La détection de propriétes stables dans les applications distribuées

Jean-Michel Hélary, Claude Jard, Noël Plouzeau, Michel Raynal

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° 628

LA DÉTECTION DE PROPRIÉTÉS STABLES DANS LES APPLICATIONS DISTRIBUÉES

Jean-Michel HELARY
Claude JARD
Noël PLOUZEAU
Michel RAYNAL

Février 1987

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

La détection de propriétés stables dans les applications distribuées

Detection of stable properties in distributed applications

Jean-Michel **HELARY**, Claude **JARD**
Noël **PLOUZEAU**, Michel **RAYNAL**

IRISA – Université de Rennes 1
Campus de Beaulieu
35042 RENNES CEDEX

Publication Interne n° 342

Janvier 1987
34 pages

groupement C³
COOPÉRATION
CONCURRENCE
COMMUNICATION

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)

Plan du document

1 Introduction
2 Nature des applications concernées (contexte et hypothèses)
- 2.1 Hypothèses générales
  - 2.1.1 Structure des applications
  - 2.1.2 Notations utilisées
  - 2.1.3 Nature et gestion de l'interface entre l'application et le contrôle
- 2.2 Forme générale des propriétés stables
  - 2.2.1 Forme des propriétés
  - 2.2.2 Stabilité de *sg*
- 2.3 Exemple de propriétés stables
  - 2.3.1 Terminaison
  - 2.3.2 Interblocage dû aux communications
  - 2.3.3 Interblocage dû à la compétition

3 Un algorithme distribué et général de détection
- 3.1 Structure de l'algorithme
  - 3.1.1 Un contrôle distribué
  - 3.1.2 Détecter = observer et décider
  - 3.1.3 Hypothèses sur les communications
- 3.2 Contexte d'une entité de contrôle
  - 3.2.1 La prise de mesures
  - 3.2.2 Le prédicat local de détection
- 3.3 L'algorithme
- 3.4 Preuve de bon fonctionnement de l'algorithme
  - 3.4.1 Notations utilisées
  - 3.4.2 L'algorithme ne fait pas de fausse détection
  - 3.4.3 La détection est effectivement assurée
- 3.5 Cas des réceptions non spécifiées

4 Conclusion
Références

La détection de propriétés stables dans les applications distribuées

Résumé

Une propriété stable est une propriété qui, une fois vérifiée, reste vraie. Un certain nombre d'états remarquables d'un système ou d'une application peuvent être caractérisés par de telles propriétés ; c'est par exemple le cas des situations d'interblocage et de terminaison.

Cette étude propose un algorithme général de détection de propriétés stables, dans le contexte des applications distribuées. L'algorithme, lui-même distribué, est indépendant d'une propriété de stabilité particulière ; en ce sens il est générique. Ceci a été rendu possible par l'emploi d'une conception méthodique séparant clairement l'aspect calcul de l'aspect contrôle. L'algorithme est original par son principe de base : il s'appuie sur une approche essentiellement observationnelle.

Abstract

When evaluated to true, *a stable property remains true forever. Such a stable property may characterize important states of a computation. For instance, this is the case of deadlocked or terminated computations.*
In this paper we expose a general algorithm for the distributed detection of stable properties in distributed applications or systems. This distributed algorithm works on every stable property of a fairly general class: in this sense the algorithm is generic. This was made possible by the mean of a methodological approach, with a strong distinction between the computation and control activities in the problem. Moreover, the detection method used by the algorithm is based on activity observation.

1 Introduction

Un grande partie de l'activité des systèmes consiste à détecter des états de l'application qu'ils contrôlent, ces états étant caractérisés par le fait qu'ils vérifient ou ne vérifient pas une propriété donnée. Parmi tous les problèmes de détection, ceux qui s'intéressent aux propriétés de stabilité sont particulièrement importants ; une propriété stable est une propriété globale sur l'état de l'application contrôlée qui, une fois vérifiée, restera vraie pour tous les états ultérieurs dans lesquels passera l'exécution de l'application [Chandy et Lamport 85]. En terme de logique temporelle, une propriété stable *sg* est donc caractérisée par l'invariant suivant (les symboles utilisés sont explicités au paragraphe 2.1.2) :

$$\Box(sg \Rightarrow \bigcirc sg)$$

Les problèmes de détection de la terminaison et de détection de l'interblocage sont des exemples de problèmes de détection de propriétés stables.

Résoudre les problèmes de détection de propriétés stables consiste généralement à introduire un algorithme de contrôle dont le rôle est d'observer l'application et de conclure à bon escient lorsque la stabilité recherchée est atteinte. La propriété de stabilité, exprimée par le prédicat *sg*, porte sur l'état de l'application ; l'algorithme de contrôle doit donc observer cet état sans l'altérer en aucune manière et conclure positivement lorsque l'état observé satisfait *sg*. A ce niveau d'observation, la spécification de l'algorithme de contrôle est donc la suivante ; celui-ci possède une variable de contrôle booléenne *détecte* et doit la gérer de telle façon que :

- Lorsque cette variable présente la valeur *vrai*, l'application vérifie la propriété *sg* ; en terme de logique temporelle ceci est décrit par :

$$\Box(détecte \Rightarrow sg)$$

- Dès que la propriété *sg* est vérifiée, la variable *détecte* doit en rendre compte au bout d'un temps fini ; en d'autres termes :

$$\Box(sg \Rightarrow \Diamond détecte)$$

Ces deux règles, qui décrivent le comportement auquel est assujetti l'algorithme de contrôle sont généralement appelées respectivement *règle d'invariance* et *règle de progression* [Chandy et Misra 86].

L'ensemble des applications dont nous cherchons à détecter des propriétés de stabilité est formé des applications conçues en termes de processus communiquant à l'aide de messages, échangés sur des canaux qui les relient deux à deux.

Le problème majeur provient du fait que dans un contexte réparti l'observation d'un état global et la détection de *sg* ne peuvent être réalisées de façon atomique ; il importe dès lors de mettre en place une collecte d'informations qui permette au(x) détecteur(s) de la propriété *sg* de conclure quant à la valeur de la variable de contrôle *détecte*. Cette collecte, qui est une opération non atomique, doit être effectuée de façon cohérente afin que l'information collectée ne provoque pas ce qu'il est convenu d'appeler une *fausse détection* de la stabilité (une fausse détection donne la valeur *vrai* au booléen *détecte* alors que la stabilité n'est pas atteinte, ce qui met en défaut la règle d'invariance). Plusieurs solutions sont possibles pour effectuer de telles collectes non atomiques. Chandy et Misra [Chandy et Misra 86] ont proposé l'emploi de marqueurs qui parcourent les différents canaux de communication ; un jeton synchronise la progression des marqueurs et collecte l'information recherchée. Il s'agit d'un mécanisme essentiellement impératif : la collecte, lancée par un processus, réalise une opération de visite des canaux de communication et des autres processus du système. Lorsque le processus qui a lancé le jeton le

reçoit à nouveau, il examine les informations récoltées par ce dernier ; si *sg* n'est pas vérifiée, le processus de détection doit lancer une nouvelle opération de visite. Une autre approche basée sur la technique du calcul diffusant est proposée dans [Shavit et Francez 86].

L'approche proposée ici, du point de vue de la collecte et de l'exploitation des informations, se distingue foncièrement de la méthode précédente, pour les raisons suivantes : le (ou les) détecteur(s) de *sg* ne lancent jamais de collectes d'informations ; lorsqu'un processus de l'application modifie son état de manière significative pour la propriété *sg*, il en informe les détecteurs. Le rôle de ceux-ci se limite donc à la prise en compte de ces informations, qui leur servent pour déterminer l'occurrence de la propriété *sg*. L'approche est donc *observationnelle* et non *impérative*. (On retrouve la même différence d'approches dans certains algorithmes d'exclusion mutuelle distribuée, entre [Dijkstra 74] et [Lamport 78] par exemple.)

Le cadre et l'esprit de l'étude étant fixés, nous allons examiner en détail la nature des applications concernées et la forme qu'y revêtent les propriétés de stabilité (§ 2), puis définir un algorithme distribué général de détection de telles propriétés (§ 3) fondé sur une approche entièrement observationnelle.

2 Nature des applications concernées (contexte et hypothèses)

2.1 Hypothèses générales

Nous énonçons tout d'abord les conditions que doivent remplir les applications concernées par l'algorithme de détection des propriétés stables. Ces hypothèses relèvent de plusieurs aspects. Il y a tout d'abord l'aspect structurel et la nature de la communication. Il y a ensuite la définition de l'interface entre l'application et l'algorithme de détection ; celle-ci est composée d'attributs (abstraits à ce niveau de présentation). Ces attributs sont des variables de contrôle qui seront gérées par l'application selon certaines règles et seront exploitées par l'algorithme de détection.

2.1.1 Structure des applications

Comme nous l'avons indiqué dans l'introduction, nous nous intéressons aux applications (que nous appelerons *calcul sous-jacent* ou *calcul contrôlé* dans la suite) structurées en un ensemble fini X de processus séquentiels dont les seules interactions possibles sont l'échange de messages. Un tel échange met en jeu deux processus, à savoir l'émetteur et le récepteur du message, et se fait via un canal de communication.

On se place sous les hypothèses suivantes en ce qui concerne le comportement des canaux :

- On supposera les processus identifiés par les noms $X = (P_1,...,P_n)$.
- Le graphe $G = (X,U)$ des communications est orienté (les canaux sont unidirectionnels) et fortement connexe.
- Les messages sont délivrés au bout d'un temps fini et ne sont ni perdus, ni altérés, ni dupliqués : en d'autres termes les canaux sont fiables.
- Nous ne faisons pas d'hypothèses sur les propriétés de séquencement des messages : les canaux peuvent être *FIFO* ou non *FIFO*.
- L'état d'un canal à un instant donné est défini par l'ensemble des messages qu'il véhicule à cet instant.

2.1.2 Notations utilisées

Dans le paragraphe qui suit, nous utiliserons la logique temporelle pour exprimer les règles que doit respecter le calcul sous-jacent. Dans une formule de logique temporelle (qui porte sur un état de l'application) apparaissent, outre les opérateurs de la logique classique, des opérateurs temporels [Lamport 80, Pnueli 86]. Nous en utiliserons trois, qui sont :

$\Box f$	est vrai dans l'état e considéré si, et seulement si, la formule f est vraie dans tous les états accessibles depuis e.
$\Diamond f$	est vrai dans l'état e considéré si et seulement si la formule f est vraie pour un état accessible depuis e.
$\bigcirc f$	est vrai dans l'état e considéré si et seulement si la formule f est vraie dans tous les états successeurs immédiats de e.

Une formule de logique temporelle portant sur des prédicats sur l'état de l'application permet de lier « dans le temps » les valeurs de vérité de ces prédicats.

2.1.3 Nature et gestion de l'interface entre l'application et le contrôle

Tout processus P_i du calcul sous-jacent est doté d'un attribut booléen sp_i. Un processus dont l'attribut sp_i vaut vrai à un instant donné est dit *passif* à cet instant là. Il est dit *stable* si il reste passif indéfiniment ($\Box sp_i$). Initialement, tous les processus sont actifs (donc instables).

Un message peut être *déstabilisant* ou non.

Tout canal C_{ij}, qui permet à P_i d'envoyer des messages vers P_j, est doté de deux attributs :

- L'attribut sc_{ij} est un booléen qui indique si le canal est *passif* ou non (un canal est *passif* si, et seulement si, il ne contient pas de messages déstabilisants).
- L'attribut fm_{ij} est un booléen qui indique si le canal C_{ij} est fermé ou ouvert ; s'il est fermé P_j ne peut recevoir de messages déstabilisants sur ce canal et P_i ne peut lui en envoyer. Ceci s'exprime, en logique temporelle, par la règle suivante sur l'attribut fm_{ji} :

  (*R0*) $$\Box \bigwedge_{(i,j)\in X^2} (fm_{ji} \Rightarrow sc_{ji})$$

  En d'autres termes, il n'y a pas de réception non spécifiées [West 78]. (Cette hypothèse sera levée au § 3.5.) Initialement les canaux existants sont vides ; les « autres » canaux C_{ij} (qui n'existent pas) sont considérés comme étant fermés en permanence.

Chaque processus P_i de l'application doit respecter à tout instant les règles suivantes :

1. Tout processus P_i peut, à tout instant, faire passer sp_i de *faux* à *vrai* ; P_i devient alors passif et il ne pourra être activé que par la réception d'un message déstabilisant arrivant par l'un de ses canaux d'entrée C_{ji} (ouvert). Ceci constitue la règle de déstabilisation *R1* :

   (*R1*) $$\Box \bigwedge_{i\in X} \left[\bigwedge_{j\in X} sc_{ji} \Rightarrow (sp_i \Rightarrow \bigcirc sp_i) \right]$$

2. D'autre part, un processus ne peut rester passif si il existe des messages déstabilisants en transit sur l'un de ses canaux d'entrée. Ceci constitue la règle de réaction *R2* :

   (*R2*) $$\Box \bigwedge_{i\in X} \left[\Box sp_i \Rightarrow \bigwedge_{j\in X} sc_{ji} \right]$$

3. Un processus passif ne peut émettre de messages déstabilisants sur ses canaux de sortie et ne peut pas ouvrir de canaux entrants. Ceci constitue la règle de stabilité interne *R3*, qui s'exprime ainsi :

   (*R3*) $$\Box \bigwedge_{i\in X} \left[sp_i \Rightarrow \bigwedge_{j\in X} \left[(sc_{ij} \Rightarrow \bigcirc sc_{ij}) \wedge (fm_{ji} \Rightarrow \bigcirc fm_{ji}) \right] \right]$$

2.2 Forme générale des propriétés stables

2.2.1 Forme des propriétés

Une propriété globale met en jeu un certain nombre de processus et de canaux du calcul sous-jacent. Une telle propriété *sg* est stable si, comme nous l'avons indiqué, une fois vérifiée elle reste vraie. Dans le contexte considéré, formé de processus et de canaux, une propriété stable sur un sous-ensemble S des processus indique :

- que les processus de S doivent être passifs,
- que les canaux qui les relient doivent être passifs,
- et que les canaux qui relient les processus de $X - S$ aux processus de S doivent être fermés (puisque une fois atteinte la stabilité ne peut être remise en cause, les processus de $X - S$ ne doivent pas pouvoir déstabiliser le sous-ensemble S des processus).

Formellement, ceci s'exprime par :

$$sg(S) \equiv \bigwedge_{i \in S} \left[sp_i \wedge \bigwedge_{j \in S} sc_{ji} \wedge \bigwedge_{j \in X-S} fm_{ji} \right]$$

2.2.2 Stabilité de *sg*

La forme de $sg(S)$ étant donnée, il s'agit de démontrer sa stabilité, c'est-à-dire :

$$\Box \left[sg(S) \Rightarrow \bigcirc sg(S) \right]$$

Une fois cette formule démontrée, il est facile de voir qu'elle implique bien la stabilité des processus de S :

$$\Box \left[sg(S) \Rightarrow \bigwedge_{i \in S} \Box sp_i \right]$$

La stabilité de $sg(S)$ peut être établie lorsque l'application obéit aux règles de comportement *R0*, *R1* et *R3*. (La règle de comportement *R2* qui indique que les processus doivent réagir aux messages déstabilisants ne sera utile que pour prouver la règle de progression de l'algorithme de contrôle.)

Démonstration

$$sg(S) \Rightarrow \bigwedge_{i \in S} \left[sp_i \wedge \bigwedge_{j \in S} sc_{ji} \wedge \bigwedge_{j \in X-S} fm_{ji} \right]$$

Ce qui est équivalent à :

$$sg(S) \Rightarrow \bigwedge_{i \in S} \left[sp_i \wedge \bigwedge_{j \in S} sc_{ij} \wedge \bigwedge_{j \in X-S} fm_{ji} \right]$$

En utilisant les règles *R1* et *R3*, et la déduction :

$$\left[(a \Rightarrow b \wedge c) \wedge (c \Rightarrow d) \right] \Rightarrow (a \Rightarrow b \wedge c \wedge d)$$

on a :

$$sg(S) \Rightarrow \bigwedge_{i \in S} \left[sp_i \wedge \bigwedge_{j \in S} (sc_{ij} \Rightarrow \bigcirc sc_{ij}) \wedge \bigwedge_{j \in S} sc_{ij} \wedge (sp_i \Rightarrow \bigcirc sp_i) \wedge \bigwedge_{j \in X-S} fm_{ji} \wedge \bigwedge_{j \in X-S} (fm_{ji} \Rightarrow \bigcirc fm_{ji}) \right]$$

Comme $\left[a \wedge (a \Rightarrow b) \right] \Rightarrow b$:

$$sg(S) \Rightarrow \bigwedge_{i \in S} \left[\bigcirc sp_i \wedge \bigwedge_{j \in S} \bigcirc sc_{ij} \wedge \bigwedge_{j \in X-S} \bigcirc fm_{ji} \right]$$

en utilisant la distributivité de $\bigcirc$ sur $\wedge$:

$$sg(S) \Rightarrow \bigcirc \bigwedge_{i \in S} \left[sp_i \wedge \bigwedge_{j \in S} sc_{ij} \wedge \bigwedge_{j \in X-S} fm_{ji} \right]$$

c'est-à-dire :

$$sg(S) \Rightarrow \bigcirc sg(S)$$

■

2.3 Exemples de propriétés stables

Nous donnons trois exemples de propriétés stables couramment rencontrées dans les systèmes ou les applications réparties. Nous indiquons pour chacune des propriétés étudiées l'interprétation des variables de contrôle sp_i, sc_{ij} et fm_{ji}. Il est facile de constater que ces trois applications obéissent bien aux règles de comportement *R0* à *R3*.

2.3.1 Terminaison

Depuis l'algorithme de Francez [Francez 80], de nombreux algorithmes de détection de la terminaison d'un calcul distribué ont été publiés, entre autres : [Dijkstra et Scholten 80, Dijkstra et al. 83, Chandy et Misra 85, Apt 85]. Dans le schéma proposé, la possibilité de communication depuis le processus P_j vers le processus P_i est représentée par le canal C_{ji}. Dans le cas général, un processus P_i peut attendre de manière non-déterministe sur plusieurs de ses canaux d'entrée [Hoare 78] : il sera activé par le premier message reçu sur l'un de ces canaux. Tout message est déstabilisant. Un processus peut être *actif* ou *passif* (il a alors terminé son traitement ou attend un message). Nous avons alors :

- $[sp_i = vrai]$: P_i est passif.
- $[fm_{ji} = vrai]$: le canal C_{ji} est fermé, c'est-à-dire :
  - ou bien ce canal n'existe pas,
  - ou bien P_i n'attend pas de messages de P_j.
- $[sc_{ij} = vrai]$: le canal C_{ij} ne contient pas de messages.

La terminaison de l'application concerne tous les processus, on a donc ici $S = X$. Le prédicat $sg(X)$ s'interprète alors : tous les processus sont passifs et les canaux sur lesquels ils attendent des messages sont vides.

2.3.2 Interblocage dû aux communications

Les possibilités de communication entre les processus d'une application peuvent conduire à des situations d'interblocage [Chandy et al. 83, Natarajan 86]. L'ensemble des canaux, l'attente non-déterministe d'un message et l'interprétation des attributs sont les mêmes qu'au paragraphe 2.3.1.

Une situation d'interblocage est caractérisée par l'existence d'un ensemble S de processus (ensemble non fixé *a priori*) qui sont soit passifs, soit en attente de messages et qui ne peuvent être réactivés.

L'interprétation du prédicat *sg* est la suivante. Lorsque *sg*(*S*) est vrai, il définit un ensemble de processus *S* tel que chacun des éléments de *S* est passif, les canaux qui les interconnectent sont soit vides soit fermés, et les canaux qui relient des processus n'appartenant pas à *S* aux processus de *S* sont fermés. Tous les processus de *S* sont alors stables.

Remarques

1. Comme nous l'avons indiqué, l'ensemble *S* des processus interbloqués n'est pas défini a priori, l'algorithme de détection devra donc le calculer et le donner en résultat. On peut bien sûr figer *S* ; l'algorithme calculera alors *sg*(*S*). Dans le cas où *S* est fixé égal à *X*, on retrouve le problème de la détection de la terminaison.
2. Lorsque pour un ensemble *S* le prédicat *sg*(*S*) est vérifié, il est facile de voir que $\forall S' \subset S$ tous les processus de *S'* sont stables : la propriété de stabilité est monotone. Cela peut être mis à profit pour détecter la stabilité d'un processus donné a priori, par exemple P_k : il suffit d'avoir $sg(S) \wedge k \in S$.

2.3.3 Interblocage dû à la compétition

Le nombre limité de ressources et l'obligation d'y accéder en exclusion mutuelle peuvent amener un interblocage des processus. On parle alors d'interblocage dû à la compétition [Chandy et Misra 82].

Un processus qui demande des ressources émet des messages de requête vers les processus qui possèdent ces ressources, et se bloque en attendant l'arrivée de messages d'allocation. Un processus ne libère ses ressources qu'après les avoir obtenues et utilisées : la préemption est interdite. Les messages déstabilisants de l'application sont les messages d'allocation. Les canaux C_{ij} concernés par la détection de l'interblocage sont ceux sur lesquels transitent les messages d'allocation. L'interprétation des attributs est la suivante.

- $[sp_i = vrai]$ ≡ P_i est bloqué en attente de ressources.
- $[fm_{ji} = vrai]$ ≡ P_i n'a pas demandé de ressources à P_j.
- $[sc_{ji} = vrai]$ ≡ il n'y a pas de message d'allocation en transit de P_j vers P_i (un tel message s'il existait serait relatif à la ressource qu'avait précédemment P_j).

Un processus qui reçoit un message d'allocation ferme le canal correspondant et fait passer sp_i à faux s'il a reçu tous les messages d'allocation.

L'interprétation de *sg* sur un ensemble *S* (non défini a priori) est alors la suivante : tous les processus de *S* sont bloqués en attente de ressources possédées par des processus de *S* et il

n'y a pas de messages d'allocation en transit entre ces processus ; ceci caractérise bien une situation d'interblocage.

3 Un algorithme distribué et général de détection

3.1 Structure de l'algorithme

3.1.1 Un contrôle distribué

La spécification externe de l'algorithme de détection a été donnée dans l'introduction. Cet algorithme gère une variable booléenne *détecte* qui, initialisée à *faux*, est telle que pour une propriété stable *sg* donnée on a :

$$\Box(détecte \Rightarrow sg) \wedge \Box(sg \Rightarrow \Diamond détecte)$$

Plusieurs mises en œuvre sont possibles. Si le support de la communication entre les processus de l'application est une mémoire commune, une mise en œuvre évidente s'impose : la mémoire supporte les variables définies par l'interface et l'algorithme de contrôle peut calculer *sg* de manière atomique grâce à l'exclusion mutuelle entre les accès à la mémoire ; le résultat de ce calcul donne sa valeur exacte à *détecte*.

Nous nous intéressons au cas où l'application et l'algorithme de détection sont tous deux distribués. Un tel contexte rend compte d'une plus grande généralité, notamment lorsque chacun des processus de l'application est associé à un site distinct.

3.1.2 Détecter = observer et décider

Sur le plan structurel, l'algorithme de détection est formé de n entités de contrôle EC_i. Chacune de ces entités est associée à un processus P_i (également appelé *site*) du calcul sous-jacent. L'activité d'une entité de contrôle EC_i se décompose en deux parties :

1. La première composante concerne la prise de mesures de l'activité du processus P_i, du point de vue de la détection des propriétés stables. La présence de cette composante (que nous désignerons par $EC(M)_i$) est nécessaire dans chacune des entités EC_i.
2. La seconde composante concerne la détection proprement dite de la stabilité. Cette composante (que nous désignerons par $EC(D)_i$) se retrouve dans chacune des entités EC_i chargées de détecter si la stabilité a été atteinte ; selon le désir du concepteur, il y a donc entre une et n entités $EC(D)_i$ dans le système.

3.1.3 Hypothèses sur les communications

Les entités EC_i communiquent, comme le calcul sous-jacent, par échange de messages. Il va de soi que les hypothèses faites par l'algorithme de contrôle sur les canaux de communication ne peuvent pas être plus fortes que celles faites par le calcul sous-jacent. En conséquence, les canaux qui servent au contrôle (qui peuvent être différents de ceux utilisés par l'application) sont fiables (pas de pertes ni d'altérations de messages) mais ne préservent pas nécessairement l'ordre de transmission. L'intérêt de cette dernière hypothèse est très manifeste dans le contexte d'algorithmes de contrôle ; le même algorithme fonctionne que les canaux déséquencent ou non les messages : l'algorithme de détection ne nécessite pas de contrôle supplémentaire destiné à assurer le respect du séquencement.

Nous supposons qu'il existe un service de diffusion dans un tel environnement (le déséquencement est possible), permettant à chacune des entités $EC(M)_i$ de diffuser un message vers chacune des entités $EC(D)_j$ (y compris à $EC(D)_i$ si elle existe) [Segall 83]. Ce service de diffusion peut dupliquer les messages de contrôle. Nous verrons au paragraphe 3.3 que l'algorithme proposé tolère la duplication des messages de contrôle. La mise en œuvre d'un tel service de diffusion est très facile.

3.2 Contexte d'une entité de contrôle

3.2.1 La prise de mesures

Détecter la stabilité nécessite une perception cohérente des attributs de l'interface de contrôle : sp_i pour tout i, sc_{ij} et fm_{ji}, pour tout (i,j) tel que le canal C_{ij} existe.

L'entité EC_i a une vision toujours exacte (c'est-à-dire non retardée) des attributs sp_i et fm_{ji} ; sp_i donne l'état du processus P_i et, lorsque sp_i vaut *vrai*, les fm_{ji} indiquent quels sont les canaux qui ne peuvent pas déstabiliser P_i.

Les entités EC_i qui ont une composante de détection ont besoin de connaître d'une part les valeurs des attributs sp_j et fm_{mj} des autres processus P_j et d'autre part les attributs sc_{kl} des canaux. Il est donc nécessaire que les entités EC_i s'échangent de l'information (ceci sera réalisé au moyen de messages de contrôle du type *ctl_st* diffusés par chaque $EC(M)_i$ vers les $EC(D)_j$). En ce qui concerne les attributs sp_j et fm_{mj}, l'acquisition de cette connaissance peut être facilement réalisée par la diffusion de leurs valeurs depuis EC_j ; par contre, en ce qui concerne les attributs sc_{kl} il est nécessaire, comme ces attributs ne sont observables directement par aucun des processus, que les entités EC_i coopèrent de façon à en avoir une perception correcte. Pour

cela, chaque EC_i est doté de structures de données destinées à matérialiser localement sa perception de l'état global nécessaire à la détection. En ce qui concerne les canaux ce sont :

obs_émis$_i$ **tableau [1..n][1..n] de naturel, initialisé à 0.**
L'élément *obs_émis*$_i$[*k*][*l*] représente le nombre total de messages déstabilisants émis par P_k vers P_l, tel que le perçoit EC_i.

obs_reçus$_i$ **tableau [1..n][1..n] de naturel, initialisé à 0.**
L'élément *obs_reçus*$_i$[*l*][*k*] représente le nombre total de messages déstabilisants reçus par P_l en provenance de P_k, tel que le perçoit EC_i.

Les deux variables suivantes sont introduites pour capter l'état des processus :

participants$_i$ **tableau [1..n] de booléen, initialisé à faux.**
L'élément *participants*$_i$[*k*] vaut *vrai* lorsque $EC(D)_i$ a reçu au moins un message de $EC(M)_k$ signifiant que P_k est devenu passif au moins une fois.

obs_fermés$_i$ **tableau [1..n][1..n] de booléen, initialisé à faux.**
L'élément *obs_fermés*$_i$[*l*][*k*] vaut *vrai* si et seulement si P_l a fermé le canal permettant à P_k de lui envoyer des messages déstabilisants, tel que le perçoit $EC(D)_i$ (EC_i perçoit alors l'attribut fm_{kl} du canal C_{kl} comme ayant la valeur *vrai*).

Les quatre variables locales de EC_i : *obs_émis*$_i$, *obs_reçus*$_i$, *participants*$_i$ et *obs_fermés*$_i$ sont nécessaires à la composante $EC(D)_i$: la perception qu'a $EC(D)_i$ du booléen *détecte* va s'exprimer au moyen de ces variables.

Afin que ces variables de détection représentent un état cohérent du système, chacune des entités EC_i doit observer le comportement du processus P_i auquel elle est associée. Les deux variables suivantes permettent à $EC(M)_i$ d'observer, en la mesurant, l'activité de P_i :

instables_émis$_i$, *instables_reçus*$_i$
tableau [1..n] de naturel, initialisé à 0.
Les variables *instables_émis*$_i$[*j*] et *instables_reçus*$_i$[*k*] permettent à EC_i de compter le nombre de messages déstabilisants d'une part émis par P_i vers P_j et d'autre part reçus par P_i en provenance de P_k.

De plus, $EC(M)_i$ possède à tout moment la vision exacte de ses canaux d'entrée C_{ji}, qui sont soit ouverts ($\neg fm_{ji}$), soit fermés (fm_{ji}). Cette connaissance est captée par la variable *canaux_fermés*$_i$ locale à $EC(M)_i$.

canaux_fermés$_i$
tableau [1..n] de booléen.
A tout moment, *canaux_fermés*$_i$[*j*] vaut *vrai* si et seulement si le canal allant

de P_j à P_i est fermé (c'est-à-dire si et seulement si fm_{ji} a la valeur *vrai*) ; P_i ne peut être déstabilisé via un tel canal.

Les messages de contrôle *ctl_st* permettent aux entités EC_i de coopérer de façon à ce que les perceptions qu'elles ont de l'état global restent cohérentes. Un tel message a la forme suivante :

ctl_st(id,copie_émis,copie_reçus,copie_fermés)

id désigne l'identité *i* de l'entité $EC(M)_i$ expédiant ce message,

copie_émis est une copie de la valeur du compteur $instable\_émis_{id}$, prise au moment de l'émission de *ctl_st*,

copie_reçus est une copie de la valeur du compteur $instable\_reçus_{id}$, prise au moment de l'émission,

copie_fermés est une copie de la valeur du tableau $canaux\_fermés_{id}$, prise au moment de l'émission.

En résumé, le contexte d'une entité EC_i, associée au processus P_i du calcul sous-jacent, est composé de :

- en ce qui concerne la prise de mesures du comportement de P_i : $instables\_émis_i$, $instables\_reçus_i$ et $canaux\_fermés_i$ (ces informations sont communiquées aux entités de détection),
- en ce qui concerne la représentation de l'état global : $obs\_émis_i$, $obs\_reçus_i$, $obs\_fermés_i$, $participants_i$ (ces variables ne sont présentes que dans les composantes $EC(D)_i$).

3.2.2 Le prédicat local de détection

Une entité EC_i chargée de détecter la stabilité de la propriété *sg* reçoit des messages *ctl_st* qui lui permettent de mettre à jour les variables qui matérialisent sa perception de l'état global (du point de vue de la propriété recherchée). La vision locale $détecte_i$ du booléen *détecte* s'exprime avec ces variables ; sa forme est la suivante :

$$\begin{aligned} détecte_i \equiv & \exists S \;/\; S \neq \phi \\ & \wedge\ \forall j \in S : participants_i[j] \wedge \{k \in X \;/\; \neg obs\_fermés_i[j][k]\} \subset S \\ & \wedge\ \forall (j,k) \in S^2 \;:\; obs\_émis_i[j][k] = obs\_reçus_i[k][j] \end{aligned}$$

Nous allons présenter maintenant l'algorithme de détection ; outre la prise de mesure, il précise quand sont envoyés les messages de contrôle *ctl_st*. La preuve consistera à montrer que $détecte_i$ est une mise en œuvre correcte du booléen *détecte*.

3.3 L'algorithme

Nous décrivons le comportement d'une entité EC_i dans le cadre général, c'est-à-dire à la fois dans son rôle de prise de mesures (qu'elle doit toujours jouer) et dans son rôle de détecteur de la stabilité (au moins une entité le joue).

Chacune des procédures définies ci-dessous est atomique. Les trois premières réalisent la prise et la diffusion des mesures relatives à l'observation du processus P_i, la quatrième réalise la mise à jour de la représentation locale à EC_i de l'état global et réalise aussi la détection de la stabilité. Le texte de l'algorithme est donné à la figure 1. Nous utilisons les notations de comparaison entre vecteurs suivantes :

pour a,b : **tableau**[1..n] de **naturel**
$a = b$ signifie : $\forall j \in [1,n] : a[j] = b[j]$
$a < b$ signifie $\forall j \in [1,n] : a[j] \leqq b[j] \wedge \exists j \in [1,n] : a[j] < b[j]$

pour a,b : **tableau**[1..n] de **booléen**
$a \subset b$ signifie $\forall j \in [1,n] : a[j] \Rightarrow b[j] \wedge \exists j \in [1,n] : \neg a[j] \wedge b[j]$

Dans la procédure *recevoir_ctl_st*, le test qui permet de savoir si le message de contrôle reçu n'est pas caduc (il aurait alors été dépassé par un autre message de contrôle plus récent que lui) est effectué à l'aide des champs *copie_reçus* et *copie_fermés*. En effet, si P_j émet deux messages de contrôle *ctl_st*, c_1 puis c_2, alors dans l'intervalle séparant l'émission de c_1 et celle de c_2 deux situations ont pu avoir lieu :

1. Le processus P_j est resté continuellement passif, et dans ce cas c_2 a été émis parce que P_j a fermé un canal ; comme il n'a pu, dans cet intervalle, ni ouvrir de canal, ni recevoir, ni émettre de message déstabilisant, les champs des messages c_1 et c_2 vérifient :

$$copie\_fermés_1 \subset copie\_fermés_2 \wedge copie\_reçus_1 = copie\_reçus_2$$

2. Le processus P_j a été activé et dans ce cas il a nécessairement reçu au moins un message déstabilisant venant d'un autre processus ; on a alors :

$$copie\_reçus_1 < copie\_reçus_2$$

Le test permettant à une entité $EC(D)_i$ de valider un message de contrôle est donc le suivant :

$$obs\_reçus_i[j] < copie\_reçus \vee (obs\_reçus_i[j] = copie\_reçus \wedge obs\_fermés_i[j] \subset copie\_fermés)$$

```
proc déstabiliser(j:1..n);
  début
    -- Cette procédure, qui ne peut être exécutée que lorsque sp_i a la valeur faux,
    -- prend en paramètre l'identité du processus P_j auquel P_i envoie un
    -- message déstabilisant (cf. règle R2 et règle R3).
    instable_émis_i[j] := instable_émis_i[j] + 1;
  fin;

proc recevoir_déstabilisant(j:1..n);
  début
    -- Cette procédure est exécutée lorsque P_i reçoit un message déstabilisant
    -- de P_j. La gestion de sp_i par l'application assure
    -- qu'alors sp_i peut passer à faux (cf. règle R1).
    instable_reçus_i[j] := instable_reçus_i[j] + 1;
  fin

proc devenir_passif;
  début
    -- Cette procédure est exécutée lorsque sp_i passe de faux à vrai et chaque fois que
    -- P_i ferme un canal d'entrée (cf. règle R3).
    diffuser ctl_st(i,instable_émis_i, instable_reçus_i, canaux_fermés_i
    fin

proc recevoir_ctl_st(j,copie_émis, copie_reçus, copie_fermés);
    début
      -- Cette procédure est exécutée lors de la réception d'un message
      -- de contrôle ctl_st.
      participants_i[j] := vrai;
      si ( copie_reçus > obs_reçus_i[j]
           ∨ (copie_reçus = obs_reçus_i[j] ∧ obs_fermés_i[j] ⊂ copie_fermés))
        alors
          -- Le message reçu n'est pas caduc.
          obs_émis_i[j] := copie_émis;
          obs_reçus_i[j] := copie_reçus;
          obs_fermés_i[j] := copie_fermés;
          si détecte_i
           alors
              -- La stabilité est détectée : sg(S) est vérifiée.
          fsi;
      fsi;
    fin;
```

Figure 1. Texte de l'algorithme

Remarque

La mise en œuvre par une entité $EC(D)_i$ du calcul du prédicat $détecte_i$ est un calcul local à $EC(D)_i$ et qui relève de l'algorithmique séquentielle. Lors de la réception d'un message *ctl_st*, $EC(D)_i$ considère le graphe $A = (VS, \Gamma)$, avec :

$$VS = \{j \ / \ participants_i[j]\}$$
$$\Gamma = \{(j,k) / \neg obs\_fermés_i[k][j]\}$$

En terme du graphe A, on peut exprimer le booléen $détecte_i$ sous la forme suivante :

$$détecte_i \equiv$$
$$\exists S,\ S \neq \phi \wedge S \subset VS \wedge \Gamma(S) \subset S$$
$$\wedge \forall (j,k) \in S^2 : obs\_émis_i[j][k] = obs\_reçus_i[k][j]$$

Comme tout graphe, A possède p composantes fortement connexes terminales $F_1, \ldots, F_p$ $(p \geq 1)$. On a alors :

$$détecte_i \equiv \exists \alpha \in [1,p] : card(F_\alpha) = 1$$
$$\vee\ card(F_\alpha) \geq 2 \wedge \forall (j,k) \in F_\alpha \times F_\alpha : obs\_émis_i[j][k] = obs\_reçus_i[k][j]$$

et d'autre part, la définition d'une composante fortement connexe terminale montre que l'on a :

$$S \neq \phi \wedge S \subset VS \wedge \Gamma(S) \subset S$$
$$\Leftrightarrow\ S \text{ contient au moins une composante fortement connexe terminale } F_\alpha$$

(Dans le cas où $card(F_\alpha) = 1$, la condition d'égalité sur les tableaux $obs\_émis_i$ et $obs\_reçus_i$ est sans objet.) Les composantes fortement connexes terminales d'un graphe peuvent être facilement obtenues, par exemple par l'algorithme d'exploration de Tarjan [Tarjan 72].

Nous allons maintenant donner une preuve de bon fonctionnement de cet algorithme de détection de propriétés stables.

3.4 Preuve de bon fonctionnement de l'algorithme

3.4.1 Notations utilisées

Comme dans la logique temporelle, nous supposerons qu'il existe un référentiel de temps global ; ce temps est abstrait : il n'est basé que sur le fait que l'émission d'un message précède sa réception et que dans un même processus deux événements ne peuvent avoir lieu simultanément. On note $f(t)$ le fait qu'un prédicat f est vrai à la date t.

Nous désignerons par $M(j,k)$ l'ensemble des messages déstabilisants émis par P_j vers P_k jusqu'à un instant t et, pour $m \in M(j,k)$:

$dec_j(m,k)$ date d'émission de m par P_j

$drc_k(m,j)$ date de réception de m par P_k

D'autre part, nous considérons un processus détecteur $EC(D)_i$ pour lequel $détecte_i$ devient vrai à la date dt_i. Soit $VS_i = \{j \ / \ participants_i[j]\}$ à la date dt_i. Tout P_j appartenant à VS_i a diffusé au moins un message *ctl_st*, reçu par P_i avant dt_i. L'ensemble des messages *ctl_st* reçus par P_i depuis P_j avant dt_i est donc non vide et strictement ordonné par les dates d'émission. Nous désignerons alors par :

$DCS_i(j)$ parmi ces messages, celui qui possède la plus grande date d'émission,

des_j sa date d'émission par P_j,

$drs_i(j)$ sa date de réception par P_i.

La relation suivante est donc vérifiée : $des_j < drs_i(j) \leqq dt_i$ Tout message *ctl_st* diffusé par P_j après des_j arrivera donc à P_i après dt_i.

3.4.2 L'algorithme ne fait pas de fausse détection

Il s'agit de montrer que lorsqu'une entité $EC(D)_i$ obtient la valeur *vrai* pour $détecte_i$, la propriété de stabilité de *sg* est effectivement atteinte ; en d'autres termes que l'on a :

$$\forall i \ \Box(détecte_i \Rightarrow sg)$$

Nous nous plaçons par conséquent sous l'hypothèse où $détecte_i$ est vrai à une date dt_i. L'ensemble S réfère alors l'ensemble non vide de processus attesté par $détecte_i$, et $j \in S$ signifie que $EC(D)_i$ perçoit l'appartenance de P_j à S à la date dt_i. On a alors :

$$j \in S \Rightarrow j \in VS_i \Rightarrow \left[(la\ date\ des_j\ est\ définie\ et\ vérifie\ :\ des_j < dt_i) \wedge\ sp_j(dep_j)\right]$$

Lemme 3.4.2 – 1 : lemme sur « les égalités à dt_i »

$\forall j \in VS_i$ $\quad obs\_émis_i[j](dt_i) = instable\_emis_j(des_j)$

$\quad obs\_reçus_i[j](dt_i) = instable\_reçus_j(des_j)$

$\quad obs\_fermés_i[j]\,dt_i = canaux\_fermés_j(des_j)$

Démonstration

Le message $DCS_i(j)$, diffusé par EC_j à la date des_j, transmet les valeurs, à cette date, des vecteurs $instable\_émis_j$, $instable\_reçus_j$ et $canaux\_fermés_j$. Ces valeurs sont prises en compte, à la date de réception $drs_i(j)$ de $DCS(j)$, par $EC(D)_i$ puisque $DCS_i(j)$ n'est pas caduc : en effet il véhicule les informations les plus à jour (voir sa définition). En conséquence (figure 2), un message ctl_st émis par EC_j avant des_j et reçu entre $drs_i(j)$ et dt_i est forcément caduc (cf. le test d'acceptation des messages de contrôle dans l'algorithme). Le dernier message de contrôle émis par EC_j est donc le dernier considéré par $EC(D)_i$. ■

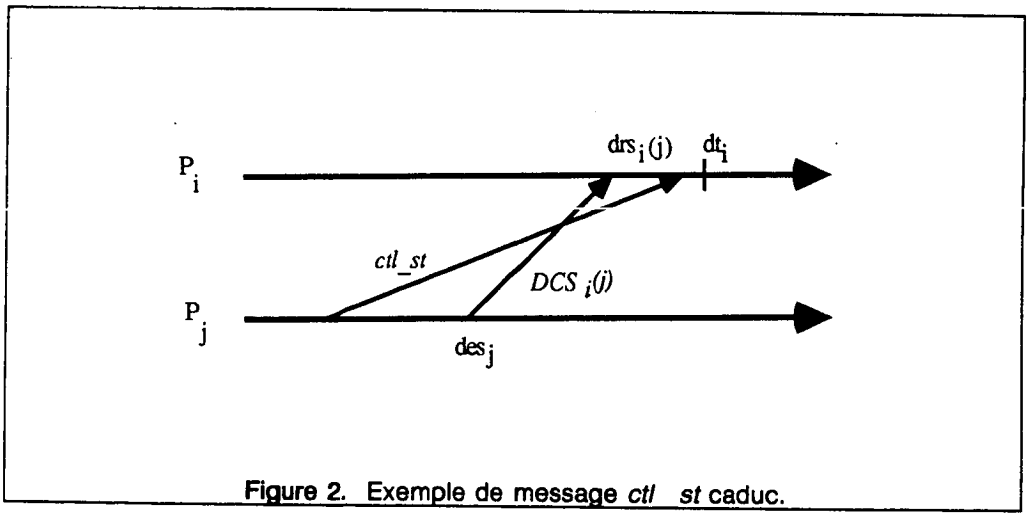

Figure 2. Exemple de message *ctl st* caduc.

Lemme 3.4.2 – 2 : lemme de « compensation des messages »

$$\left[\exists (P_j, P_k) \in VS_i \times VS_i \quad \exists m \in M(j,k) \quad dec_j(m,k) < des_j \wedge drc_k(m,j) > des_k\right]$$

$$\Rightarrow \left[\exists m' \in M(j,k)\ \ dec_j(m',k) > des_j \wedge drc_k(m',j) < des_k\right]$$

Ce lemme indique que (sous l'hypothèse que $EC(D)_i$ trouve $détecte_i$ à vrai à la date dt_i) on a :

si la situation décrite par la figure 3 se produit alors nécessairement le message m' a été émis, cette émission a eu lieu après la date des_j et sa réception a eu lieu avant des_k.

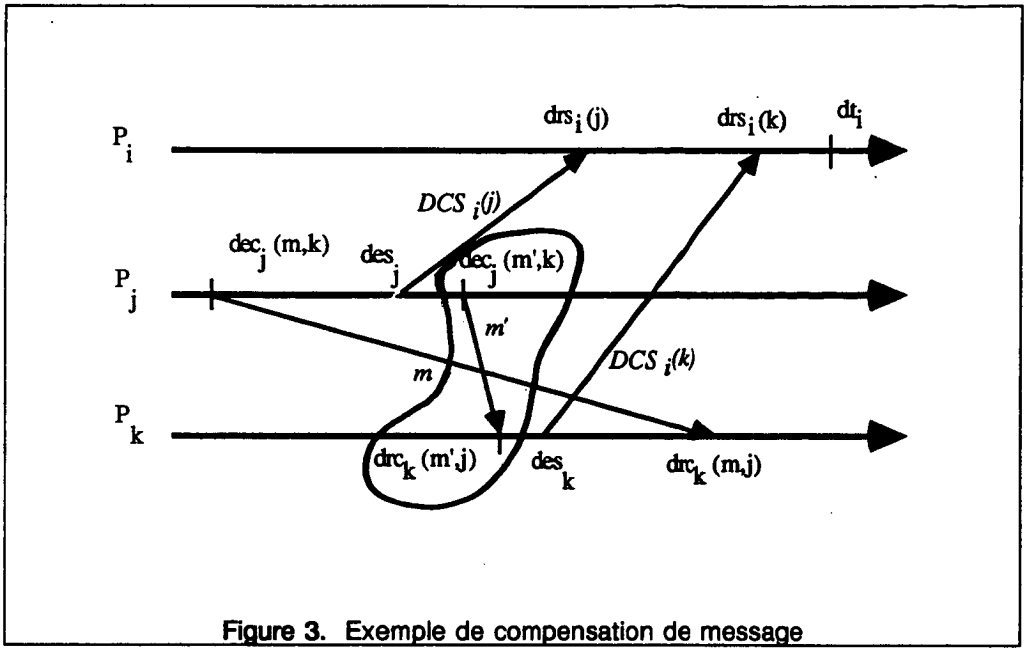

Figure 3. Exemple de compensation de message

Démonstration

D'après le lemme 3.4.2 – 1, l'égalité à la date dt_i :

$$obs\_émis_i[j][k](dt_i) = obs\_reçus_i[k][j](dt_i)$$

permet de conclure que :

$$instable\_émis_j[k](des_j) = instable\_reçus_k[j](des_k)$$

Le lemme en découle immédiatement : m a été « compensé » par un message m'. ■

Lemme 3.4.2 – 3 : lemme sur les « activations en chaîne »

S'il existe $P_j \in S$ recevant au moins un message déstabilisant après des_j désignons par m_1 le premier de ceux-ci au sens des dates de réception par P_j. Alors l'émetteur P_k de ce message vérifie :

1. $k \in S$

2. P_k a reçu au moins un message déstabilisant après des_k (soit m_2 le premier de ces messages reçus par P_k).
3. En appelant P_l l'émetteur du message m_2, on a :

$$drc_k(m_2,l) < drc_j(m_1,k)$$

Démonstration

Montrons tout d'abord le point 1. D'après la règle *R1*, $\forall t \in [des_j, drc_j(m_1,k)]$ on a $sp_j(t)$. Comme d'après la règle *R3* : $canaux\_\_fermés_j[x](des_j) \Rightarrow canaux\_\_fermés_j[x](drc_j(m_1,k))$
de : $\neg canaux\_\_fermés_j[k](drc_j(m_1,k))$
on déduit : $\neg canaux\_\_fermés_j[k](des_j)$
Or d'après le lemme 3.4.2 – 1 : $canaux\_\_fermés_j(des_j) = obs\_\_fermés_i[j](dt_i)$
d'où on conclut : $\neg obs\_\_fermés_i[j][k](dt_i)$
Par ailleurs, par définition de $détecte_i$, on a :

$$\Box\left[(détecte_i \wedge j \in S \wedge j \in VS_i) \Leftrightarrow \left[\neg obs\_\_fermés_i[j][k] \Rightarrow k \in S\right]\right]$$

Comme par hypothèse on a $détecte_i(dt_i)$ on conclut que $k \in S$.

Pour démontrer les points 2 et 3, deux cas sont à considérer selon la date d'émission de m_1 par P_k.

Premier cas : $dec_k(m_1,j) > des_k$ (figure 4).

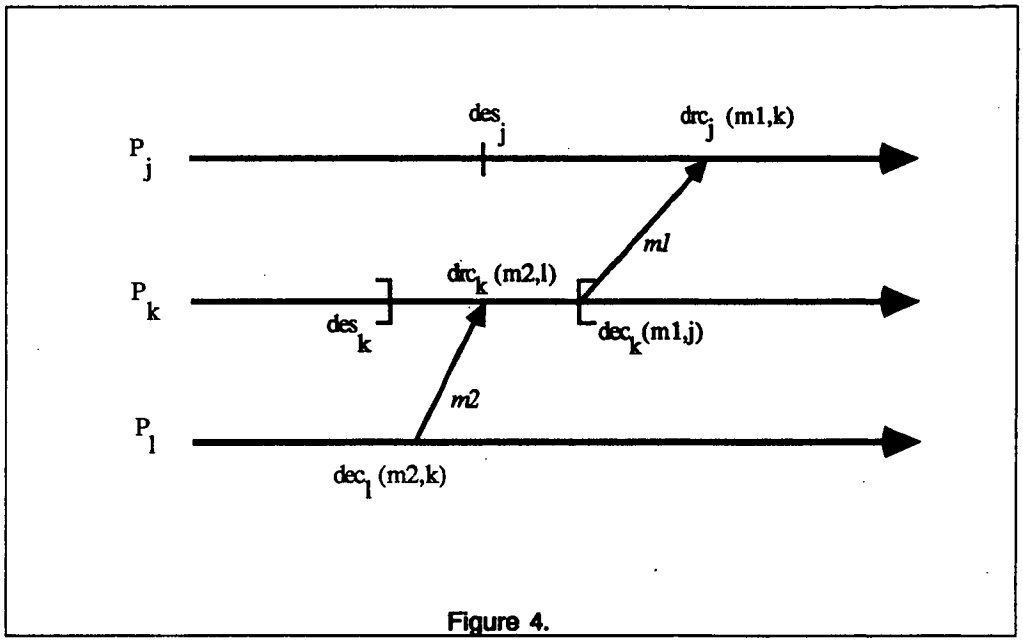

Figure 4.

On a : $sp_k(des_k)$, et $\neg sp_k(dec_k(m_1,j))$. D'après la règle de comportement *R1*, P_k a donc nécessairement reçu au moins un message déstabilisant dans l'intervalle $]des_k, dec_k(m_1,j)[$. Soit m_2 le premier de ceux-ci (au sens des dates de réception), émis par P_l. On a alors les inégalités suivantes :

$drc_k(m_2,l) < dec_k(m_1,j)$ (temps local de P_k)

$\wedge$ $dec_k(m_1,j) < drc_j(m_1,k)$ (entre émission et réception de m_1)

d'où $drc_k(m_2,l) < drc_j(m_1,k)$.

Deuxième cas : $dec_k(m_1,j) < des_k$ (figure 5).

D'après le lemme 3.4.2 – 2 (lemme de compensation), on a :

$$\exists m' \in M(k,j),\ dec_k(m',j) > des_k \wedge drc_j(m',k) < des_j$$

Comme dans le cas précédent, $sp_k(des_k)$ et $\neg sp_k(dec_k(m',j))$ impliquent la réception d'un message déstabilisant dans l'intervalle ; soit m_2 le premier de ceux-ci (au sens des dates de réception sur P_k), émis par P_l. On a alors les inégalités :

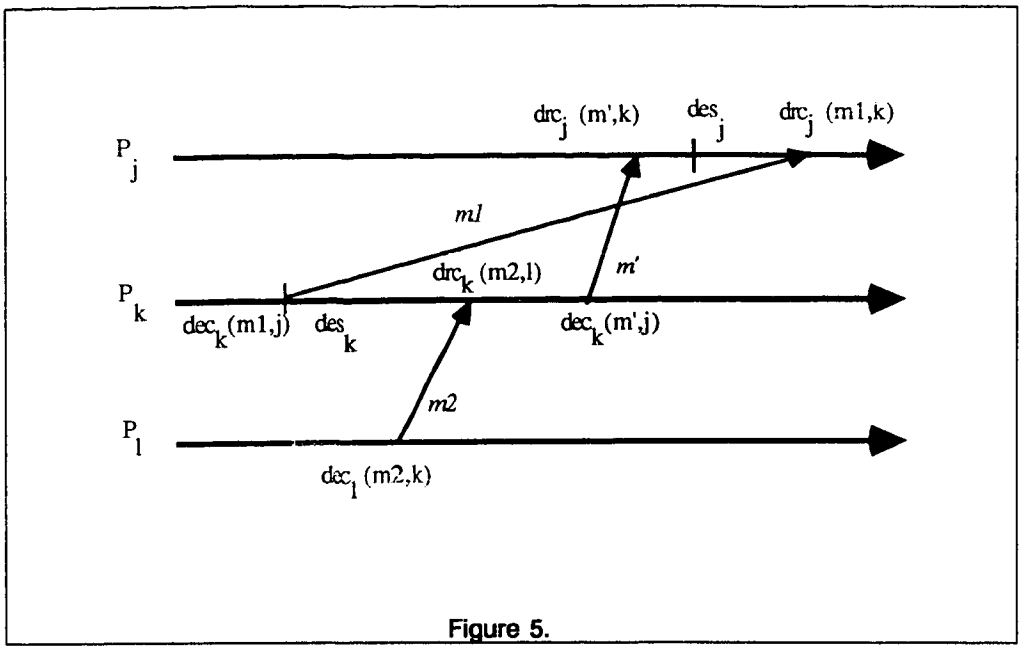

Figure 5.

$drc_k(m_2,l) < dec_k(m',j)$ (temps local de P_k)

$\wedge$ $dec_k(m',j) < drc_j(m',k)$ (entre l'émission et la réception de m')

$\wedge$ $drc_j(m',k) \leqq des_j < drc_j(m_1,k)$ (temps local de P_k)

d'où $drc_k(m_2,l) < drc_j(m_1,k)$. ■

Théorème 3.4.2 : *L'algorithme ne fait pas de fausse détection.*

Démonstration

Montrons d'abord la propriété suivante :

$$\left[détecte_i(S)(dt_i \wedge \neg sg(S)(dt_i)\right] \Rightarrow \left[\exists t \in [des_j, dt_i] \;/ \bigvee_{(j,k)\in S^2} \neg sc_{kj}(t)\right]$$

$sg(S)(dt_i)$ peut être faux pour trois raison :

1. $\exists j \in S,\ \exists k \in X - S : \neg fm_{kj}(dt_i)$
2. $\exists j \in S : \neg sp_j(dt_i)$

3. $\exists j \in S,\ \exists k \in S : \neg sc_{kj}(dt_i)$

Nous allons montrer que les propositions 1 et 2, sachant $détecte_i(S)(dt_i)$, impliquent

$$\exists t \in [des_j, dt_i],\ \exists (j,k) \in S^2 : \neg sc_{kj}(t)$$

ce qui prouvera la propriété.

Cas 1 :

$$\forall l \notin S,\ j \in S : détecte_i(S)(dt_i) \Rightarrow fm_{lj}(des_j)$$

puisque $obs\_\_fermés_i[j][l](dt_i)$.
Or nous avons $\neg fm_{kj}(dt_i)$; le processus P_j, ne pouvant ouvrir de canaux que s'il est actif (règle *R3*), a nécessairement reçu un message déstabilisant entre des_j et dt_i. Le processus émetteur de ce message est dans l'ensemble S, puisque d'après *R0*, P_j ne peut recevoir de messages déstabilisants sur un canal fermé, d'où $\exists t \in [des_j, dt_i] : \neg sc_{kj}(t)$.

Cas 2 :

Par la règle *R1* :

$$\left[\exists j \in S : \neg sp_j(dt_i)\right] \Rightarrow$$

$$\left[\exists j \in S,\ \exists k \in X,\ \exists t \in [des_j, dt_i] : \neg sc_{kj}(t)\right]$$

On en déduit par la règle *R0* :

$$\exists j \in S,\ \exists k \in X,\ \exists t \in [des_j, dt_i] : \neg fm_{kj}(t)$$

or nous avons $détecte_i(S)(dt_i)$ donc $k \in S$.

La propriété est donc démontrée. Elle indique que si $détecte_i$ et $\neg sg(S)$ sont vraies à la date dt_i alors il existe un processus P_j qui recevra nécessairement un message déstabilisant après des_j. On peut alors appliquer le lemme 3.4.2 – 3 sur les activations en chaîne pour construire une suite de processus incluse dans S : $P_j, P_{j_1}, \ldots, P_{j_\mu}$, et une suite de messages déstabilisants

$$m_1 \in M(j_1, j), m_2 \in M(j_2, j_1), \ldots, m_\mu \in M(j_\mu, j_{\mu-1}), \ldots$$

vérifiant

$$drc_j(m_1, j_1) > drc_{j_1}(m_2, j_2) > ... > drc_{j_{\mu-1}}(m_\mu, j_\mu) > ...$$

induisant une relation d'ordre strict sur cette suite de processus, d'où contradiction, le nombre des processus étant fini. ■

3.4.3 La détection est effectivement assurée

Il s'agit ici de montrer que si la stabilité est atteinte alors les entités de détection $EC(D)_i$ détecteront cet événement au bout d'un temps fini après son occurrence, en d'autres termes que l'on a :

$$\forall i \in X,\ \Box(sg \Rightarrow \Diamond détecte_i)$$

Démonstration

Supposons que, à l'instant t, il existe $S \subset X$, $S \neq \phi$, tel que $sg(S)$ soit vérifié. D'après la stabilité de $sg(S)$, on a alors $(\forall j \in S, \Box sp_j)$, si bien que, compte tenu des règles de comportement *R2* et *R3*, P_j ne recevra ni n'émettra aucun message déstabilisant après l'instant t (la seule action que P_j puisse encore entreprendre après l'instant t est la fermeture de canaux d'entrée).

Soit $DCS(j)$ le dernier message *ctl_st* (au sens des dates d'émission) diffusé par $EC(M)_j$ avant t, reçu par un détecteur $EC(D)_i$ à la date $drc_i(j)$. Les informations transportées par $DCS(j)$ sont alors prises en compte par $EC(D)_i$ (ce message n'est pas caduc, par définition) et ne seront plus modifiées (tout message *ctl_st* en provenance de $EC(M)_j$ parvenant à $EC(D)_i$ après $drs_i(j)$ est nécessairement caduc). Posons

$$dt_i = \max_{j \in S} \left[drc_i(j) \right]$$

A cet instant, le contexte de $EC(D)_i$ vérifie donc, d'une part :

$$\forall j \in S :\ participants_i[j](dt_i) \qquad (1)$$

d'autre part :

$$\forall j \in S,\ obs\_fermés_i[j](dt_i) = canaux\_fermés_j(des_j)$$

$$\text{et } obs\_fermés_i[j](dt_i) = canaux\_fermés_j(t)$$

et donc, d'après $sg(S)$:

$$k \notin S \Rightarrow obs\_fermés_i[j][k](dt_i) \qquad (2)$$

enfin :

$\forall j \in S, \forall k \in S$:

$$obs\_émis_i[j][k](dt_i) = instable\_émis_j[k](des_j)$$

$$obs\_reçus_i[k][j](dt_i) = instable\_reçus_k[j](des_k)$$

et, puisque tous les messages émis par P_j vers P_k avant des_j ont été reçus par P_k avant des_k, on a :

$$obs\_émis_i[j][k](dt_i) = obs\_reçus_i[k][j](dt_i) \qquad (3)$$

Les termes (1), (2) et (3) montrent que, à l'instant dt_i, $détecte_i$ est vérifié. ■

3.5 Cas des réceptions non spécifiées

Certaines applications ne respectent pas l'hypothèse *R0* qui, rappelons-le, interdit les réceptions non spécifiées [West 78]. Un processus P_i peut alors avoir fermé un de ses canaux d'entrée C_{ji} sans que pour autant P_j ne puisse lui envoyer de messages déstabilisants via ce canal ; ce canal étant fermé, ces messages ne peuvent en aucun cas déstabiliser P_i tant que sp_i a la valeur *vrai* (puisqu'un processus passif ne peut ouvrir un canal, d'après la règle *R3*). Le processus P_i ne pourra les consommer, s'il le désire, qu'une fois sp_i passé à *faux*. En termes opérationnels, un canal fermé n'est plus alors nécessairement vide ; nous appelerons cette hypothèse : *R0'*. Pour tenir compte de *R0'* la forme des propriétés stables doit être généralisée ; elle s'écrit alors :

$$sg(S) \equiv \bigwedge_{i \in S} \left[sp_i \wedge \bigwedge_{j \in S} (sc_{ji} \vee fm_{ji}) \wedge \bigwedge_{j \in X - S} fm_{ji} \right]$$

La détection de la stabilité doit être modifiée en conséquence. Le contrôle réalisé par l'algorithme est le même à la définition près du prédicat $détecte_i$ qui est alors :

$$détecte_i \equiv \exists S \,/\, S \neq \phi$$

$$\wedge\ \forall j \in S : \left[participants_i[j] \wedge \left\{ k \in X \,/\, \neg obs\_fermés_i[j][k] \right\} \subset S \right]$$

$$\wedge\ \forall (j,k) \in S^2 : \left[obs\_reçus_i[k][j] \neq obs\_émis_i[j][k] \Rightarrow obs\_fermés_i[k][j] \right]$$

4 Conclusion

L'algorithme de détection de propriétés stables présenté est intéressant à plus d'un titre. Le principe qui a présidé à sa conception est une approche essentiellement observationnelle : une entité de contrôle ne demande jamais d'informations, par contre elle observe le comportement du processus qu'elle est chargée de contrôler et, lorsque ce dernier manifeste des changements d'états, elle en fait systématiquement part aux autres entités de contrôle. La détection d'un propriété globale revient alors à tester un prédicat exprimé localement en termes de l'état global, tel qu'il est perçu par l'entité de contrôle. Une telle approche observationnelle ne se limite pas aux problèmes de détection de propriétés : elle est générale. Ce genre d'approche est adopté lors de la distribution de machines d'états telles qu'en proposent [Lamport 78] et [Lamport et Schneider 85].

Dans une telle approche, les échanges de messages qu'exige la coopération entre les contrôleurs sont provoqués par les changements d'états des processus contrôlés et non par les contrôleurs eux-mêmes. Il n'est pas nécessaire d'introduire une notion de période d'observation [Chandy et Misra 85] ou d'interrogation [Chang 83, Francez 80] durant laquelle s'effectue une collecte cohérente d'informations, l'observation est ici non impérative (on peut y voir la même différence qu'entre l'attente passive et l'attente active pour l'accès en exclusion à un objet partagé).

Un intérêt de l'approche retenue concerne l'efficacité de l'algorithme général proposé. Le nombre de messages de contrôle (et donc de diffusions de tels messages) est au plus égal à p, p étant le nombre total de changements d'état des processus contrôlés. De plus, lorsque la stabilité est atteinte, le temps au bout duquel cet événement est détecté par toutes les entités de contrôle est égal à la durée maximale nécessitée par la diffusion d'un message de contrôle ; la complexité temporelle est $O(1)$ en nombre d'opérations de diffusion. Le protocole de diffusion requis par l'algorithme est très simple : il autorise le déséquencement et la duplication de messages, il n'a pas besoin d'être pourvu de propriétés complexes telle que l'atomicité, par exemple. Le nombre de messages nécessaires pour réaliser de telles diffusions est fonction du nombre d'entités de détection et du maillage de communication sous-jacent ; si des arborescences de diffusion sont préalablement construites, une telle diffusion a une complexité temporelle en $O(d)$ où d est le diamètre du graphe des communications ($1 \leqq d < n$) [Segall 83].

Si l'on s'intéresse à un problème particulier, à la terminaison par exemple, le nombre de changements d'états p est au pire le nombre de messages échangés par le calcul sous-jacent. Dans ce cas particulier, rappelons à titre de comparaison les complexités d'autres algorithmes de terminaison. Les algorithmes présentés dans [Misra 83] et [Lai 86] requièrent (sous des hypothèses analogues) respectivement $p.e$ et $p.n$ messages de contrôle (e étant le nombre de canaux de communication), et présentent une complexité temporelle respectivement en $O(e)$ et $O(n)$. (De plus, l'algorithme présenté dans [Misra 83] ne tolère pas le déséquencement des messages.)

Parmi les algorithmes de détection de propriétés stables, il existe un algorithme particulier, qui est relatif à la terminaison, fondé sur une approche observationnelle [Dijkstra 80]. Mais dans cet algorithme, seul un processus privilégié va détecter la terminaison. L'algorithme proposé ici généralise en quelque sorte ce dernier algorithme, selon deux aspects. Tout d'abord, le nombre de détecteurs peut varier entre 1 et n, au gré du concepteur ; celui-ci peut donc en quelque sorte définir le « degré de symétrie » de son algorithme de contrôle. De plus, l'algorithme proposé est indépendant de la propriété particulière qu'est la terminaison : il détecte les propriétés stables de la forme indiquée au paragraphe 2.2.1.

Notre algorithme général de détection a pu être obtenu grâce à l'emploi d'une conception méthodique, séparant clairement l'aspect calcul de l'aspect contrôle, conformément à l'approche proposée dans [Chandy et Misra 86]. Le travail original présenté suggère une ouverture intéressante : la composition d'algorithmes de détection de propriétés (éventuellement non stables) lorsque l'intersection de ces propriétés est stable (par exemple deux algorithmes dont chacun détecte la terminaison d'une moitié de l'application). Trouver les règles d'une telle composition constituerait un outil puissant de conception d'algorithmes répartis et pourrait aider à une meilleure appréhension du concept d'état global dans un système réparti [Chandy et Lamport 85].

Références

[Apt et Richier 85]

APT, K.R. et RICHIER, J.L., *Real Time Clock versus Virtual Clocks,* NATO ASI Series Vol. 14, Concepts for Distributed Programming, (Springer Verlag), (1985), pp. 475 – 501.

[Chandy et Misra 82]

CHANDY, K.M. et MISRA, J., *A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems,* ACM SIGACT-SIGOPS, Symp. Principles of Distributed Computing, (August 1982), pp. 157 – 164.

[Chandy et al. 83]

CHANDY, K.M., MISRA, J. et HAAS, L., *Distributed Deadlock Detection,* ACM TOCS, Vol. 1,2, (May 1983), pp. 144 – 156.

[Chandy et Lamport 85]

CHANDY, K.M. et LAMPORT, L., *Distributed Snapshots: Determining Global States of Distributed Systems,* ACM TOCS, Vol. 3,1, (February 1985), pp. 63 – 75.

[Chandy et Misra 85]

CHANDY, K.M. et MISRA, J., *A Paradigm for Detecting Quiescent Properties in Distributed Computations,* NATO ASI Series, Vol. 13, Logics and Models of Concurrent Systems, (Apt Ed.), Springer Verlag, (1985), pp. 325 – 341.

[Chandy et Misra 86]

CHANDY, K.M. et MISRA, J., *An Example of Stepwise Refinement of Distributed Programs: Quiescence Detection,* ACM TOPLAS, Vol. 8,3, (July 1986), pp. 326 – 343.

[Chang 82]

CHANG, E., *Echo Algorithms: Depth Parallel Operations on General Graphs,* IEEE Trans. on SE, Vol. SE8,4, (July 1982), pp. 391 – 401.

[Dijkstra 74]

DIJKSTRA, E.W.D., *Self-Stabilizing Systems in Spite of Distributed Control,* Comm. ACM, Vol. 17,11, (November 1974), pp. 643 – 644.

[Dijkstra et Scholten 80]

DIJKSTRA, E.W.D. et SCHOLTEN, C.S., *Termination Detection for Diffusing Computation,* Inf. Proc. Letters, Vol. 11,1, (August 1980), pp. 1 – 4.

[Dijkstra et al. 83]

DIJKSTRA, E.W.D, FEIJEN, W.H.D. et VAN GASTEREN, A.J.M., *Derivation of a Termination Detection Algorithm for Distributed Computations,* Inf. Proc. Letters, Vol. 16, (1983), pp. 217 – 219.

[Francez 80]

FRANCEZ, N., *Distributed Termination,* ACM TOPLAS, Vol. 2,1, (January 1980), pp. 42 – 55.

[Hoare 78]

HOARE, C.A.R., *Communicating Sequential Processes,* Comm. of ACM, Vol. 21,8, (August 1978), pp. 666 – 677.

[Lai 86]

LAI, T.H., *A Termination Detector for Static and Dynamic Distributed Systems with Asynchronous non FIFO Communication,* Proc. 13th ICALP Conf., LNCS #226, (Springer Verlag), (1986), pp. 196 – 205.

[Lamport 78]

LAMPORT, L., *Time, Clocks and the Ordering of Events in a Distributed System,* Comm. of ACM, Vol. 21,7, (July 1978), pp. 558 – 565.

[Lamport 80]

LAMPORT, L., *"Sometimes" is sometimes "not never": on the Temporal Logic of Programs,* Proc. 7th ACM Symposium on POPL, (January 1980), pp. 174 – 185.

[Lamport et Schneider 85]

LAMPORT, L. et SCHNEIDER, F.R., *Paradigms for Distributed Programs in Distributed Systems,* LNCS #190, (Springer Verlag), (1985), pp. 431 – 480.

[Misra 83]

MISRA, J., *Detecting Termination of Distributed Computations using Markers,* Proc. ACM Symposium on PODC, Montréal, (August 1983), pp. 290 – 294.

[Natarajan 86]

NATARAJAN, N., *A Distributed Scheme for Detecting Communication Deadlocks,* IEEE Trans. on Soft. Eng., Vol. SE-12,4, (April 1986), pp. 531 – 537.

[Pnueli 86]

PNUELI, A., *Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends,* LNCS #224, Current Trends in Concurrency, (Springer Verlag), (1986), pp. 510 – 584.

[Segall 83]

SEGALL, *Distributed Networks Protocols,* IEEE Trans. on Inf. Theory, Vol. IT29,1, (January 1983), pp. 23 – 35.

[Shavit et Francez 86]

SHAVIT, N. et FRANCEZ, N., *A New Approach to Selection of Locally Indicative Stability,* 13th ICALP, LNCS #226, (Springer-Verlag), (1986), pp. 344 – 358.

[Tarjan 72]

TARJAN, R., *Depth-first Search and Linear Graph Algorithms,* SIAM Journal of Computing, Vol. 1, (1972), pp. 146 – 160.

[West 78]

WEST, C.H., *An Automated Technique of Communication Protocol Validation,* IEEE Trans. on Comm., Vol. COM-26, (1978), pp. 1271 – 1275.

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