De la nécessite de specifier des propriétes pour la vérification des algorithmes distribués

Claude Jard, Michel Raynal

IRIA

CENTRE DE RENNES

IRISA

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° 590

DE LA NÉCESSITÉ DE SPÉCIFIER DES PROPRIÉTÉS POUR LA VERIFICATION DES ALGORITHMES DISTRIBUÉS

Claude JARD
Michel RAYNAL

Décembre 1986

De la nécessité de spécifier des propriétés pour la vérification des algorithmes distribués

Specification of properties is required to verify distributed algorithms

Claude JARD, Michel RAYNAL

IRISA, Campus de Beaulieu,
35042 RENNES Cedex
Tél. 99 36 20 00

Résumé: cet article argumente la thèse suivante: la vérification d'un algorithme distribué s'effectue par l'analyse d'un modèle de celui-ci. Raisonner uniquement à l'intérieur du modèle et s'assurer de sa cohérence ne suffit pas en général. On a besoin d'un méta-langage de spécification pour décrire les propriétés attendues du modèle, qui dépendent de l'algorithme.

Ce fait, bien qu'admis par plusieurs spécialistes du domaine de la vérification, n'est pas reconnu par tous. Notre but est de le présenter de façon pédagogique à l'aide d'un exemple simple. L'article fournit, en même temps, une introduction à la vérification des algorithmes distribués.

Abstract: this paper argue the following thesis: distributed algorithms verification is performed by analysing a formal model of its. Reasoning solely within the model and verifying its consistency is not enough in general. We need a meta specification language in order to describe expected properties of the model, which depend on the algorithm under study. This fact, although acknowledged by several specialists in verification, does not get yet unanimity. We aim at presenting this thesis with a teaching view using a small example. The paper provides at the same time, an introduction to verification techniques for distributed algorithms.

I. Introduction

Vérifier un algorithme, c'est s'assurer qu'il fonctionne (ou fonctionnera) correctement sous une hypothèse d'environnement donnée.

Par algorithme distribué on entend tout algorithme conçu en termes d'entités (ou processus) coopérant par échange de messages. La répartition des entités peut être logique ou physique. Les protocoles de communication sont des exemples de tels algorithmes dans le contexte de la répartition géographique rencontrée dans les réseaux.

Concevoir un algorithme distribué est un exercice malaisé: se convainvre de son bon fonctionnement est, dans l'état actuel de l'art, difficile. L'exemple simple traité tout au long de cet article va illustrer quelques unes des difficultés auxquelles se trouve confronté le concepteur-validateur de tels algorithmes.

C'est ce constat de la difficulté de maîtrise qui justifie les nombreux travaux d'équipes sur la vérification des algorithmes distribués. Nous notons d'ailleurs que de plus en plus de protocoles et d'algorithmes distribués sont conçus et mis en oeuvre sur des réseaux de calculateurs et nombreux sont ceux qui présentent des comportements inattendus, voire erronés.

Cet article présente, à travers un exemple très simple, la démarche générale de vérification. Partant de l'algorithme à étudier, on déduit un modèle formel. C'est ce modèle qui est ensuite analysé. Nous utilisons ici le formalisme facilement accessible des automates communicants [Rubin 82, Brand 83].

La plupart des outils de vérification se contentent d'une "analyse interne" du modèle en examinant les propriétés générales de sa structure (vivacité) [Bochmann 78, Diaz 84, Berthelot 82, Gouda 84]. Nous voulons ici montrer que celà n'est pas suffisant.

Quelques chercheurs ont déjà montré la voie [Queille 82, Clarke 84]: une fois le modèle relatif à l'algorithme établi, il faut exprimer à l'aide d'un méta-langage, c'est à dire un véritable *langage de spécification*, les propriétés que doit présenter le modèle. Nous donnerons dans cet article, un aperçu de l'utilisation d'une logique temporelle.

Le plan est le suivant:

- on part d'un exemple de deux processus en communication,
- l'analyse générale conduit à isoler un comportement bloquant,
- on tente alors une correction guidée par l'analyse précédente,
- l'étude du modèle obtenu ne met pas en évidence d'anomalies, alors que l'algorithme ne satisfait pas à la spécification intuitive du problème,
- on montre alors comment formaliser, à l'extérieur du modèle, cette spécification, puis on propose une correction du modèle, donc de l'algorithme, satisfaisant la spécification énoncée.

II. Un exemple: deux processus en communication

II.1. Le modèle*

Deux processus, A et B, veulent entrer en communication. Pour celà, ils doivent ouvrir une connexion logique entre eux. Supposons pour simplifier que seul A peut prendre en prendre l'initiative, en émettant le message (a). Il peut alors, après un temps indéterminé, décider de fermer la connexion par l'émission d'un message de déconnexion (b). De même, après avoir reçu le message d'ouverture (a), et aprés un temps indéterminé pendant lequel la connexion est ouverte, le processus B peut demander sa fermeture en émettant le message (c).
Initialement, la connexion est fermée et aucun message n'est en transit.

Nous supposons que les interaction entre A et B s'effectuent par l'intermédiaire

de canaux de communication fifo, non bornés, fiables et initialement vides.

La figure 1 présente une *modélisation* du protocole d'échange. Le formalisme utilisé est volontairement trés simple. Les processus sont représentés par des automates d'états finis, dont les transitions sont étiquetées par des émissions (notées +) ou des réceptions de messages (notées -). Un tel automate est doté d'un état initial et d'un ou de plusieurs états d'acceptation; ce sont dans ces états que les processus réalisant l'algorithme distribué parviennent après que le service qu'ils mettent en oeuvre soit rendu.

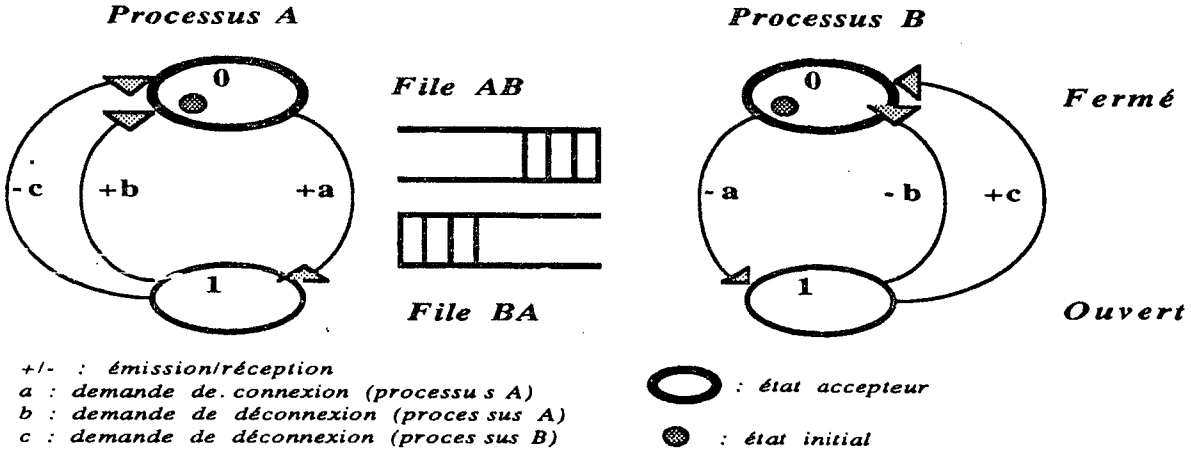


Figure 1: schéma d'un protocole de connexion-déconnexion

L'état global d'un tel système est formé par l'état de chaque processus (les états sont notés 0 ou 1) et par l'état des canaux (séquence de messages en transit). On le représentera par un quadruplet $<état_A, file_{AB}, état_B, file_{BA}>$.

Sémantique d'exécution: une transition étiquetée (+x) est exécutable par le processus Y si et seulement si l'automate qui lui est associé se trouve dans un état d'où la transition est sortante. L'exécution de la transition fait passer l'automate dans l'état de sortie de cette transition et dépose le message (x) dans la file d'émission du processus Y. Une transition étiquetée (-x) est exécutable si et seulement si l'automate se trouve dans un état d'où cette transition est sortante et si le message (x) est présent en tête de la file de réception du processus (on notera la disymétrie de la communication). L'exécution de la transition fait passer l'automate dans l'état de sortie de la transition et consomme le message (x): celui-ci est retiré de la file; le message suivant, s'il existe, devient alors tête de file.

II.2. L'analyse

La première idée (et aussi la plus simple) pour examiner les propriétés d'un protocole, est de construire l'ensemble de ses comportements, c'est à dire l'ensemble des séquences d'exécution qu'il est apte à produire. Ces comportements peuvent être obtenus à partir du modèle du protocole par l'algorithme de construction suivant.

Partant de l'état global initial, on détermine quelles sont les actions possibles. On effectue alors chaque action, en notant les états globaux obtenus si ceux-ci n'ont pas déjà été explorés. L'opération est recommencée pour chaque nouvel état jusqu'à ce que tous les états globaux aient été explorés.

L'exécution de cet algorithme est appelée l'analyse d'accessibilité.

Le graphe formé des états globaux, reliés par des arcs étiquetés par les actions des processus est appelé *graphe des états globaux*. Lorsque celui-ci est fini, il représente l'automate d'état fini du système global, modélisant le système formé de plusieurs automates d'état finis communicants.

Une question fondamentale à propos de cette technique est la propriété de terminaison de l'algorithme de construction. L'application de cet algorithme à l'exemple produit le graphe dont l'allure est donnée par la figure 2. Il semble clair que dans ce cas précis, la construction ne se termine pas: en effet, le nombre d'états de ce graphe est infini!

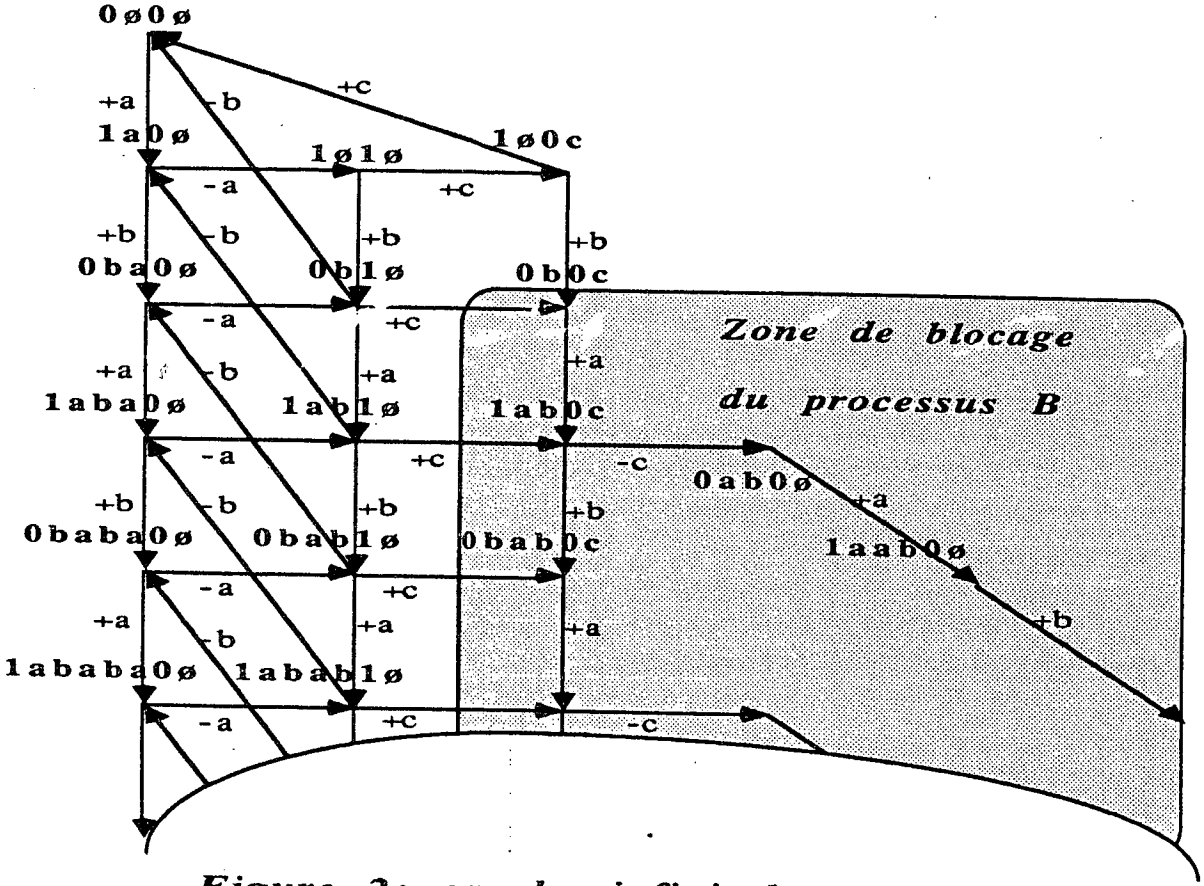


Figure 2: graphe infini des états

On peut noter sur ce graphe:

- des comportements infinis *non équitables*: si on suit la colonne de flèches à gauche de la figure par exemple, on s'aperçoit que le processus A évolue indéfiniment tout seul, alors que le processus B pourrait recevoir un message (-a).
- le blocage du processus B qui, dans l'état 0, ne peut que consommer un message de type (a) alors qu'en tête de la file fifo AB se trouve un message du type (b). Un tel comportement est appelé *réception non spécifiée*.

II.3. Le problème de l'équité

Les séquences inéquitables ont pour cause le *non-déterminisme* qui peut provenir de diverses origines.

A l'intérieur d'un même processus X, si plusieurs actions sont toujours possibles,

la possibilité que dans un comportement du système, ce soit toujours la même qui soit choisie, conduit à des comportements non équitables. Le non-déterminisme qui conduit à de telles situations est inhérent à la définition du processus X.

La séquence non équitable mise en évidence dans l'exemple a une autre origine. Elle est dûe à la modélisation du parallélisme entre processus par une technique d'entrelacement de leurs actions. Cette technique, qui consiste à n'observer qu'une action à la fois dans le système modélisé, peut produire de telles séquences inéquitables si, à tout instant, les deux processus du système peuvent produire des actions. Le choix non-déterministe effectué par la technique d'entrelacement définit alors des comportements équitables et des comportements non équitables: dans la séquence inéquitable montrée dans l'exemple, le processus B peut consommer (a), le processus A peut produire, et c'est toujours la production par A qui est choisie. Dans la réalité, si les processus s'exécutent en simultanéité et respectent l'hypothèse de progression (leur vitesse est strictement positive), les actions de A et B finiront par avoir lieu: les séquences inéquitables précédentes deviennent alors finies (et donc équitables).

II.4. Le problème de la finitude du graphe

La construction systématique du graphe des états pose un problème majeur pour l'étude du comportement des algorithmes distribués. En effet, pour la classe de modèles d'automates communiquant par des files fifo non bornées, le caractère fini du graphe d'état n'est pas décidable (la puissance du modèle est celle de la machine de Türing [Rubin 82]).

Il est courant dans la pratique de se limiter à des files de communication finies: le modèle qui en résulte est alors un graphe d'états fini. La sémantique de l'émission doit être alors modifiée: celle-ci n'est possible que si la file correspondante n'est pas pleine. Si les processus A et B de l'exemple communiquent via des files limitées à deux éléments, on obtient le graphe d'états présenté à la figure 3. On constate sur celui-ci que la propriété de blocage du processus B est maintenue: la réception non spécifiée relative à B y est transformée en un interblocage certain.

Comme nous l'avons indiqué, le caractère indécidable de la finitude du graphe des états, est un problème majeur, et ce, du point de vue pratique. En effet, si tel n'était pas le cas, l'étude des comportements d'un algorithme distribué ou d'un protocole pourrait être initialement faite en supposant que les processus (modélisés par des automates) communiquent via des files infinies. Si la procédure de décision indiquait alors que le graphe est fini, sa construction permettrait d'en déduire des résultats intéressants comme, par exemple, les capacités maximales de chacune des files. Malheureusement il n'en est rien. Le concepteur doit donc borner lui-même la capacité des files afin d'obtenir un graphe d'états fini. Il s'agit là d'un problème difficile: comment être sûr que le choix de telle capacité de file n'élimine pas des comportements intéressants de l'algorithme?

Remarque: dans le cas où l'une des deux files de communication est a priori bornée, il existe un résultat théorique très intéressant: la finitude du graphe d'états est décidable [Gouda 85, Rubin 82]. Signalons également que sous la même hypothèse, est également décidable le fait de savoir si le système est exempt d'interblocage et de réception non spécifiée. De plus, rappelons d'une part que la finitude est également décidable lorsque la modélisation est faite par un réseau de Petri [Karp 69] et d'autre part, que les réseaux de Petri ne permettent pas de modéliser une file fifo non bornée lorsque celle-ci peut contenir des messages de plusieurs types.

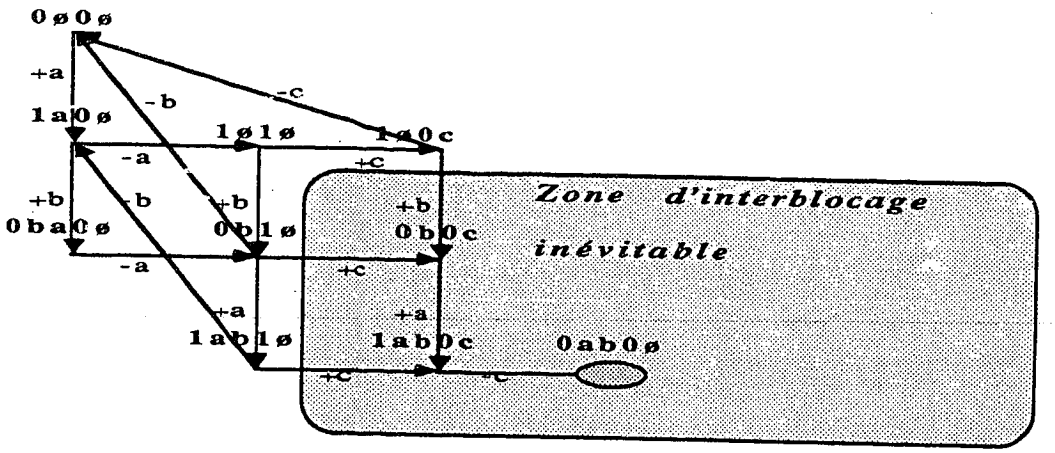
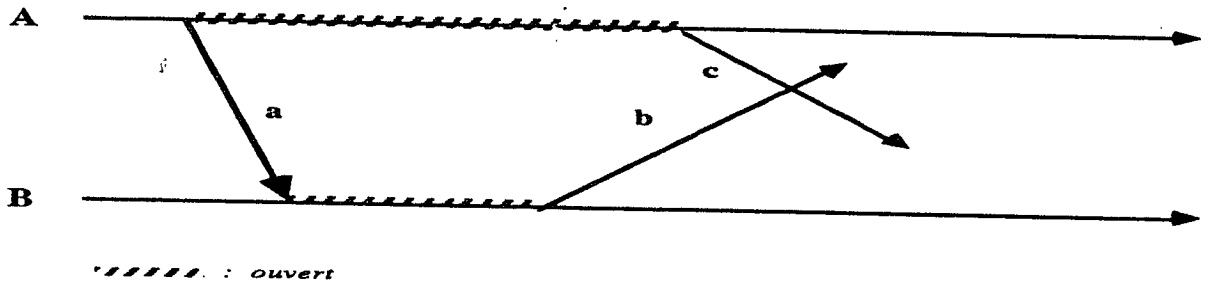


Figure 3: graphe d'état avec communication bornée (à 2)

III. Tentative de correction

Partant du modèle obtenu, nous allons effecteur une première correction visant à supprimer le comportement bloquant de l'algorithme. L'état de blocage est inévitable à partir de l'état <0b0c>. Il est obtenu après la séquence de transitions illustrée par la figure 4.



'////// : ouvert

Figure 4: séquence de blocage

Le problème est dû à la collision des deux fermetures (b) et (c) de la connexion ouverte par A avec (a). Dans une telle circonstance, le processus A (respectivement B) peut voir venir une déconnexion (c) (respectivement (b)) alors qu'il est dans l'état fermé (0). Une première idée est donc de consommer ces messages dans l'état (0), la fermeture étant demandée par les deux partenaires. La figure 5 présente cette tentative de correction sur les automates initiaux.

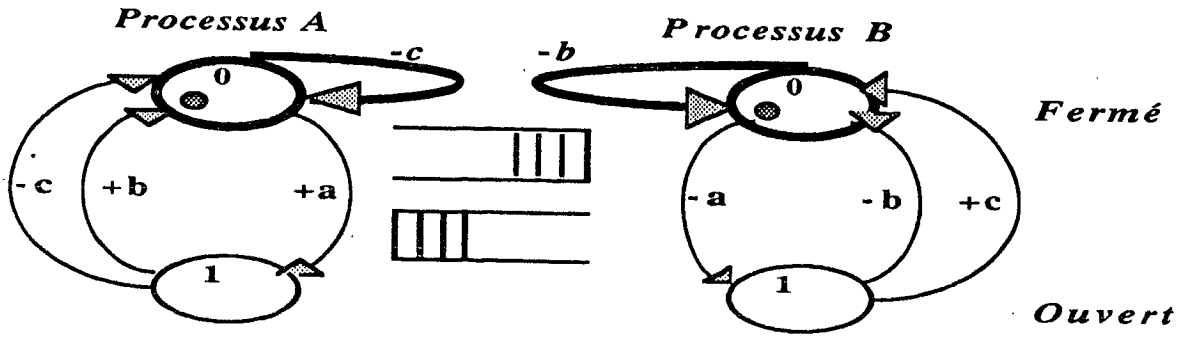


Figure 5: tentative de correction du protocole

La figure 6 donne le graphe d'état obtenu en bornant la communication à 1: cette restriction permet de visualiser un graphe encore simple. Si on ne borne pas la communication, le graphe est bien sûr, comme précédemment, de taille infinie (par ailleurs, on peut constater que des bornes supérieures à 1, en ce qui concerne la capacité des files de communication ne modifient pas les résultats de l'analyse).

L'analyse du graphe d'état ne fait apparaître aucun blocage (état puits); de plus, le graphe est *vivant*, au sens où n'importe quel état est accessible depuis n'importe quel autre. Peut-on, pour autant, en conclure que le protocole est correct?

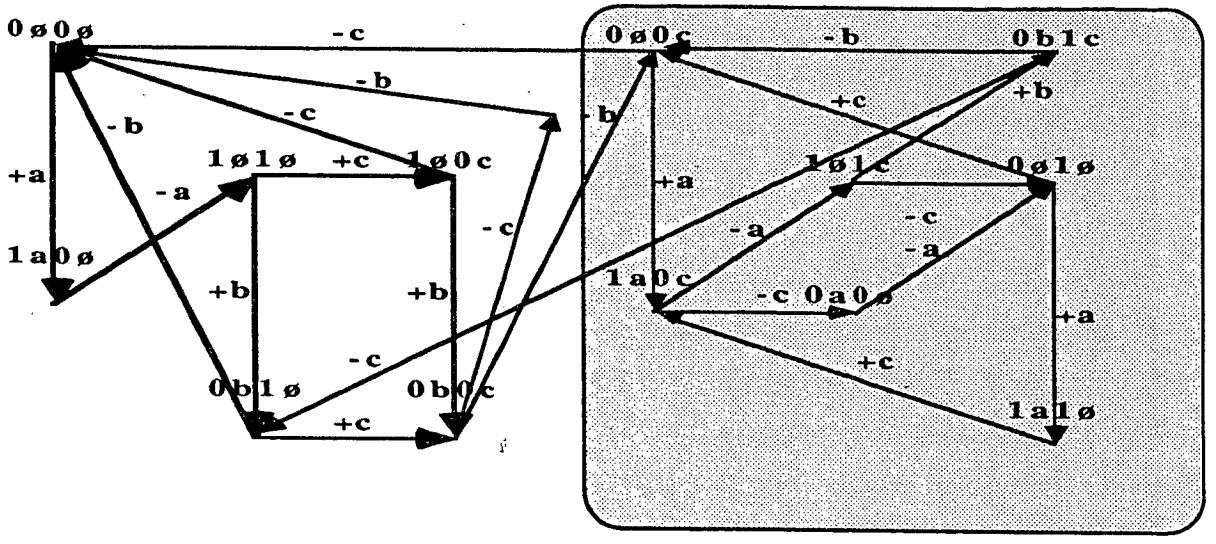


Figure 6: graphe de la tentative de correction, (avec communication bornée à 1)

IV. Spécifier le service

Pour répondre à la question précédente, il faut d'abord se demander ce que l'on attend exactement du protocole de connexion-déconnexion.

Une façon de voir est de considérer un observateur global, "omniscient" et notant les ouvertures et fermetures de connexions. On veut alors observer une suite d'ouvertures et de fermetures, chaque ouverture est suivie par une fermeture, demandée soit par A, soit par B, soit par les deux (ce cas est celui de la collision des fermetures).

La figure 7 montre l'automate d'état fini décrivant les comportements que cet observateur doir percevoir. Cet automate prend en compte explicitement les comportements de collisions (ce problème doit être réglé dés la conception) et s'écrit naturellement sous une forme non-déterministe. La figure 8 montre l'automate déterministe équivalent; ce dernier multiplie les états accepteurs.

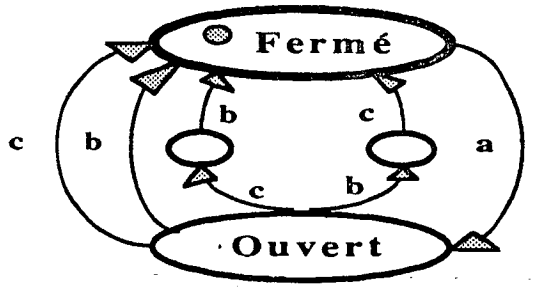


Figure 7: service abstrait global

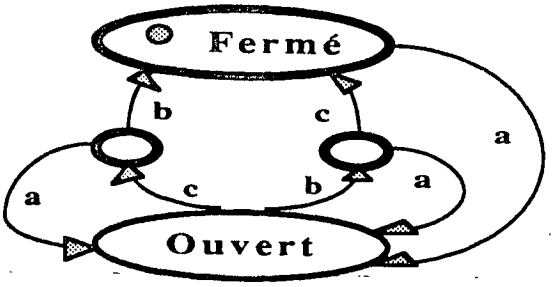


Figure 8: équivalent déterministe

IV.1. Vérifier par abstraction

Prenons le graphe global de la figure 6. Supposons maintenant que seuls les événements (+a), (+b) et (+c) nous intéressent, parce que correspondants à la vision abstraite du service: ce sont les décisions prises par les utilisateurs du protocole. Les autres événements ne vont pas être observés et donc seront considérés comme invisibles.

Ceci revient à remplacer les arcs du graphe étiquetés (-a), (-b) et (-c) par un symbole vide, noté λ, élément neutre pour la concaténation. La question est maintenant de vérifier l'équivalence (ici, au sens de l'égalité des langages reconnus) entre ce nouvel automate global avec transitions vides et l'automate abstrait du service de la figure 7.

La question de l'équivalence peut être aisément tranchée en éliminant les transitions vides, celles-ci ne figurant pas dans les mots du langage reconnu. C'est cette élimination que l'on appelle une abstraction. A chaque chemin de la forme $<+x\lambda^*>$ dans l'automate global correspond un arc (x) dans l'automate abstrait. On vérifie alors l'isomorphisme de l'automate obtenu avec celui du service.

La figure 9 montre le graphe abstrait obtenu. Celui-ci n'est pas équivalent au service de la figure 7.

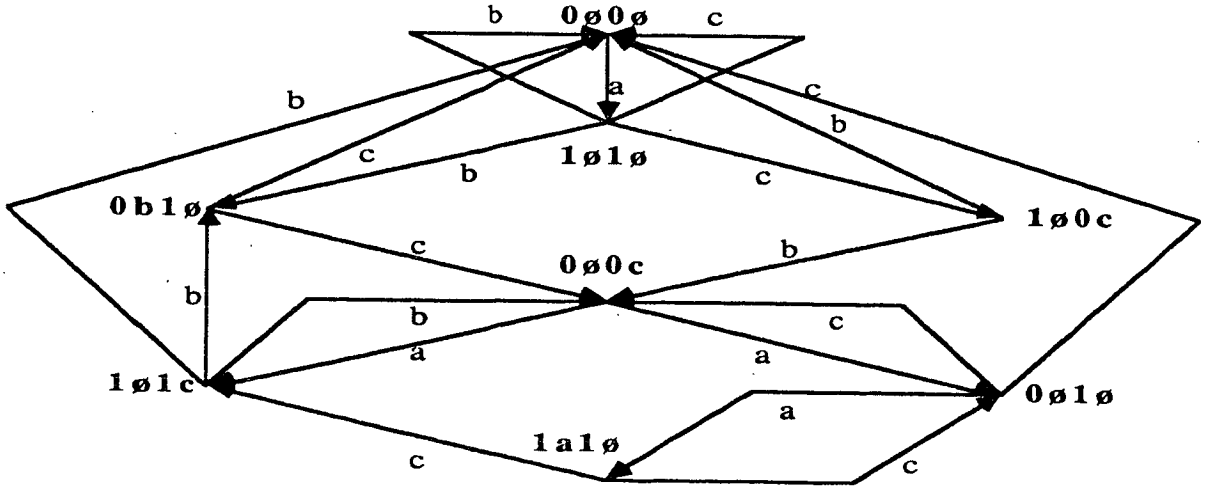


Figure 9: abstraction de la tentative de correction

Parmi tous les comportements qu'il définit, ce graphe d'états présente en effet le comportement particulier <a.b.c.a.a> illustré par le diagramme de la figure 10 (on y a indicé les messages par la connexion à laquelle ils se rapportent afin de mieux appréhender le comportement décrit). Ce comportement met en évidence une confusion dans l'interprétation du message de déconnexion (c). Celui-ci est pris pour la demande de fermeture de la deuxième connexion ouverte par le processus A, alors qu'il correspond en réalité pour B à la demande de fermeture de la première connexion; cette déconnexion, émise par le processus B, est entrée en collision avec la demande analogue émanant du site A. Dans le comportement observé la parenthésage attendu entre les ouvertures et les fermetures de connexions n'est pas réalisé: deux ouvertures s'y succèdent.

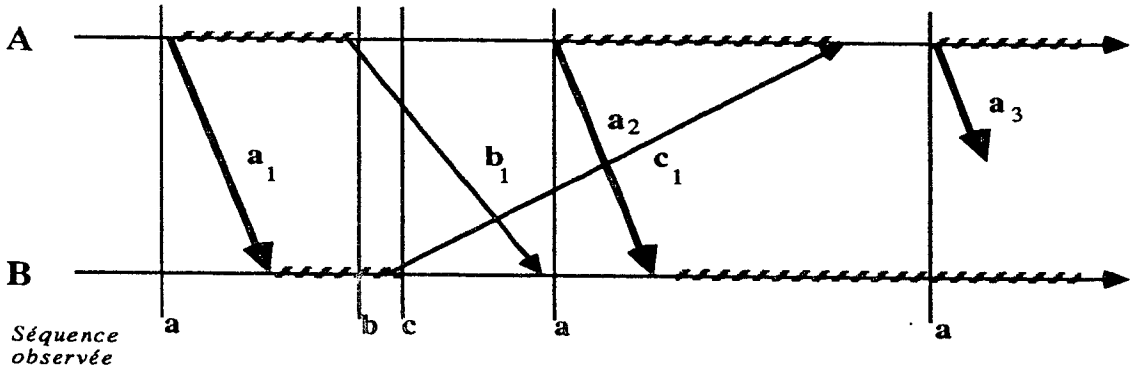


Figure 10: exécution violant la propriété de service (confusion entre deux connexions)

IV.2. Vérifier des propriétés

Une autre approche consiste à exprimer les propriétés que doit satisfaire la modélisation de l'algorithme distribué. Un type de formalisme souvent utilisé est la logique temporelle.

Le cadre logique offre une concision et une abstraction intéressante pour décrire les services des protocoles. Les logiques temporelles permettent de décrire l'ordonnancement dans le temps d'occurences d'événements.

Le modèle abstrait de la figure 7 peut s'exprimer sous forme de propriétés. Informellement:

i) Initialement, seule une demande d'ouverture (a) peut être effectuée.

ii) Il est toujours vrai qu'aprés une demande d'ouverture (a), une nouvelle demande (a) n'est possible que si entre temps, il y a eu une demande de fermeture (de la part de A (b), de la part de B (c), ou de la part des deux: collision (b) et (c)).

Ces propriétés peuvent être formalisées dans la logique CTL (*Conditional Time Logic*). Nous ne définissons pas ici cette logique (le lecteur en trouvera une présentation dans [Queille 83]); nous nous limitons ici à l'interprétation informelle des prédicats et opérateurs utilisés pour exprimer les propriétés i) et ii) sur une exécution, perçue sous la forme d'une suite d'états globaux du système modélisé.

Prédicats:

initial : vrai seulement dans l'état initial.
possible (+x) : vrai si l'action +x est possible dans l'état considéré.
après (+x) : vrai dans l'état atteint après avoir effectué l'action +x.

Opérateurs:

toujours (f) : vrai dans l'état e considéré si f est vrai dans tous les états accessibles depuis e.

inévitable (f1) *jusqu'à* (f2) : vrai dans l'état e considéré si pour chacun des chemins du graphe d'état issu de e $<e=e_0,e_1,...,e_i,...,e_k,...>$, f1 est vrai dans tous les états e_i jusqu'à un état e_{k-1} tel que f2 est vrai dans l'état e_k.

$\sim$, $\wedge$, $\vee$ et $\supset$ sont les opérateurs classiques du calcul des propositions (non, et, ou et implique).

On note *possible* (+x,+y) = *possible* (+x) $\vee$ *possible* (+y)
après (+x,+y) = *après* (+x) $\vee$ *après* (+y)

Les propriétés i) et ii) s'expriment alors par:

i) *initial* $\supset$ (*possible* (+a) $\wedge \sim$ *possible* (+b,+c))

ii) *toujours* (*après* (+a) $\supset$ *inévitable* ($\sim$ *possible* (+a))
jusqu'à (*après* (+b,+c)))

Il existe des algorithmes généraux pour décider si de telles formules sont vraies sur un graphe d'état fini [Queille 82, Clarke 83].

L'exécution montrée dans la figure 10, qui correspond à la séquence <a.b.c.a.a>, viole la propriété b).

V. Le protocole corrigé

Corriger le protocole consiste donc à résoudre correctement la collision entre fermetures sans perturber les comportements qui ne présentent pas de collision. Pour que les processus A et B puissent capter la possibilité de collision et la résoudre sans créer de comportements erronés, on introduit dans l'automate qui les modélise, un état supplémentaire.

Ainsi après avoir émis un message de fermeture, un processus passe dans l'état (2) de collision éventuelle. De cet état, un processus retourne à l'état fermé si il reçoit un message de fermeture (la collision a eu lieu dans ce cas). Mais en l'absence de collision, le processus est débloqué par la réception d'un nouveau message de confirmation de fermeture (d et e). Ces messages sont émis par un processus à la réception d'un message de fermeture, si il n'a pas lui-même émis un message de fermeture.

La figure 11 donne des exemples d'exécutions de ce protocole.

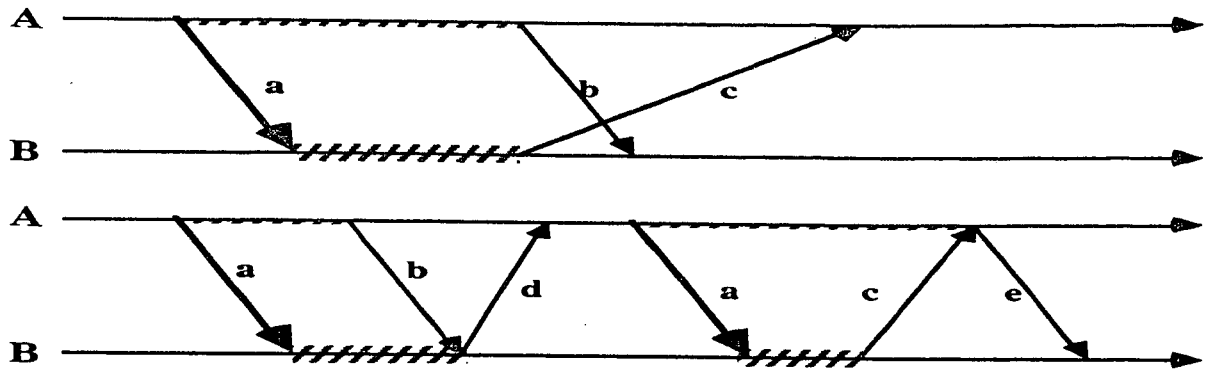


Figure 11: exemples d'exécutions de la version corrigée

La figure 12 présente le modèle du protocole corrigé. Le graphe d'état, montré en figure 13 est fini, le canal AB est borné par 3 et le canal BA par 1.

On peut vérifier alors aisément que l'abstraction de ce graphe est équivalente à celle de la figure 7, et que les propriétés i) et ii) sont assurées.

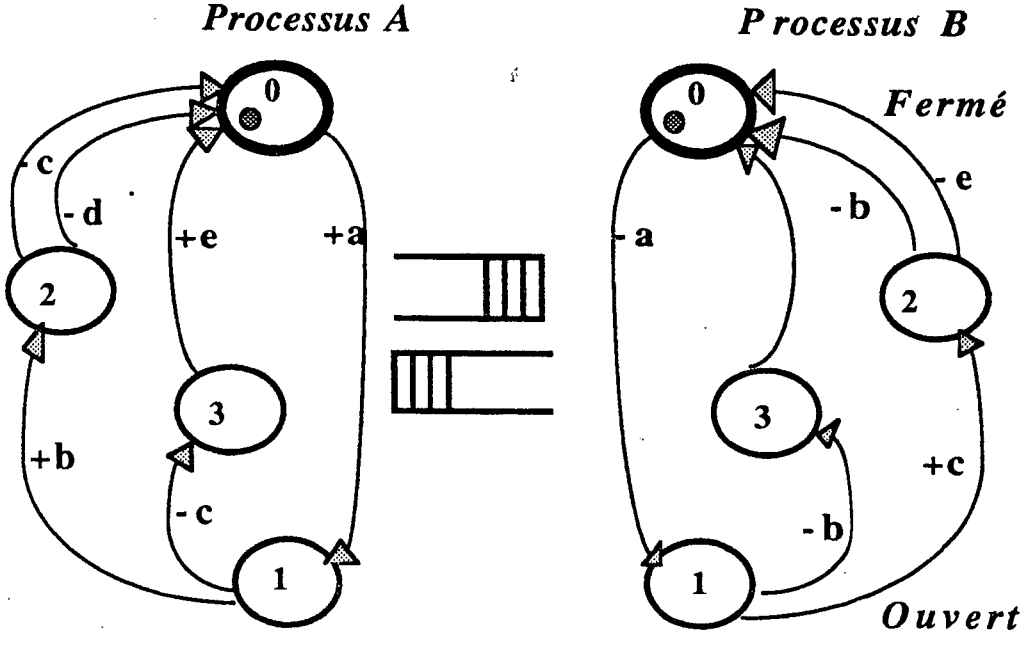


Figure 12: protocole corrigé de connexion-déconnexion

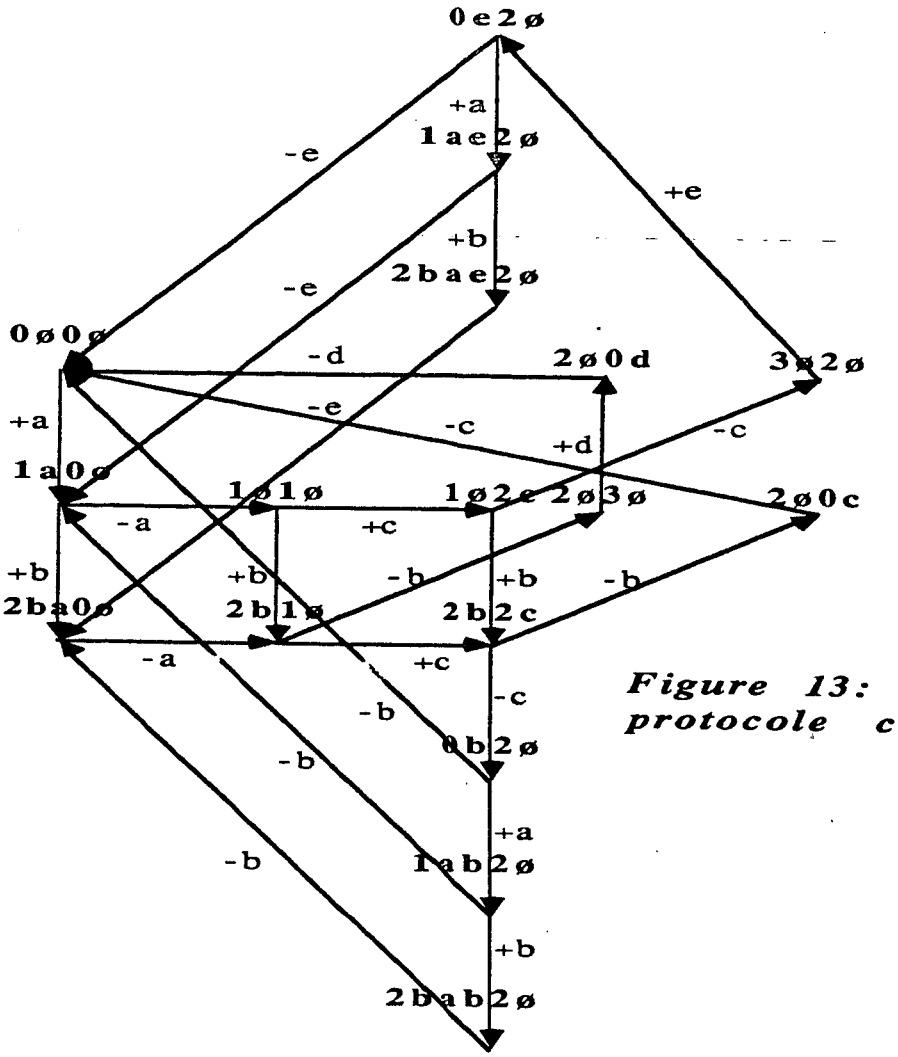


Figure 13: graphe du protocole corrigé

VI. Conclusion

Dans cet article, nous avons illustré, à l'aide d'un exemple simple, un certain nombre de points liés à la modélisation et à la vérification des algorithmes distribués et des protocoles.

Nous nous sommes placés, en ce qui concerne la modélisation, dans le contexte de la théorie des automates; un processus y est modélisé par un automate d'état fini, tout comme une file de communication finie. Rappelons les résultats, mis en évidence par Rubin et West [Rubin 82] et Gouda et Han [Gouda 85] dans ce contexte.

Lorsque au moins l'une des deux files de communication est finie, savoir si le graphe d'état modélisant l'algorithme (qui résulte de la composition des automates modélisant les processus, par les files de communication) est fini, et savoir si l'algorithme présente une possibilité d'interblocage ou de réception non spécifiée, sont des problèmes décidables. Si l'on ne sait rien sur la finitude de l'une des files, rien n'est décidable. Ces résultats revêtent un intérêt pratique comme nous l'avons indiqué.

L'exemple que nous avons exhibé a permis de montrer que la vivacité (non blocage) de la modélisation d'un protocole n'implique pas que celui-ci fonctionne correctement: toutes les propriétés ne se ramènent pas à des propriétés de vivacité; raisonner à l'intérieur du modèle n'est donc en général pas suffisant.

Il est donc nécessaire d'une part de décrire les propriétés attendues de l'algorithme distribué et d'autre part, de vérifier que la modélisation de l'algorithme étudié rend compte de ces propriétés.

Parmi les langages possibles de spécification des propriétés de service, la logique temporelle semble être un outil intéressant. Elle permet en effet de définir la sémantique qui lie les événements dont les occurrences constituent la syntaxe du service rendu par le protocole. La logique temporelle exprime cette sémantique par la donnée de relations temporelles entre les événements.

VII. Bibliographie

[Berthelot 82] G. Berthelot, R. Terrat, *Petri Net Theory for the Correctness of Protocols*, IEEE trans. on Comm., Vol 30, 12, Décembre 1982, pp 2497-2505.

[Bochmann 78] Gv. Bochmann, *Finite State Description of Communication Protocols*, Computer Networks, Vol 2, Octobre 1978, pp 361-372.

[Brand 83] D. Brand, P. Zafiropoulo, *On Communicating Finite State Machines*, Journal of the ACM, Vol 30,2, Avril 1983, pp 323-342.

[Clarke 83] EM. Clarke, EA. Emerson, AP. Sistla, *Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specification: a Practical Approach*, Proc. 10th Symp. of POPL, Austin, Texas, Janvier 1983, pp 117-126.

[Diaz 84] M. Diaz, P. Azéma, *Petri Net based Models for the Specification and Validation of Protocols*, LNCS 188, Springer Verlag, 1984, pp 101-117.

[Gouda 85] MG. Gouda, JY. Han, *Protocol Validation by Fair State Exploration*, Computer Networks, Vol 9, 1985, pp 353-361.

[Gouda 84] MG. Gouda, *Closed Covers: to Verify Progress for Communicating Finite State Machines*, IEEE trans. on SE, Vol 10, 1984, pp 846-855.

[Karp 69] R. Karp, R. Miller, *Parallel Program Schemata*, Journal of Computer and System Science, Vol 3,4, Mai 1969, pp 167-195.

[Pnueli 86] A. Pnueli, *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.

[Queille 83] JP. Queille, J. Sifakis, *Fairness and Related Properties in Transition Systems, a Temporal Logic to deal with Fairness*, Acta Informatica, Vol 19, 1983, pp 195-220.

[Queille 82] JP. Queille, J. Sifakis, *Specification and Verification of Concurrent Systems in Cesar*, Int. Symp. of Programming, LNCS 137, 1982.

[Rubin 82] J. Rubin, CH. West, *An improved Protocol Validation Technique*, Computer Networks, Vol 6, Avril 1982, pp 65-73.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique