loop then there exists a fair q-loop of length less than $k\times(2N+1)$.

Whence there exists an algorithm to determine whether there exists a fair loop and a fair command may be obtained by just pushing $\vec{p}$ into that fair loop (that will be a deterministic command).

Remark : from the above discussion it is clear that one can do better and for exemple build a command which allows $\vec{p}$ to take all the behaviours of the form $\vec{g}(F_q)^\omega$ where $\vec{g}$ is some elementary word such that $\lambda(q_0,\vec{g}) = q$ and F_q the set of words $\vec{f}$ such that $(q,\vec{f})$ is a loop and $|\vec{f}|$ is bounded by any integer. And thus we can build a command D such that, for any given fair behaviour $\vec{u}$, $\vec{u}\in C_D(\vec{p}) \subset F_S(\vec{p})$.

VI - CONCLUSION

In the litterature about synchronized processes, closed rational processes are the most widely studied class of processes. They cover a wide range of applications.

Other interesting classes however exist such as the class of "producer-consumers" which communicate via infinite buffers. The consideration of such processes lead to similar developments, with closed algebraic processes (whose set of behaviours is aclosed algebraic ω-language ie a set of the form Adh(L) for some algebraic language L) or with closed FIFO processes, very similar to the algebraic ones, one simply replacing the push-down automaton (or counter) in the definition of L by a queue working in the FIFO mode.

Before being able to treat these non rational processes as we did treat rational ones some work has to be done about ω-languages which are not rational.

Another question which can be well formalized along the preceding lines is the measure for a given synchronized behaviour of a "mean waiting time" of the processes and the search for a synchronized behaviour which minimizes this waiting time.

VII - ACKNOWLEDGEMENTS

The author is greatly indebted to several people.

- his previous work on ω-languages and topological properties of A^{∞} has been partly done in cooperation with A. Arnold, L. Boasson, B. Courcelle.
- his acquaintance which the synchronisation problem was started and developed in a working group organized jointly by the Laboratoire d'Informatique Théorique et programmation and the Laboratoire Central de recherches Thomson-CSF. The weekly meetings of this working group were the occasion of stimulating discussions with J. Beauquier, F. Boussinot, G. Cousineau, G. Ruggiu and G. Roucairol
- this paper was mainly written during a few days spent in Pisa, at the invitation of U. Montanari who provided the author both with a very pleasant working atmosphere and constructive criticism.

BIBLIOGRAPHY

[1] L. Boasson et M. Nivat : Adherences of languages, Report LITP 79-13, University of Paris 7 (1979).

[2] F.W. Dijkstra : Cooperating sequential processes, in F. Genuys (Ed.) : Programming languages, Academic-Press, New York (1968).

[3] N. Francez, C.A.R. Hoare, D.J. Lehmann, W.P. de Roever : Semantics of non determinism, concurrency and communication, report, USC Los Angeles, (1978).

[4] C.A.R. Hoare : Communnicating sequential processe, Communications of the Ass. Comp. Mach. Vol 21 (1978), 666-677.

[5] G. Kahn and D. MacQueen : Coroutines and networks of parallel processes, in Proceedings IFIP Congress 1977, North Holland, Amsterdam (1977).

[6] P.R. Lauer and R.H. Campbell : Formal semantics of a class of high level primitives for coordinating concurrent processes, Acta Informatica, Vol 5 (1975), 297-332.

[7] A. Mazurkiewicz : Concurrent program schemes and their interpretations, in Proceedings Aarhus workshop on Verification of parallel processes (1977).

[8] R. Milner : Synthesis of communicating behaviours, in Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science, Springer Lecture Notes, (1978).

[9] M. Nivat : Mots infinis engendrés par une grammaire algébrique, RAIRO Informatique Théorique Vol 11 (1977) 311-327.

[10] M. Nivat : Sur les ensembles de mots infinis engendrés par une grammaire algébrique, RAIRO Informatique Théorique Vol 12 (1978) 259-278.

[11] P.R. Torrigiani and P.E. Lauer : An object oriented notation for path expressions, in Proceedings AICA Conference (1977), 3rd Vol, Software methodologies.