

Dérivation d'un algorithme réparti de détection de la terminaison d'une application répartie

Jean-Michel Hélary, Michel Raynal

▶ **To cite this version:**

Jean-Michel Hélary, Michel Raynal. Dérivation d'un algorithme réparti de détection de la terminaison d'une application répartie. [Rapport de recherche] RR-1148, INRIA. 1989. inria-00075411

HAL Id: inria-00075411
https://inria.hal.science/inria-00075411

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.

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° 1148

Programme 3
Réseaux et Systèmes Répartis

DERIVATION D'UN ALGORITHME REPARTI DE DETECTION DE LA TERMINAISON D'UNE APPLICATION REPARTIE

Jean-Michel HELARY
Michel RAYNAL

Décembre 1989

RR_1148

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042-RENNES CÉDEX
FRANCE
Téléphone: 99 36 20 00
Télex: UNIRISA 950 473 F
Télécopie: 99 38 38 32

Publication Interne n° 505
Décembre 1989 - 16 Pages

Dérivation d'un algorithme réparti de détection de la terminaison d'une application répartie

Jean-Michel HELARY, Michel RAYNAL
IRISA, Campus de Beaulieu, 35042 RENNES CEDEX
e-mail helary@irisa.fr, raynal@irisa.fr.

Résumé

La construction méthodique d'algorithmes répartis est un élément fondamental de la maîtrise du parallélisme; les algorithmes de contrôle en constituent une classe importante, par le rôle qu'ils sont amenés à jouer dans les systèmes répartis. Ces algorithmes reposent souvent sur l'évaluation d'un prédicat global, qui ne peut être mise en œuvre que de manière non simultanée et retardée. Après avoir défini formellement ce type d'évaluation, cet article présente une construction méthodique d'un algorithme réparti de détection de la terminaison (il s'agit là d'un des paradigmes de l'observation dans un contexte réparti). Cette construction est obtenue par étapes successives, chacune de ces étapes constituant un pas de dérivation. L'approche suivie ici constitue donc un pas vers l'extension au contexte réparti d'une méthode analogue à celle proposée par Gries dans le contexte séquentiel pour la dérivation de programmes.

Derivation of a Distributed Algorithm to detect termination of Distributed Application

Abstract

Methodological design of distributed algorithms is of major concern to master parallelism. Distributed control algorithms constitute an important class, due to their role in distributed systems. These algorithms lie often on a global predicate whose evaluation cannot be done instantaneously and concurrently on every site. First, a formal definition of such evaluations is given, then applied to the methodical design of a distributed termination detection algorithm (this problem is a paradigm for observation in a distributed context). The design is performed through a sequence of steps, constituting a derivation. The approach taken here is similar to the one given by Gries in a sequential context.

1 Introduction

Parmi les algorithmes de contrôle des systèmes parallèles il est habituel de distinguer ceux réalisant un service offert à une application de ceux qui réalisent une observation de cette application. L'exclusion mutuelle et la détection de la terminaison sont deux exemples

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

relevant respectivement de ces deux classes. De nombreux algorithmes ont été proposés pour résoudre ces problèmes dans un contexte de parallélisme à mémoire répartie (voir [19] pour l'exclusion mutuelle et [7, 8, 21, 12, 17] pour la terminaison). Ces algorithmes se distinguent par les propriétés qu'ils supposent sur les processus en jeu, la structure et le comportement des canaux de communication, etc. A côté de (et grâce à) ces solutions particulières un certain nombre de blocs de construction de base (*building blocks*,[9]) et de structures de contrôle réparti ont été proposés: les parcours de réseaux [6, 13, 23], les itérations réparties que sont les phases [1] et les vagues [22, 23, 15], ainsi que des outils algorithmiques pour le calcul réparti (jeton [16, 4], marqueurs [18, 3, 11], utilisation de nombres premiers [20]). Il importe donc de savoir composer les éléments de base qui se dégagent de façon à offrir des solutions méthodiques aux problèmes posés [2, 5]. Ceci nécessite un important effort méthodologique dans lequel la dérivation joue un rôle clef. C'est dans cet état d'esprit que se situe cet article, avec comme objet d'étude la détection répartie de la terminaison d'une application répartie. Il s'agit d'un pas dans le sens de la dérivation d'algorithme au sens de Gries [10], mais ici réalisé dans un contexte de répartition.

L'article s'articule autour de deux parties: dans la première (§2), le modèle de calcul est présenté, et le problème de la détection répartie de la terminaison est défini formellement; les difficultés spécifiques à l'évaluation dans un contexte distribué conduisent à donner une spécification formelle des *évaluations non simultanées retardées*. Dans la seconde (§3) la solution est dérivée par une technique d'enrichissements successifs de l'ensemble des événements, du contexte et des règles de comportement; ces enrichissements intègrent des techniques et outils déjà connus (acquittements, vagues et train de vagues) ou plus nouveaux (vague et train de vagues avec contrainte, modèles d'évaluation du §2). Enfin, un exemple de mise en œuvre est proposé, et une expression dans un formalisme à la CSP (mais avec communications asynchrones) est donné en annexe.

2 Le problème de la détection de la terminaison

2.1 Modèle de calcul

L'application dont on veut détecter la terminaison (encore appelée *calcul sous-jacent*) est composée de n processus $P_1, P_2, \ldots, P_n$, ne pouvant communiquer entre eux qu'à l'aide de messages via des canaux de communication. Les canaux sont logiquement orientés, et physiquement bi-directionnels. Le graphe orienté $G = (X, \Gamma)$ qui modélise l'application est supposé fortement connexe. Les canaux sont supposés fiables, nous préciserons la notion de fiabilité plus loin.

Afin d'observer le comportement de l'application, à chaque processus P_i est associé un processus *observateur* (ou *contrôleur*) C_i, capable de capter immédiatement certains états de P_i et d'observer certains événements produits par P_i. Ces contrôleurs peuvent aussi communiquer entre eux, à l'aide de messages - dits *messages de contrôle* - via le même réseau de communication que le calcul sous-jacent [2]. L'ensemble des contrôleurs définit un *algorithme de contrôle*.

Les événements du calcul sous-jacent, produits par un processus P_i et observables par C_i, sont de trois types (tous les messages sont supposés distincts):

PAPIER RÉCUPÉRÉ ET RECYCLÉ

- **émission** d'un message m sur un canal c_{ij} (reliant P_i à P_j), noté $E_{ij}(m)$
- **réception** d'un message m sur un canal c_{ji}, noté $R_{ji}(m)$
- évènement **interne** de P_i, noté I_i

Une exécution d'un algorithme définit une séquence d'états de cet algorithme. A toute exécution du calcul sous-jacent, observée par les contrôleurs, correspond une exécution de l'algorithme de contrôle, et donc une séquence d'états de cet algorithme: $s_0 < s_1 < \ldots < s_k < \ldots$. Lorqu'un événement e, produit ou observable par C_i, a lieu dans un état s_k, l'état suivant est s_{k+1}. Nous supposerons que deux événements distincts ne sont pas simultanés, si bien que l'on peut parler de séquence *événement-état* $s < e < s' < e' < s'' < \ldots$. Un événement e ne peut se produire dans un état s que si un prédicat *pré(e)* (précondition) est vérifié dans l'état s (noté $pré(e)[s]$); on dira encore que e est *possible* dans l'état s. Lorsque le prédicat *pré(e)* reste vrai tant que e n'a pas eu lieu, on dira que e est *continuement possible à partir de* s. Si e se produit dans s, alors un prédicat *post(e)* (postcondition) est vérifié sur l'état suivant s. Cette implication (appelée *règle de comportement*) sera notée de façon classique $\{pré(e)\}e\{post(e)\}$. Enfin, nous supposerons que la sémantique d'exécution du calcul sous-jacent, comme de l'algorithme de contrôle est *vivace*, *non-déterministe* et *équitable*[5]:

- lorsqu'au moins un événement est possible dans un état s, l'un de ces évènements a lieu (*vivacité*), et l'un quelconque (*non-déterminisme*)
- lorsqu'un évènement est continuement possible à partir de s, il aura lieu après un nombre fini de changements d'états (*équité*)

Il en résulte qu'un algorithme est terminé dans un état s si, et seulement si, aucun événement de l'algorithme n'est possible dans l'état s. Enfin nous dirons qu'un évènement a *implique* un évènement b si l'occurence de a est nécessairement suivie de l'occurrence de b après un nombre fini de changements d'états ($a \stackrel{*}{\Rightarrow} b$).

2.2 Comportement de l'application et de son contrôle

Dans tout état du calcul sous-jacent, chaque processus P_i est *actif* ou *passif*; ce statut est observable par le contrôleur associé C_i, qui le capte dans une variable $état_i$. De même, un canal c_{ij} peut être *vide* ou *non vide* (il est vide dans un état s lorsque tous les messages émis par P_i sur c_{ij} avant cet état ont été reçus par P_j avant cet état). Toutefois, aucun des deux processus C_i ou C_j ne peut capter immédiatement ce statut; l'observation partielle de c_{ij} est donc exercée via deux variables: $sent_{ij}$, variable de C_i, représente l'histoire du canal en émission; c'est l'ensemble de tous les messages émis sur c_{ij} depuis le début du calcul sous-jacent; de manière analogue, rec_{ij}, variable de C_j, est l'histoire du canal en réception. Le canal est vide lorsque $sent_{ij} = rec_{ij}$.

Les événements observables produits par un processus P_i définissent des événements de C_i; les règles de comportement du calcul sous-jacent, relatives à ces événements observables, deviennent donc des règles de comportement de l'algorithme de contrôle. Ce sont (règles 1) :

(E) $\{etat_i = actif \wedge m \notin sent_{ij}\}$ $E_{ij}(m)$ $\{etat_i = actif \wedge m \in sent_{ij} \wedge m \notin rec_{ij}\}$
(R) $\{m \in sent_{ji} \wedge m \notin rec_{ji}\}$ $R_{ji}(m)$ $\{etat_i = actif \wedge m \in sent_{ji} \wedge m \in rec_{ji}\}$
(I) $\{etat_i = actif\}$ I_i $\{etat_i = actif \vee etat_i = passif\}$

Nous pouvons définir la *fiabilité* d'un canal: c_{ij} est fiable si, et seulement si, $\forall m$ le prédicat $m \in sent_{ij}$ est stable (pas de perte) et le prédicat $m \notin rec_{ji}$ est vrai tant que l'événement $R_{ij}(m)$ n'a pas eu lieu (pas de création spontanée ou de duplication). Il en résulte que, si les canaux sont fiables et si l'exécution est équitable, toute émission de message implique la réception correspondante.

Le calcul sous-jacent est dit *terminé* lorque tous les processus P_i sont passifs et tous les canaux sont vides. En effet, si le prédicat

$$terminé \equiv \bigwedge_{i \in X} (etat_i = passif) \wedge \bigwedge_{(i,j) \in \Gamma} (sent_{ij} = rec_{ij})$$

est vérifié dans un état s, aucun événement observable n'est possible. Dans ce qui suit, nous utiliserons les notations suivantes :

$$passif_i \equiv (etat_i = passif) \quad , \quad vide_{ij} \equiv (sent_{ij} = rec_{ij})$$

2.3 Détection de la terminaison

Détecter la terminaison d'une application répartie consiste à détecter un état s dans lequel le prédicat *terminé* est vrai. La difficulté de cette détection tient au caractère asynchrone et réparti du calcul sous- jacent [3, 4]:

1. ni C_i ni C_j ne peuvent évaluer instantanément - c'est-à-dire dans un état donné - le prédicat $vide_{ij}$
2. aucun des C_i ne peut évaluer simultanément, c'est-à-dire dans un même état s, l'ensemble des prédicats $passif_i$ et $vide_{ij}$

La première difficulté peut être levée si l'on considère un prédicat a_vide_i, évaluable instantanément par C_i, tel que, dans tout état s de l'algorithme de contrôle:

$$a_vide_i[s] \Rightarrow \bigwedge_{j \in \Gamma(i)} vide_{ij}[s]$$

Si on pose $a_terminé \equiv \bigwedge_{i \in X} passif_i \wedge a_vide_i$ on aura, dans tout état s, $a_terminé[s] \Rightarrow terminé[s]$.

Pour circonvenir la deuxième difficulté, il faut considérer une *évaluation non-simultanée retardée* ($EVNSR$) de $a_terminé$; soit $\mathcal{P} = f(R_1, R_2, \ldots, R_n)$ un prédicat tel que R_i est un prédicat évaluable instantanément par C_i. Une évaluation *non simultanée* de $\mathcal{P}$ est une évaluation $f(R_1[s^{(1)}], \ldots, R_n[s^{(n)}])$, où $s^{(i)}$ désigne l'état dans lequel C_i évalue R_i. Cette évaluation est *retardée* si son résultat est connu d'au moins un processus C_i lorsque l'algorithme de contrôle est dans un état $s \geq \max(s^{(1)}, \ldots, s^{(n)})$. Une telle évaluation sera notée

$$\mathcal{P}[s^{(1)}, \ldots, s^{(n)} \mid s]$$

Une *EVNSR* de $\mathcal{P}$ sera dite *sûre* par rapport à un prédicat T si on a :

$$\mathcal{P}[s^{(1)}, \ldots, s^{(n)} \mid s] \Rightarrow T[s]$$

De manière analogue, $\mathcal{P}$ sera dit *vivace* par rapport à T si

$$T[s] \Rightarrow \exists (s^{(1)}, \ldots, s^{(n)}, s') \text{ avec}$$

$$\left(s \leq \min(s^{(1)}, \ldots, s^{(n)}) \leq \max(s^{(1)}, \ldots, s^{(n)}) \leq s'\right) \wedge \mathcal{P}[s^{(1)}, \ldots, s^{(n)} \mid s']$$

Le problème de la détection de la terminaison consiste donc à:

1. déterminer, pour chaque i, un prédicat a_vide_i
2. déterminer un prédicat $\mathcal{P}$ vivace par rapport à *terminé*
3. donner un procédé d'évaluation non simultanée retardée de $\mathcal{P}$, sûre par rapport à $a_terminé$.

3 L'algorithme de détection de la terminaison

3.1 Les prédicats a_vide_i

La définition de ces prédicats repose sur l'existence d'un protocole permettant à C_i d'apprendre (avec retard) que les messages émis par P_i ont été reçus. Un tel protocole définit de nouveaux événements de l'algorithme de contrôle: $ack_{ij}(m) \equiv$ (C_i *apprend que le message* m *émis par* P_i *vers* P_j *a été reçu*). Le contexte local de C_i peut alors être enrichi avec la variable S_i: ensemble de messages, tel que, dans tout état, S_i contient l'ensemble des messages émis par P_i pour lesquels C_i n'a pas encore appris qu'ils ont été reçus; les règles de comportement de l'algorithme de contrôle deviennent alors (règles 2):

(E) $\{etat_i = actif \wedge m \notin sent_{ij} \wedge m \notin S_i\}$
$E_{ij}(m)$
$\{etat_i = actif \wedge m \in sent_{ij} \wedge m \notin rec_{ij} \wedge m \in S_i\}$
(R) inchangé
(I) inchangé
(A) $\{m \in rec_{ji} \wedge m \in S_i\}$ $ack_{ji}(m)$ $\{m \in rec_{ji} \wedge m \notin S_i\}$

Si les canaux sont fiables et l'exécution équitable, alors toute émission implique l'acquittement correspondant.

On définit alors $a_vide_i \equiv (S_i = \emptyset)$

Proposition 3.1

$$\forall s : \; a_vide_i[s] \Rightarrow \bigwedge_{j \in \Gamma(i)} vide_{ij}[s]$$

Démonstration. Soit m émis sur c_{ij} avant l'état s. D'après la règle (E), une postcondition de $E_{ij}(m)$ est $m \in S_i$. Dans l'état s, on a $S_i = \emptyset$ donc, dans un état $\underline{s} < s$ il y a eu un évènement e tel que $\{m \in S_i\}e\{m \notin S_i\}$; e est donc l'évènement $ack_{ji}(m)$. D'après la règle (A), $m \in rec_{ij}$ est vérifié dans $\underline{s}$, et ce prédicat est une postcondition de $R_{ij}(m)$ (règle (R)), d'où : $R_{ij}(m)$ a eu lieu dans un état $< \underline{s} < s$□

Exemple de protocole: acquittement des messages. Pour chaque i, l'ensemble S_i est perçu via une variable entière $nack_i = card(S_i)$. L'évènement $ack_{ji}(m)$ est la réception d'un message d'acquittement $ack(m)$ envoyé par P_j à P_i lorsque m est reçu. Le protocole est alors défini par les règles de comportement suivantes (règles 2'):

(E) $\{etat_i = actif \wedge m \notin sent_{ij} \wedge nack_i = \alpha \wedge \alpha \geq 0\}$
$E_{ij}(m)$
$\{etat_i = actif \wedge m \in sent_{ij} \wedge m \notin rec_{ij} \wedge nack_i = \alpha + 1\}$
(R) $\{m \in sent_{ji} \wedge m \notin rec_{ji} \wedge ack(m) \notin sent_{ij}\}$
$R_{ij}(m)$
$\{etat_i = actif \wedge m \in sent_{ji} \wedge m \in rec_{ji} \wedge ack(m) \in sent_{ij} \wedge ack(m) \notin rec_{ij}\}$
(I) inchangé
(A) $\{m \in rec_{ji} \wedge ack(m) \in sent_{ji} \wedge ack(m) \notin rec_{ji} \wedge nack_i = \alpha \wedge \alpha > 0\}$
$ack_{ji}(m)$
$\{m \in rec_{ji} \wedge ack(m) \in sent_{ji} \wedge ack(m) \in rec_{ji} \wedge nack_i = \alpha - 1\}$

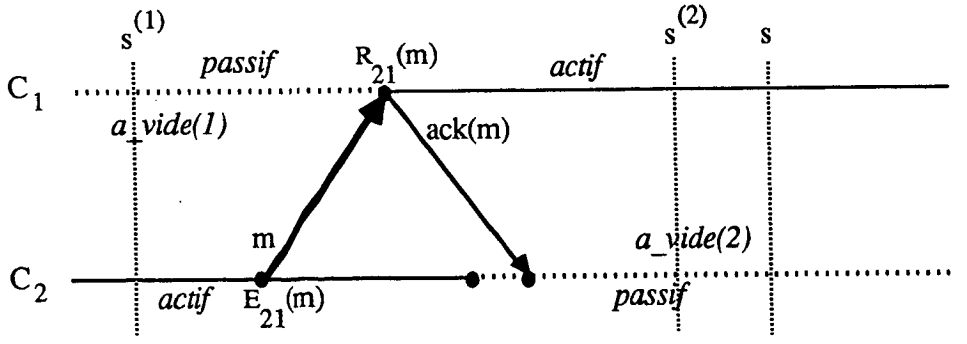
Il est facile de voir que ce protocole satisfait bien les spécifications, et que la relation $nack_i = card(S_i)$ est invariante, donc : $a_vide_i \Leftrightarrow nack_i = 0$.

3.2 Le prédicat $\mathcal{P}$

Une première idée serait de considérer le prédicat

$$a_terminé \equiv \bigwedge_{i \in X} (passif_i \wedge a_vide_i)$$

car chaque processus C_i peut évaluer immédiatement le terme qui le concerne. Mais la sûreté d'une *EVNSR* dépend du comportement du calcul sous-jacent, comme le montre le contre-exemple suivant, mettant en jeu deux processus:



On a: $(passif_1 \wedge a_vide_1)[s^{(1)}], (passif_2 \wedge a_vide_2)[s^{(2)}]$ d'où $a_terminé[s^{(1)}, s^{(2)} \mid s]$ mais par contre, $\neg terminé[s]$. Cette absence de sûreté provient du fait que P_1 n'est pas resté *continuement passif* entre les états $s^{(1)}$ et s. Ceci suggère donc de renforcer (comme dans [10, 5]) le prédicat $a_terminé$ de la manière suivante: soit $\underline{s}$ un état de l'algorithme de contrôle. On définit le prédicat $cp_i(\underline{s})$, évaluable par C_i, tel que, dans un état s:

$$cp_i(\underline{s})[s] \equiv (\underline{s} \leq s) \wedge (\underline{s} \leq \sigma \leq s \Rightarrow passif_i[\sigma])$$

autrement dit, P_i est resté continuement passif entre les états $\underline{s}$ et s. Puis on considère

$$b_terminé[s] \equiv \bigwedge_{i \in X} (\exists \underline{s}^{(i)}) \left(cp_i(\underline{s}^{(i)})[s] \wedge a_vide[\underline{s}^{(i)}] \right)$$

autrement dit: $b_terminé[s]$ est vrai si, et seulement si, pour tout i, il existe un état $\underline{s}^{(i)}$ antérieur à s tel que P_i soit resté continuement passif entre $\underline{s}^{(i)}$ et s, et dans l'état $\underline{s}^{(i)}$ tous les canaux étaient vides.

Lemme 3.1. Pour tout état s: $b_terminé[s] \Rightarrow terminé[s]$

Démonstration. On a déjà, d'après la proposition 3.1:

$$a_vide_i[\underline{s}^{(i)}] \Rightarrow \bigwedge_{j \in \Gamma(i)} vide_{ij}[\underline{s}^{(i)}]$$

Ensuite, par définition des $cp_i(\underline{s}^{(i)})[s]$:

$$\forall i \ (\underline{s}^{(i)} \leq s) \wedge passif_i[s]$$

D'autre part, comme $(\underline{s}^{(i)} \leq \sigma \leq s) \Rightarrow passif_i[\sigma]$, la règle de comportement (E) montre qu'aucune émission n'a eu lieu sur les canaux c_{ij} dans les états σ tels que $\underline{s}^{(i)} \leq \sigma \leq s$, et donc:

$$vide_{ij}[\underline{s}^{(i)}] \Rightarrow vide_{ij}[s] \text{ d'où } \left(\bigwedge_{i \in X} passif_i \wedge \bigwedge_{(i,j) \in \Gamma} vide_{ij} \right)[s]$$

□

Proposition 3.2. Toute *EVNSR* $b_terminé[s^{(1)}, \ldots, s^{(n)} \mid s]$ avec

$$\max(\underline{s}^{(1)}, \ldots, \underline{s}^{(n)}) \leq \min(s^{(1)}, \ldots, s^{(n)}) \leq s$$

est sûre par rapport à $terminé$.

Démonstration. Soit $\underline{s}$ un état tel que $\max(\underline{s}^{(1)}, \ldots, \underline{s}^{(n)}) \leq \underline{s} \leq \min(s^{(1)}, \ldots, s^{(n)})$. D'après la définition de cp, on a $\forall i \ cp_i(\underline{s}^{(i)})[\underline{s}]$ et donc, d'après le lemme 3.1, $terminé[\underline{s}]$. Comme $\underline{s} \leq s$ on a aussi $terminé[s]$□

Proposition 3.3. $b_terminé$ est totalement correct par rapport à $terminé$.

Démonstration. Soit s un état tel que $terminé[s]$, donc:

$$\left(\bigwedge_{i \in X} passif_i \wedge \bigwedge_{(i,j) \in \Gamma} vide_{ij} \right)[s]$$

D'après les règles de comportement (E), (R), (I), aucun événement de type émission, réception ou interne du calcul sous-jacent ne peut avoir lieu dans les états ultérieurs à s, en particulier:

$$(1) \ (s \leq s' \leq \overline{s}) \Rightarrow cp_i(s')[\overline{s}]$$

D'autre part, envisageons deux cas quant à la situation des canaux par rapport au protocole d'acquittement:
1er cas: $\bigwedge_{i\in X} a_vide_i[s]$. Alors, aucun événement de l'algorithme de contrôle ne peut plus avoir lieu après s, qui est donc le dernier état. En prenant $\forall i :, \underline{s}^{(i)} = s$ on a:

$$\bigwedge_{i\in X} cp_i(s)[s] \wedge a_vide_i[s] \text{ soit } \boldsymbol{b_terminé}[s]$$

2ème cas: $\neg \bigwedge_{i\in X} a_vide_i[s]$ Considérons alors un i tel que $\neg a_vide_i[s]$, et $(m \in S_i)[s]$. D'après la règle (E), il existe $j \in \Gamma(i)$ et un événement $E_{ij}(m)$ antérieur à l'état s. Puisque $terminé[s] \Rightarrow (sent_{ij} = rec_{ij})[s]$, la précondition de la réception $R_{ij}(m)$ correspondante n'est pas vérifiée dans l'état s, et donc cet événement a lieu dans un état $s'(m) \geq s$. Il existe donc un état $s"(m) \geq s'(m)$ dans lequel l'événement $ack_{ji}(m)$ a lieu et, d'après la règle (A), dans tout état $> s"(m)$ on a $m \notin S_i$. Si on prend

$$\forall i : \ \underline{s}^{(i)} = \max_{(m\in S_i)[s]}(s"(m)) \quad \text{on a:} \quad a_vide_i[\underline{s}^{(i)}]$$

De plus, $(S_i \neq \emptyset)[s]$ et donc $\underline{s}^{(i)} > s$. D'après (1), dans l'état $\overline{s} = \max(\underline{s}^{(1)}, \ldots, \underline{s}^{(n)})$: $\forall i : \ cp_i(\underline{s}^{(i)})[\overline{s}]$ d'où

$$\bigwedge_{i\in X} (\exists \underline{s}^{(i)} cp_i(\underline{s}^{(i)})[\overline{s}] \wedge a_vide_i[\underline{s}^{(i)}]$$

d'où $b_terminé[\overline{s}]\square$

Exemple de protocole d'évaluation des prédicats cp. L'évaluation d'un prédicat $cp_i(\underline{s})$ débute dans l'état $\underline{s}$. Nous désignerons par deb_obs_i un tel événement; il a lieu dans l'état $\underline{s}$. Une variable booléenne de C_i: $flag_i$, permet d'enregistrer la valeur du prédicat $cp_i(\underline{s})$. Les règles de comportement deviennent alors (règles 3):

(E) identique aux règles 2
(R) {précond. id. aux règles 2} $R_{ji}(m)$ {postcond. des règles 2 $\wedge \neg flag_i$}
(I) identique aux règles 2
(A) identique aux règles 2
(D) {**vrai**} deb_obs_i $\{flag_i = passif_i\}$

Grâce à ces règles, il est facile de voir que, si $\underline{s}$ est le dernier état après lequel un événement deb_obs_i a eu lieu, alors pour tout état $s \geq \underline{s}$ on a: $cp_i(\underline{s})[s] = flag_i[s]$.

3.3 Mise en œuvre d'évaluations non simultanées retardées

3.3.1 Vague

L'évaluation non simultanée retardée d'une fonction $f(x_1, \ldots, x_n)$, où x_i peut être évaluée localement par le processus C_i, repose sur l'existence d'un protocole permettant de visiter une fois et une seule chacun des processus participant à l'évaluation de f, et de déterminer un état dans lequel toutes les visites ont eu lieu. Un tel protocole, appelé *vague*, [22, 13, 23] définit un algorithme de contrôle $\{C_1, C_2, \ldots, C_n, C_0\}$ où C_0 est un processus "abstrait" contrôlant la fin de la vague. Les événements de cet algorithme sont les suivants:

- sur chaque processus C_i : $visite_i$ (lorsque C_i est visité)
- sur le processus C_0: *départ* (aucun C_i n'a été visité)
- sur le processus C_0: *retour* (tous les C_i ont été visités)

Le processus C_0 possède une variable *collecté* (ensemble de valeurs), et les règles de comportement de la vague sont les suivantes:

(DEP)	$\{\mathbf{vrai}\}$	*départ*	$\{collecté = \emptyset\}$
(V)	$\{x_i \notin collecté\}$	$visite_i$	$\{x_i \in collecté\}$
(RET)	$\{collecté = (x_1, \ldots, x_n)\}$	*retour*	$\{\mathbf{vrai}\}$

Il résulte de ces règles [15, 11] et de l'équité que, d'une part

$$départ \stackrel{*}{\Rightarrow} \forall i\ visite_i \stackrel{*}{\Rightarrow} retour$$

et, d'autre part, si sd, $sv^{(i)}$, sr, désignent les états dans lesquels les évènements respectifs $départ, visite_i, retour$ ont lieu, on a:

$$sd \leq \min(sv^{(1)}, \ldots, sv^{(n)}) \leq \max(sv^{(1)}, \ldots, sv^{(n)}) \leq sr$$

De plus, $collecté[sr]$ permet à C_0 de réaliser une $EVNSR$ de $f(x_1, \ldots, x_n)$, à savoir $f(x_1[sv^{(1)}], \ldots, x_n[sv^{(n)}])[sr]$

Nous définissons le protocole *vague avec contraintes*: chaque événement $visite_i$ est préconditionné par un prédicat (local) b_i; la visite n'étant pas nécessairement atomique, il y a lieu de la décomposer en deux événements deb_visite_i, fin_visite_i, et d'attribuer au contexte local de chaque C_i une variable booléenne vp_i (*vague présente*), ce qui donne les nouvelles règles:

(DEP)	$\{\mathbf{vrai}\}$	*départ*	$\{collecté = \emptyset\}$
(DV)	$\{\neg vp_i \wedge x_i \notin collecté\}$	deb_visite_i	$\{vp_i \wedge x_i \notin collecté\}$
(FV)	$\{vp_i \wedge b_i \wedge x_i \notin collecté\}$	fin_visite_i	$\{\neg vp_i \wedge x_i \in collecté\}$
(RET)	$\{\bigwedge_{i \in X} b_i[sfv^{(i)}] \wedge collecté = (x_1, \ldots, x_n)\}$	*retour*	$\{\mathbf{vrai}\}$

où $sfv^{(i)}$ est l'état dans lequel fin_visite_i a lieu.

Dans ce protocole, la propriété de vivacité $\forall i\ deb_visite_i \stackrel{*}{\Rightarrow} fin_visite_i$ n'est plus a priori assurée; en effet, elle dépend de la véracité de b_i, qui peut dépendre à son tour d'autres événements (par exemple du calcul sous-jacent); il en résulte que l'événement fin_visite_i peut ne plus être continuement possible; son occurrence - même dans une exécution équitable - n'est donc pas impliquée par deb_visite_i.

3.3.2 Train de vagues

Le train de vagues est un protocole d'enchaînement séquentiel de vagues (contraintes ou non) permettant de calculer une séquence d'$EVNSR$; c'est donc un schéma itératif

{INV} **tantque** ¬ arret **faire**
{INV ∧¬ arret } vague {INV}
ftantque
{INV ∧ arrêt}

où INV et arrêt sont des prédicats sur l'état de l'algorithme.

Notons $vague(\mu)$ la vague exécutée lors du pas n°μ. Les évènements et les états de cette vague seront eux aussi qualifiés par (μ); en abrégé, $x_i[sfv^{(i)}(\mu)]$ sera noté $x_i[\mu]$ (valeur de la variable x_i dans l'état $sfv^{(i)}(\mu)$ où a lieu l'événement $fin_visite(\mu)$). La séquentialité des vagues est exprimée par les règles de comportement suivantes:

(DEP)	$\{collecté = \emptyset\}$	$départ(\mu)$	$\{collecté = \emptyset\}$
(DV)	$\{\neg vp_i \wedge x_i[\mu] \notin collecté\}$	$deb_visite_i(\mu)$	$\{vp_i \wedge x_i[\mu] \notin collecté\}$
(FV)	$\{vp_i \wedge b_i[\mu] \wedge x_i[\mu] \notin collecté\}$	$fin_visite_i(\mu)$	$\{\neg vp_i \wedge x_i[\mu] \in collecté\}$
(RET)	$\{\bigwedge_{i\in X} b_i[\mu] \wedge collecté = (x_1[\mu], \ldots, x_n[\mu])\}$	$retour(\mu)$	$\{collecté = \emptyset\}$

d'où l'on déduit:

$$\begin{aligned} \forall \mu : \max(sfv^{(1)}(\mu), \ldots, sfv^{(n)}(\mu)) &\leq sr(\mu) \leq sd(\mu+1) \\ &\leq \min(sfv^{(1)}(\mu+1), \ldots, sfv^{(n)}(\mu+1)) \end{aligned}$$

D'autre part, le prédicat $\bigwedge_{i\in X} b_i[\mu]$ est un invariant de l'itération (*invariant de vague*).

3.3.3 Application à la détection de la terminaison

Rappelons qu'il s'agit d'obtenir une *EVNSR* $b_terminé[s^{(1)}, \ldots, s^{(n)} \mid s]$ où

$$b_terminé[s] \equiv \bigwedge_{i\in X} (\exists \underline{s}^{(i)}) \left(cp_i(\underline{s}^{(i)})[s] \wedge a_vide[\underline{s}^{(i)}] \right)$$

avec $\max(\underline{s}^{(1)}, \ldots, \underline{s}^{(n)}) \leq \min(s^{(1)}, \ldots, s^{(n)}) \leq s$. Si l'on considère deux vagues successives $vague(\mu)$ et $vague(\mu+1)$, en posant:

$$\underline{s}^{(i)} = sfv^{(i)}(\mu), \underline{s} = sr(\mu), s^{(i)} = sfv^{(i)}(\mu+1), s = sr(\mu+1)$$

on a bien

$$\max(\underline{s}^{(1)}, \ldots, \underline{s}^{(n)}) \leq \underline{s} \leq \min(s^{(1)}, \ldots, s^{(n)}) \leq \max(s^{(1)}, \ldots, s^{(n)}) \leq s$$

Le train de vagues va alors être construit de telle sorte que son invariant (vérifié à la fin de la vague μ) soit:

$$\text{INV} \equiv \bigwedge_{i\in X} (a_vide_i \wedge passif_i)$$

et la condition d'arrêt, testée à la fin de la vague μ soit:

$$\text{arrêt} \equiv \bigwedge_{i\in X} cp_i(sfv^{(i)}(\mu-1))[sfv^{(i)}(\mu)]$$

La conjonction de l'invariant (exprimé à la fin de la vague $\mu-1$) et de la condition d'arrêt (exprimée à la fin de la vague μ) implique donc $b_terminé[sr(\mu)]$, c'est-à-dire, d'après la proposition 3.2, $terminé[sr(\mu)]$. Le processus abstrait C_0 détecte donc la terminaison du calcul sous-jacent, et cette détection est sûre.

Pour montrer la vivacité, supposons qu'il existe un état s tel que $terminé[s]$. Dans tout état $s' \geq s$ on a $passif_i[s']$; d'autre part, d'après la proposition 3.3, il existe $\underline{s}^{(1)}, \ldots, \underline{s}^{(n)}$ tels que $\forall i : \underline{s}^{(i)} \geq s$ et dans tout état $\geq \underline{s}^{(i)}$ on a $(S_i = \emptyset) \wedge cp_i(\underline{s}^{(i)})$. La contrainte de vague $(S_i = \emptyset) \wedge passif_i$ devient donc stable à partir de l'état $\underline{s}^{(i)}$. Considérons une vague μ telle que

$$sd(\mu) \leq \min(\underline{s}^{(1)}, \ldots, \underline{s}^{(n)}) \text{ et } \max(\underline{s}^{(1)}, \ldots, \underline{s}^{(n)}) < sr(\mu) \text{ ou bien}$$

$$sd(\mu) > \max(\underline{s}^{(1)}, \ldots, \underline{s}^{(n)})$$

Chaque événement fin_visite_i, étant continuement possible à partir de l'état $\underline{s}^{(i)}$, aura lieu dans l'état $sfv^{(i)}(\mu) \geq \underline{s}^{(i)}$, et donc l'événement $retour(\mu)$ aura lieu dans l'état $sr(\mu)$.

De plus, si $\forall i\ \underline{s}^{(i)} < sfv^{(i)}(\mu - 1)$ alors, dans l'état $sr(\mu)$ le prédicat $\bigwedge_{i \in X} cp_i(sfv^{(i)}(\mu - 1))[sfv^{(i)}(\mu)]$ est vérifié, donc $b_terminé[sr(\mu)]$ aussi et la vague μ est la dernière; sinon, une nouvelle vague $(\mu + 1)$ est lancée, et à son retour le prédicat $\bigwedge_{i \in X} cp_i(sfv^{(i)}(\mu))[sfv^{(i)}(\mu + 1)]$ est vérifié, donc $b_terminé[sr(\mu + 1)]$ aussi et la vague $\mu + 1$ est la dernière □.

3.4 Exemple de mise en œuvre du train de vagues: anneau logique

Dans cette mise en œuvre, le processus abstrait C_0 est réalisé par un des processus contrôleurs, soit $C_\alpha (1 \leq \alpha \leq n)$. Les processus C_i sont connectés en *anneau logique*; une vague est matérialisée par un message de contrôle appelé *jeton* qui effectue un tour complet d'anneau à partir de C_α. La variable abstraite *collecté* est réalisée par une valeur booléenne transportée par le jeton.

Sur chaque $C_i (i \neq \alpha)$ l'événement $deb_visite_i(coll)$ correspond à la réception de $jeton(coll)$ (depuis le prédécesseur $pred_i$ sur l'anneau), et $fin_visite_i(coll')$ à l'émission de $jeton(coll')$ (vers le successeur $succ_i$ sur l'anneau). Sur C_α, l'événement *départ* correspond à l'émission de $jeton(\textbf{vrai})$, et l'événement *retour* se confond avec $fin_visite_\alpha(coll)$; ce dernier est soit le dernier événement de l'algorithme (si $coll = \textbf{vrai}$), soit rend possible l'événement *départ* de la vague suivante (si $coll = \textbf{faux}$). Nous supposerons ([13]) qu'il existe un protocole de gestion de l'anneau qui assure que:

$\forall i \neq \alpha : \ fin_visite_i(coll) \stackrel{*}{\Rightarrow} deb_visite_{succ_i}(coll)$
si $\neg coll$ **alors** $fin_visite_\alpha(coll) \stackrel{*}{\Rightarrow} départ_\alpha$
$départ_\alpha \stackrel{*}{\Rightarrow} deb_visite_{succ_\alpha}(\textbf{vrai})$

Enfin, C_α est muni des variables booléennes $lancé_\alpha$ et $arrêt_\alpha$; la première est mise à vrai par le départ de chaque nouvelle vague, et mise à faux par le retour; la seconde enregistre la valeur du jeton au retour d'un vague. Les règles de comportement relatives aux événements *DV, FV, DEP, RET* et avec les protocoles d'évaluation des prédicats a_vide_i et cp_i vus respectivement aux §3.1 et 3.2 sont données ci-dessous:

$$
\begin{array}{ll}
\text{(DEP)} & \{\neg arrêt_\alpha \wedge \neg lancé_\alpha \wedge \neg vp_\alpha\} \\
 & départ_\alpha \\
 & \{\neg arrêt_\alpha \wedge lancé_\alpha \wedge \neg vp_\alpha \wedge collecté\} \\
(\mathrm{DV})_{(\alpha)} & \{collecté = coll \wedge lancé_\alpha \wedge \neg vp_\alpha\} \\
 & deb_visite_\alpha(coll) \\
 & \{collecté = coll \wedge lancé_\alpha \wedge vp_\alpha\} \\
(\mathrm{DV})_{(i\neq\alpha)} & \{collecté = coll \wedge \wedge \neg vp_i\} \\
 & deb_visite_i(coll) \\
 & \{collecté = coll \wedge \wedge vp_i\} \\
(\mathrm{FV})_{(i\neq\alpha)} & \{collecté = coll \wedge vp_i \wedge nack_i = 0 \wedge etat_i = passif \wedge (coll'_i = coll \wedge flag_i)\} \\
 & fin_visite_i(coll'_i) \\
 & \{collecté = coll'_i \wedge \neg vp_i \wedge nack_i = 0 \wedge etat_i = passif \wedge flag_i\} \\
(\mathrm{FV})_{(\alpha)} & \{collecté = coll \wedge vp_\alpha \wedge nack_\alpha = 0 \wedge etat_\alpha = passif \wedge \\
 & \quad (coll'_\alpha = coll \wedge flag_\alpha) \wedge lancé_\alpha\} \\
 & fin_visite_\alpha(coll'_\alpha)(\equiv retour) \\
 & \{collecté = coll'_\alpha \wedge \neg vp_\alpha \wedge nack_\alpha = 0 \wedge etat_\alpha = passif \wedge flag_\alpha \wedge \\
 & \quad \neg lancé_\alpha \wedge (arrêt_\alpha = collecté)\}
\end{array}
$$

En annexe nous donnons une expression correspondante, dans un formalisme identique à celui de CSP, mais dans lequel les primitives de communication ? et ! sont asynchrones. Dans [14] d'autres expressions sont proposées.

4 Conclusion

La construction méthodique d'algorithmes répartis est un élément fondamental de la maîtrise du parallélisme; les algorithmes de contrôle en constituent une classe importante, par le rôle qu'ils sont amenés à jouer dans les systèmes répartis. Ces algorithmes reposent souvent sur l'évaluation d'un prédicat global, qui ne peut être mise en œuvre que de manière non simultanée et retardée. La définition précise et la spécification de propriétés de sûreté et de vivacité de cette notion permettent de mieux cerner les difficultés inhérentes aux schémas d'exécution répartis. Nous avons montré que la décomposition, classique en contexte centralisé, d'un résultat en invariant et condition d'arrêt, pouvait être étendue au contexte réparti, et conduisait à un schéma itératif à deux niveaux: une itération spatiale (*pour tout processus faire ...*), mise en œuvre par une vague, et une itération temporelle (*tant que ¬ arrêt faire vague ftantque*), mise en œuvre par train de vagues. La construction d'une vague (qui constitue un pas de l'itération temporelle) se déduit de la spécification de cette itération: l'invariant, assuré grâce aux contraintes d'"attente" contrôlant la progression de la vague, et la condition d'arrêt, évaluée selon le schéma d'évaluation non simultanée retardée mis en œuvre par l'itération spatiale. Il s'agit donc bien là d'une technique de dérivation au sens de Gries[10], dans le contexte de la répartition.

Bibliographie

[1] J.C. Bermond, J.C. König and M. Raynal. General and efficient decentralized consensus protocols. *Proc. 2nd Int. Workshop on Distr. Alg.*, Amsterdam, july 1987, Springer-Verlag LNCS 312:41-56.

[2] L. Bougé and N. Francez. A compositionnal approach to superimposition. *Proc. ACM Symposium on POPL*, San Diego, 1988.

[3] K.M Chandy and L. Lamport. Distributed snapshots : determining global states in distributed systems. *ACM TOCS*, 3(1):63–75, Feb. 1985.

[4] K.M. Chandy and J. Misra. An example of stepwise refinement of distributed programs : quiescence detection. *ACM Toplas*, 8(3):326–343, July 1986.

[5] K.M. Chandy and J. Misra. *Parallel programming design: a foundation.* Addison Wesley, 1988, 516p.

[6] T. Cheung. Graph traversal techniques and the maximum flow problem in distributed computation. *IEEE Trans. on SE*, SE 9(4):504-512, 1983.

[7] E.W. Dijkstra and C.S. Scholten. Termination detection for diffusing computations. *Inf. Proccessing Letters*, Vol. 11:1–4, 1980.

[8] E.W. Dijkstra, W. H. J. Feijen, and A. J. M. Van Gasteren. Derivation of termination detection algorithm for distributed computation. *Inf. processing Letters*, Vol. 16:217–219, June 1983.

[9] E. Gafni. Perspectives on distributed networks protocols: a case for building blocks. *Proc. IEEE Military Comm. Conf.*, Monterrey, Oct. 1986.

[10] D. Gries. *The science of programming.* Springer-Verlag, 1981, 366p.

[11] J.M. Hélary. Observing global states of asynchronous distributed computations. *3rd Int. Workshop on Distributed Algorithms*, Nice, sept. 1989. Springer-Verlag LNCS 392:124-135.

[12] J. M Hélary, C. Jard, N. Plouzeau, and M. Raynal. Detection of stable properties in distributed applications. In *Proc. 6th annual ACM Symposium on Principles of Distributed Computing*, pages 125–136, Vancouver, August 1987.

[13] J. M. Hélary, M. Raynal. *Synchronisation et contrôle des Systèmes et Programmes Répartis.* Eyrolles, Paris, septembre 1988, 195p.

[14] J.M. Hélary et M. Raynal. Construction méthodique d'un algorithme réparti de détection de la terminaison. *Rapport de recherche IRISA 440*, Rennes, dec. 1988, 15p.

[15] J.M. Hélary et M. Raynal. Un schéma abstrait d'itération répartie; application au calcul des chemins de valeur minimale. *TSI*, vol. 7, n° 3:259-268, mai 1989.

[16] G. Le Lann. Distributed systems: towards a formal approach. *Proc. IFIP Congress*, Toronto, August 1977, pp. 155-160.

[17] F. Mattern. Algorithms for distributed termination detection. *Distributed Computing*, 2:161–175, 1987.

[18] J. Misra. Detecting termination of distributed computation using markers. *Proc. 2d ACM Symp. on PODC*, Montreal, 1983, pp 290-294.

[19] M. Raynal. *Algorithmique du parallélisme: le problème de l'exclusion mutuelle.* Dunod, Paris, 1984, 160 p.

[20] M. Raynal. Prime numbers as a tool to design distributed algorithms. *Inf. Proc. Letters*, Vol 33(1):53-58, 1989.

[21] N. Shavit and N. Francez. A new approach to detection of locally indicative stability. In *Proc. 13th ICALP*, LNCS, 226, pages 344-358, Springer-Verlag, 1986.

[22] F.P. Schneider Paradigms for Distributed Programs. In *Distributed Systems*, LNCS 190: 431-480, Springer-Verlag Ed., (1985).

[23] G. Tel. Total Algorithms. *Proc. Int. Workshop on Parallel and Distributed Algorithms*, Bonas, France, oct. 1988, Cosnard, Quinton, Raynal, Robert ed., North Holland, 1989.

Annexe

La primitive $peek(x)$ permet à un contrôleur C_i de lire l'état de la variable x du processus observé P_i ([2]). Ici, x sera l'état (*passif* ou *actif*) de P_i.

Texte de $C_i (i \neq \alpha)$

$$
\begin{array}{l}
C_i ::[init_i ::arrêt := \textbf{faux}; dernier := \textbf{faux}; nack := 0; \\
\qquad \forall j \in \Gamma^{-1}(i) : acquitt[j] := 0; etat := peek(x); flag := (etat = passif); vp := \textbf{faux}; \\
\quad * [\Box\ \neg arrêt \rightarrow \\
\qquad [p_{i0} :: \Box \textbf{vrai} \rightarrow etat := peek(x) \\
\qquad\ p_{i1} :: \Box_{j \in \Gamma(i)} etat = actif; C_j!m \rightarrow nack := nack + 1 \\
\qquad\ p_{i2} :: \Box_{j \in \Gamma^{-1}(i)} C_j?m \rightarrow etat := actif; flag := \textbf{faux}; acquitt[j] := acquitt[j] + 1 \\
\qquad\ p_{i3} :: \Box_{j \in \Gamma^{-1}(i)} acquitt[j] > 0; C_j!ack \rightarrow acquitt[j] := acquitt[j] - 1 \\
\qquad\ p_{i4} :: \Box_{j \in \Gamma(i)} nack > 0; C_j?ack \rightarrow nack := nack - 1 \\
\qquad\ p_{i5} :: \Box \neg vp; C_{pred}?jeton(collecté) \rightarrow vp := \textbf{vrai}; coll := collecté \\
\qquad\ p_{i6} :: \Box vp \wedge etat = passif \wedge nack = 0; C_{succ}!jeton(coll \wedge flag) \rightarrow vp := \textbf{faux}; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad flag := \textbf{vrai} \\
\qquad\ p_{i7} :: \Box C_{pred}?fini \rightarrow arrêt := \textbf{vrai} \\
\qquad\ ] \\
\qquad p_{i8} :: \Box arrêt \rightarrow \\
\qquad\ [\Box \neg dernier; C_{succ}!fini \rightarrow dernier := \textbf{vrai} \\
\qquad\ \Box dernier \rightarrow SKIP \\
\qquad\ ] \\
\quad\ ] \\
]
\end{array}
$$

Texte de C_α.

$$
\begin{array}{l}
C_\alpha ::[init_\alpha ::arrêt := \textbf{faux}; dernier := \textbf{faux}; nack := 0; \\
\qquad \forall j \in \Gamma^{-1}(\alpha) : \ acquitt[j] := 0; etat; = peek(x); flag := (etat = passif); \\
\qquad vp := \textbf{vrai}; lancé := \textbf{faux}; \\
\quad * [\Box\ \neg arrêt \rightarrow \\
\qquad [p_{\alpha 0} \text{ à } p_{\alpha 5} ::\text{id. à leurs homologues } p_{i0} \text{ à } p_{i5} \\
\qquad\ p_{\alpha 6} :: \Box vp \wedge etat = passif \wedge nack = 0 \wedge lancé \rightarrow arrêt := coll \wedge flag; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad lancé := \textbf{faux}; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad flag := \textbf{vrai} \\
\qquad\ p_{\alpha 7} :: \Box vp \wedge \neg lancé; C_{succ}!jeton(\textbf{vrai}) \rightarrow vp := \textbf{faux}; lancé := \textbf{vrai} \\
\qquad\ ] \\
\qquad p_{\alpha 8} :: \text{analogue à } p_{i8} \\
\qquad\ ] \\
\quad\ ] \\
]
\end{array}
$$

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 497 **LE TRAITEMENT D'EXCEPTIONS - ASPECTS THEORIQUES ET PRATIQUES**
Valérie ISSARNY
80 Pages, Octobre 1989.

PI 498 **IMPLEMENTATION D'UN LANGAGE DE PROGRAMMATION LOGIQUE D'ORDRE SUPERIEUR AVEC MALI**
Pascal BRISSET
28 Pages, Octobre 1989.

PI 499 **QUANTIFICATION IMAGE PAR LA METHODE DE LA VRAISEMBLANCE DU LIEN (A.V.L.) AVEC UN CODAGE PREORDONNANCE**
Israël-César LERMAN, Nadia GHAZZALI
60 Pages, Octobre 1989.

PI 500 **EFFICACITE DE LA METHODE DES PUISSANCES UNIFORMISEES POUR LES CHAINES DE MARKOV RAIDES**
Haîscam ABDALLAH, Raymond MARIE
18 Pages, Octobre 1989.

PI 501 **THE CAUSAL ORDERING ABSTRACTION AND A SIMPLE WAY TO IMPLEMENT IT**
Michel RAYNAL, André SCHIPER
12 Pages, Novembre 1989.

PI 502 **ALGEBRAIC SETS OF TREEVECTORS AND RATIONAL TREE-TRANSDUCTIONS**
Jean-Claude RAOULT
20 Pages, Novembre 1989.

PI 503 **MEMENTO SUR LES ENSEMBLES ORDONNES**
Jean-Claude RAOULT
22 Pages, Novembre 1989.

PI 504 **ON SEQUENTIAL FUNCTIONS**
Boubakar GAMATIE
18 Pages, Novembre 1989.

PI 505 **DERIVATION D'UN ALGORITHME REPARTI DE DETECTION DE LA TERMINAISON D'UNE APPLICATION REPARTIE**
Jean-Michel HELARY, Michel RAYNAL
16 Pages, Décembre 1989.

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