L'unisson

Christian Boulinier

L'Unisson

par Christian Boulinier

thèse en vue d'obtenir le grade de
Docteur
de l'Université de Picardie Jules Verne
Spécialité : Informatique

présentée le mercredi 10 octobre 2007
à Amiens

Devant le jury composé de

- Joffroy Beauquier, Professeur à l'Université Paris Sud (rapporteur)
- Yves Métivier, Professeur à l'Universit Bordeaux 1 (président)
- Laurent Fribourg, Professeur à l'ENS de Cachan (examinateur)
- Christian Lavault, Professeur à l'Université Paris Nord (examinateur)
- Sbastien Tixeuil, Professeur à l'Université Pierre et Marie Curie (rapporteur)
- Vincent Villain, Professeur à l'Université de Picardie Jules Verne (co-directeur)
- Franck Petit, Professeur à l'Université de Picardie Jules Verne (co-directeur)

Je n'ai fait [cette lettre] plus longue que
parce que je n'ai pas eu le loisir de la faire plus courte.
Blaise Pascal : *Les provinciales*, lettre XIV[Pas56]

1 Remerciements

Je me souviens, il y a bien longtemps, j'étais en classe de première, monsieur **Mukendi**, mon professeur de mathématiques, m'avait pris sous sa protection. Il était exilé politique, ancien professeur de mathématiques à l'université de Kinshasa. Il avait fui le Congo dans des conditions tragiques et rocambolesques, menacé de mort. Comme il fallait bien gagner sa vie, il était devenu, pour quelques années, maître auxilliaire dans mon lycée. Je lui dois mon amour des sciences, et ma curiosité pour l'Afrique. Il était fou de culture africaine, une de ses principales réflexions portait sur les relations complexes entre les coutumes africaines et les mathématiques. Un jour, heureusement, il a obtenu un poste à l'UNESCO pour poursuivre son oeuvre au service de ce merveilleux continent dont il était originaire. Lors de l'une de nos nombreuses rencontres, il m'avait raconté un conte qui exprimait l'idée que chaque fois que l'on croise quelqu'un, même si on ne lui parle pas, notre vie est changée, et cela pour toujours. Il imageait cela de la manière suivante : un homme n'est pas seulement un corps et une âme, c'est aussi un immense halo, en quelque sorte une boule ouverte centrée en soi. Quand deux boules ouvertes se croisent, elles s'interpénètrent sans même que l'on s'en rende compte, et notre vie est irrémédiablement modifiée - la topologie au service de la psychologie. Nous forgeons notre destin à partir d'un chaos initial et d'une histoire, c'est-à-dire de rencontres. Rencontres de personnes, mais aussi d'œuvres, qui elles aussi transportent leur propre halo. Alors qui remercier ? ou plutôt comment remercier tant de halos croisés, et ce durant tant d'années ?

Je vais limiter mes remerciements à deux sphères, celle de mon environnement scientifique et celle de ma famille.

Je voudrais commencer par remercier **Franck**. Tu fus mon élève, nous sommes devenus amis, tu es maintenant mon directeur de thèse. Je voudrais te remercier pour ces nombreuses discussions que nous avons pu avoir, pour ta disponbilité permanente et ton énergie que j'ai parfois eu du mal à suivre. En ta compagnie, j'ai appris beaucoup de choses et pas seulement du point de vue scientifique, j'ignorais tout de l'algorithmique répartie en arrivant. Nous avons fait du bon boulot ensemble et je souhaite profondément que cette collaboration puisse se poursuivre.
Merci **Vincent** pour les nombreuses discussions que nous avons pu avoir. Merci d'avoir lu entièrement et dans les moindres détails le document présent, l'état final de ce mémoire

te doit beaucoup. J'espère que nous aurons de nombreuses autres occasions de travailler ensemble, le travail accompli fut pour moi un grand plaisir. L'aventure du "PODC" fut une belle aventure partagée avec Franck ; sans vous deux, rien n'aurait été possible pour moi.
Merci à **Ted Herman** que je n'ai rencontré que peu de fois, mais dont les questions simples et directes m'ont fait avancer de manière décisive. Je ne te rencontrerai malheureusement pas à SSS07, mais plus tard certainement.
Merci à toi, **Bernard Piettre**, sans la philosophie dont tu m'as fait découvrir des secrets, je ne me serais sans doute jamais réconcilié avec la science.
Le laboratoire du **LaRIA** m'a ouvert grandes ses portes, et j'ai eu tous les moyens nécessaires pour travailler dans les meilleures conditions. Malheureusement, étant donné mes occupations professionnelles, je n'ai pas pu profiter pleinement de toutes ces conditions, je le regrette. J'ai été peu disponible, j'ai le sentiment d'avoir manqué de nombreuses occasions de discussions scientifiques, en particulier avec les autres thésards, qu'ils veuillent bien m'en excuser. Je voudrais remercier **Gilles**, directeur du LaRIA, pour son soutien sans faille. Merci à tous les membres du laboratoire, notamment **Alain** dont j'ai beaucoup apprécié la culture de logicien et l'imagination. Merci à **Laure**, à **Gilles** et à tous les autres. Merci à **Marie-Lise** pour sa gentillesse.
C'est pour moi un grand honneur que **Joffroy Beauquier** et **Yves Métivier** aient accepté de rapporter cette thèse. Que vous soyez remerciés ici chaleureusement.
Merci, non moins chaleureux, à **Sébastien Tixeuil**, **Gérard Lavaud**, et **Laurent Fribourg** d'avoir accepté de faire partie de mon jury.

Je voudrais maintenant remercier ma famille en brisant, pour une fois et sans doute maladroitement, une pudeur naturelle. D'abord **Marie-Hélène,** ma tendre compagne**,** a subi pendant ces années mes longues absences mentales, tu as fait tourner la maisonnée avec un Zombi à tes côtés ; c'est promis, je ne me lancerai plus jamais dans une aventure aussi longue et je vais rattrapper le retard que j'ai pris sur les tâches ménagères ! Merci de ton soutien, merci d'avoir relu les dernières versions de ce mémoire, ce n'était pas une chose facile, merci de ta confiance et de ton amour. Merci à mes enfants, **Thomas**, **Matthieu** et **Etienne**, vous êtes le sel de ma vie et je vous dédie ce travail. Merci aussi à mes parents, **Anie** et **Paul** (non, il n'y a pas de faute d'orthographe), j'ai été un enfant difficile et vous avez tenu bon : ce mémoire de thèse, c'est à vous que je le dois.

Christian B.

Table des matières

Table des figures

Liste des tableaux

Table des spécifications

Table des programmes

Chapitre 1

Le contexte, le modèle et les contributions

... humans think sequentially,
except perhaps on the football field, the
ballet stage and others kinesthetic activities...
Ted Herman [Her01]

Sommaire

1 Le contexte

1.1 Les systèmes répartis

Les systèmes répartis se sont développés de manière considérable. Chaque processus est autonome et décentralisé. Il communique avec d'autres processus, ses voisins. Chaque processus est représenté par un nœud, et si deux processus communiquent il y a un lien physique ou virtuel entre les deux nœuds associés. On classe souvent les systèmes répartis en trois catégories dont les frontières sont floues [Dol00, Tel00, Tix06] :

1. **Les machines parallèles**. Dans ce cas les différents processus peuvent être lancés sur une même machine, qui peut être multiprocesseurs. Un exemple est la "Connection machine CM-5" de la société Thinking machine Corporation. C'est une machine massivement parallèle d'une capacité de calcul de l'ordre de 1 teraflops (10^{12} opérations en virgule flottante par seconde), elle s'organise en un réseau de 128, 256 ou 512 processeurs, avec trois réseaux de communications, l'un dédié aux données, le deuxième dédié au contrôle du réseau et le troisième consacré au diagnostic des pannes du réseau [LAD+96]. Ces machines massivement parallèles se sont développées à partir des années 1970 jusqu'à la fin des années 1980.

   Un exemple plus contemporain et moins massivement parallèle (mais aussi moins documenté) est la famille des systèmes NovaScale de la société Bull. Ces grands serveurs utilisent la gamme des processeurs Itanium d'Intel. Ils reposent sur une

architecture parallèle à mémoire partagée de deux à trente deux processeurs. Ce type de système présente l'inconvénient d'être très coûteux à développer. Il semble que la tendance actuelle soit de préférer des solutions plus décentralisées.

La société Bull a développé pour le CEA (Commissariat à l'Energie Atomique) une grappe de 544 serveurs Bull NovaScale nommée Tera10, cette grappe est d'une puissance de 60 teraflops et est dédiée à la simulation numérique garantissant la pérénité de la dissuasion nucléaire française après l'arrêt des essais nucléaires. Cela nous conduit à la deuxième catégorie.

2. **Les réseaux locaux et les grappes de serveurs**. Le plus souvent, les différents processus sont localisés sur des machines différentes. Ils bénéficient de systèmes de communications rapides et de faible coût comme *FAST-ETHERNET* ou *MYRINET* [Aum02]. Le réseau est à échelle réduite ; ce peut être un réseau local qui connecte des ordinateurs, des serveurs et des terminaux, ce peut être aussi une grappe de serveurs. Les grappes sont souvent dédiées au calcul scientifique. Les utilisateurs semblent préférer maintenant les réseaux de calcul en paire à paire, ce qui nous fait à nouveau changer d'échelle et passer dans la catégorie des réseaux à grande échelle.
3. **Les réseaux à grande échelle**. Ils se développent maintenant et deviennent des enjeux stratégiques pour le calcul et les échanges. Dans le cas des **Les grilles**[MDN+06], les noeuds peuvent être à grande distance et très nombreux, plusieurs milliers, voire plusieurs centaines de milliers. Un exemple expérimental est la plateforme de recherche Grid 5000 qui développe 5000 Unités de Calcul sur 9 sites en France. Dans ces systèmes à grande échelle, les liens physiques de communication peuvent être des câbles, des fibres optiques, ou des communications hertziennes (via un satellite par exemple). Pour la gestion de ces nouveaux réseaux, se développe une nouvelle approche de la programmation : le métacomputing, qui virtualise l'interconnexion des ressources de calcul réparti pour former une plateforme virtuelle décentralisée.

   Un autre exemple est la gestion des **environnements faiblement couplés**. Un environnement faiblement couplé est caractérisé par l'hétérogénéité des liaisons de communication des agents du réseau, en particulier faibles débits et grande latence de communication, connexions intermittentes et limitation des ressources (cycles disponibles du processeur , mémoire), et parfois partition du réseau (perte de son aspect connexe). L'objectif de tels réseaux est de supporter des applications coopératives avec des données partagées (**réseaux paire à paire** [RM05]) ou d'exploiter les cycles non utilisés des processeurs d'un réseau. Dans ce cas, il s'agit en général de distribuer un calcul dans cet environnement composé d'un grand nombre de machines. Un exemple classique est le projet *SETI AT HOME* d'analyse des signaux captés par le radio-téléscope d'Arecibo sur l'île de Puerto Rico (Université de Cornell), qui est à la recherche d'une "intelligence extraterrestre" . Les données sont enregistrées et transmises à Berkeley en Californie ; 4 serveurs séquencent les données qui sont ensuite transmises aux participants au projet. Le réseau de communication est tout simplement internet. Le site annonce que chaque jour il est effectué l'équivalent de près de 1100 années de calculs sur une station de travail.

Un autre projet moins spectaculaire est le *GIMPS*, acronyme de *Great Internet Mersenne Prime Search*. C'est un projet de calcul partagé de recherche des nombres premiers de Mersenne. Le plus grand nombre connu trouvé par ce projet est $2^{32\ 582\ 657}-1$, découvert le 4 septembre 2006. C'est le 44-ième nombre de Mersenne connu, il est composé de 9 808 358 chiffres décimaux. Une prime de 100 000 dollars est proposée à celui qui trouvera un nombre premier de Mersenne de plus de 10 millions de chiffres décimaux !

De nouveaux types de réseaux à grande échelle se développent maintenant, les **réseaux de mobiles** dits **réseaux ad-hoc** et les **réseaux de capteurs**.

Dans un système réparti asynchrone, que ce soit un réseau local ou un réseau à grande échelle, chaque noeud a son temps propre, son code propre et ses registres propres. Les données sont réparties, chaque noeud n'a qu'une connaissance locale du réseau, il ne connaît que ses voisins. Dans un tel système, les communications sont essentielles et la coordination des processus est une tâche cruciale. Un exemple classique de protocole de coordination est l'exclusion mutuelle, qui consiste à faire en sorte qu'un seul processus puisse exécuter son application à un moment donné, tout en assurant que chaque processus puisse une infinité de fois exécuter son code d'application. L'exclusion mutuelle est aussi une notion cruciale dans le cas des systèmes synchrones, comme les machines parallèles ; par exemple, pour la gestion en lecture et écriture d'une mémoire partagée [Dij65, Lam74].

1.2 La modélisation des systèmes répartis

1.2.1 Les modèles de communication

Les modèles de communication sont divers, on trouve dans la littérature 4 grands modèles :

1. **Le modèle à passage de messages** : c'est le modèle le plus courant en algorithmique distribuée. Dans ce modèle, à chaque étape atomique, un processus envoie un message à l'un des noeuds voisins ou reçoit un message d'un des noeuds voisins, mais ne fait pas les deux en même temps.
2. **Le modèle à partage de registres**. Dans ce modèle, un processus peut lire l'état d'un registre d'un de ses voisins, ou écrire dans ses registres, mais ne fait pas les deux actions en même temps.
3. **Le modèle à états** (modèle à mémoire partagée). Il est introduit par Dijkstra en 1974 [Dij74] : à chaque étape atomique, un processus peut lire les variables de ses voisins, faire un calcul et écrire dans ses registres.
4. **Le modèle à réétiquetage de graphe à distance 1**. Ce modèle a été introduit par Mazurkiewicz [Maz97] ; dans ce modèle un pas de calcul atomique dans le réseau est le choix d'une boule de rayon 1 et l'application d'une règle de calcul sur cette boule de rayon 1. Ce modèle se généralise par le modèle de réétiquetage de graphe à distance k. Ces modèles sont essentiellement des modèles de calcul séquentiel sur un graphe. Ces modèles sont simples et permettent d'obtenir de nombreux résultats

d'impossiblilité ou de connaissance minimale sur le réseau afin d'effectuer une tâche donnée. Pour des applications voir [GM01, GM02, GMM04].

Ces quatre modèles sont donnés avec une atomicité croissante. Il est clair qu'un résultat positif à un certain niveau d'atomicité est encore vrai avec un grain atomique plus gros. De même, il est clair qu'un résultat d'impossibilité pour un certain niveau d'atomicité demeure dans le cas d'une atomicité plus fine.
[Dol00] montre que dans le cas des réseaux bidirectionnels, on peut simuler le *modèle à partage de registre* dans le *modèle à passage de messages*, et cela de manière auto-stabilisante. [NA02] montre que l'on peut simuler le modèle à états dans le modèle à partage de registres, et cela de manière autos-tabilisante aussi. Dans le chapitre 8 de cette thèse, nous donnons un transformateur qui permet de simuler avec un fort parallèlisme le *modèle à réétiquetage de graphe à distance k* dans le *modèle à états*.
En principe, moins le modèle est fin, plus l'analyse et la résolution des problèmes sont "aisées". Comme nous l'avons déjà dit, les résultats d'impossibilité dans un modèle sont valables pour les modèles plus fins. Mais aussi la résolution d'un problème dans un modèle ouvre une piste pour résoudre ce problème dans un modèle à grain plus fin. On trouvera dans ce travail un exemple intéressant : celui d'un algorithme de construction d'un arbre couvrant en largeur d'abord qui est optimal en espace et en temps (pas de compromis entre les deux types de complexités, contrairement à la littérature [Lyn96a]). L'algorithme proposé est une conséquence de l'étude effectuée au chapitre 2, il s'exprime "naturellement" dans le modèle à états. Son écriture et sa correction dans le modèle par passage de messages ne pose pas de problème, une fois l'algorithme défini dans le modèle à états.
Dans le cas de réseaux non bidirectionnels, ou de réseaux avec collisions dans les communications (capteurs sans fils et réseaux ad-hoc), d'autres modèles sont envisagés dans la littérature, voir [Tix06].

1.3 La modélisation des fautes.

1.3.1 Qu'est-ce qu'une faute, ou une panne ?

Dans de tels systèmes, à mesure que le nombre de composants augmente, le risque de panne augmente aussi.[Ray92, Tix06] proposent une taxonomie des pannes dans les systèmes répartis suivant deux critères, celui de leurs occurrences dans le temps et celui de leur nature. Commençons par les occurrences de pannes arbitraires :

1. les **pannes transitoires** : il existe un moment de l'exécution à partir duquel elles n'apparaissent plus ;
2. les **pannes définitives** : il existe un moment de l'exécution à partir duquel elles ne disparaissent pas ;
3. les **pannes intermittentes** : elles peuvent apparaître ou disparaître à tout moment de l'exécution ;

Tixeuil distingue les fautes sur l'état d'un processus ou sur le code de celui-ci et les fautes sur son exécution :

1. **corruption d'état** : le changement de l'état d'une ou plusieurs variables ;
2. **panne de code** ou **panne d'exécution** :
   (a) **panne crash** ou **panne franche** : à un moment donné de l'exécution, un élément cesse définitivement son exécution ;
   (b) les **omissions** : un processus omet de communiquer, soit en émission, soit en réception ;
   (c) les **pannes byzantines** : pannes de type arbitraire de pannes, ce sont donc les plus malicieuses.

On a les inclusions :

$$\textbf{panne crash} \subset \textbf{omission} \subset \textbf{panne byzantine}$$

On peut ajouter aux pannes d'exécution les **pannes de transmission** : les **pertes de messages**, les **duplications d'émission** et les **déséquencements en émission** (émission dans le désordre).

1.3.2 La détection des fautes

Détection des corruptions d'état
Une méthode peut consister à faire régulièrement une photographie instantanée (snapshot) du système pour évaluer les fautes. Il s'agit d'une approche globale peu appropriée aux réseaux à grande échelle. Une catégorie intéressante de corruption d'état est celle des pannes localement contrôlables (locally checkable) [AKY90]. Par exemple, sur un arbre en largeur, si on code la distance à la racine et si l'arbre est incorrect, il existe un noeud dont la distance à la racine n'est pas égale à celle de son père plus 1. Dans ce mémoire, nous montrons que la correction de l'unisson est localement contrôlable à certaines conditions simples (Voir page 69).
Contourner les pannes de code
Cette section n'est qu'une esquisse, sa structure est directement inspirée de [Ray04], dans lequel sont considérés deux types de modèles de communication :

1. Une ligne de communication entre un processus p et un processus q est *fiable* si elle ne crée ni ne duplique des messages, et si tout message envoyé par le processus p vers le processus q finit pas être reçu par q, si q est correct.
2. Une ligne de communication entre un processus p et un processus q est *avec pertes équitables* si elle ne crée ni ne duplique des messages, mais peut perdre des messages ; cependant, si p envoie une infinité de messages à q, alors si q fait une infinité de réceptions demessages alors il reçoit une infinité de messages provenant de p.

et deux types de modèles de calcul asynchrone :

1. Le modèle FLP, pour lequel les processus sont susceptibles de crashs définitifs avec des transmissions fiables.

2. Le modèle FLL, pour lequel les processus sont susceptibles de crashs définitifs et tel que les transmissions sont avec pertes de messages équitables.

Le modèle FLP (pour Fisher Lynch et Paterson) est le modèle historique de [FLP85], c'est le modèle que nous considérons dans cette esquisse. Pour le modèle FLL (pour *Fear Lossy* Link) voir [FRT04, ACT00, Ray04].
Précisons les spécifications du consensus :

Spécification 1 [Consensus]

1. Sûreté :
   - Accord : si deux processus corrects décident, ils décident de la même valeur.
   - Validité : si un processus correct décide, sa valeur de décision est une des valeurs initiales.
2. Vivacité :
   - Terminaison : tous les processus corrects décident.

Dans un système asynchrone chaque processus a son temps propre (time free) et chaque lien a aussi son temps propre. Lors d'une exécution, il est impossible pour un processus de savoir si un voisin est en panne ou s'il est simplement en retard. Fischer, Lynch et Paterson ont montré que dans un réseau asynchrone sujet à des pannes définitives, le problème du consensus est impossible si une seule panne est possible [FLP85]. Pour le prouver, ils étudient le problème du consensus sur un bit, et ils introduisent la notion d'état bivalent, à partir duquel la décision 0 et la décision 1 sont possibles. Il reste à montrer qu'il existe une exécution où tous les états sont bivalents, ce qui est fait par récurrence. L'impossiblilité en résulte.
Pour contourner cette impossibilité, plusieurs approches ont été explorées : l'algorithme de Paxos a été proposé par Lamport [Lam98]. Il n'aboutit pas nécessairement mais les hypothèses sont très faibles. Pour une présentation détaillée voir [Lam01]. Cet algorithme utilise un ensemble de leaders et d'agents. Le consensus est atteint si, durant un temps assez long, il n'y a qu'un leader. Le concept de synchronisme partiel est développé dans [DLS88], il est intermédiaire entre les systèmes synchrones où une borne est connue sur les temps de transmission et sur la vitesse des processus, et les systèmes asynchrones où aucune borne n'existe. Dans les systèmes partiellement synchrones, les bornes existent et ne sont pas connues, ou les bornes existent ultimement, c'est-à-dire à partir d'un certain moment seulement, et ne sont pas connues. Pour une approche probabiliste du problème voir [Asp03].
Une autre approche très féconde fut l'introduction des détecteurs de pannes non fiables par Chandra et Toueg [CT96]. Carole Delporte et Hugues Fauconnier combinent les deux approches avec la notion de système à phases synchronisées [DGF98, DGF99], où pendant des phases "assez longues" le comportement du système est synchrone. Ces différentes approches reviennent à introduire une forme de synchronisme dans le réseau. Les détecteurs de pannes doivent être considérés comme des abstractions, des oracles qui permettent de résoudre des problèmes malgré les crashs. Le choix d'un détecteur détermine un modèle de synchronisme.

Une bonne méthode pour comparer différents modèles de synchronisme consiste à examiner un problème particulier à l'aune de ces modèles. Ce peut être le problème du *consensus*, le problème de la *diffusion*, ou encore le problème de la *consistence interactive*, pour une définition et une hiérachie entre ces problèmes parmi d'autres problèmes voir [Ray04].
Suivant [FLP85], nous ne traitons ici que la question du consensus. Comment parvenir à un consensus entre des processus en présence de pannes définitives. Dans un système synchrone, c'est-à-dire tel que "la vitesse des noeuds est bornée", on peut détecter les pannes définitives par un mécanisme de limite de temps (ou time-out). Dans un système asynchrone on peut aussi implanter un système de time-out, mais alors celui-ci ne sera pas fiable à cause de la possibilité de processus lents. En asynchrone, les algorithmes fondés sur le time-out donnent des exemples de détecteurs de panne non fiables. Dans la théorie, on abstrait l'implantation du détecteur de panne, il devient un oracle distribué sur le système. Plus le système est synchrone, plus l'oracle est fiable.

Hiérarchisation des détecteurs de pannes

[CT96] définit formellement la notion de détecteur de pannes : un détecteur associé à un processus p donne en permanence à celui-ci une liste de processus suspectés d'être en panne. Un détecteur de pannes peut être vu comme un service réparti composé de détecteurs locaux attachés à chaque processus. Il fournit sur demande une liste des processus qu'il soupçonne d'être défaillants. Ces détecteurs sont des abstractions, la question de leur implantation n'est pas posée. Il est fait l'hypothèse que les pannes sont franches (crash) et sans reprise de communication fiable asynchrone.
Les qualités possibles des détecteurs proposées par [CT96] sont :

1. La complétude
   (a) La complétude forte : tout processus en panne finit par être soupçonné par tout processus correct.
   (b) La complétude faible : tout processus en panne finit par être constamment soupçonné par un processus correct.
2. L'exactitude
   (a) L'exactitude forte : tout processus correct n'est jamais suspecté.
   (b) l'exactitude faible : il existe au moins un processus correct qui n'est jamais suspecté.
   (c) L'exactitude finalement forte : tout processus correct finit par ne jamais être suspecté par aucun processus correct.
   (d) L'exactitude finalement faible : il existe au moins un processus correct qui finit par ne jamais être suspecté par aucun processus correct.

On obtient 8 classes de détecteurs. [CT91] introduit la notion de réduction entre détecteurs de panne. Un détecteur de panne D est réductible au détecteur D' s'il existe un algorithme réparti qui transforme D en D' . On dit alors que D' est plus faible que D, on le note $D \rightarrow D'$. Clairement la relation $\rightarrow$ est une relation de préordre. Dans le cas où $D \rightarrow D'$

		Exactitude			
		forte	faible	finalement forte	finalement faible
Complé -tude	forte	P	S	$\Diamond P$	$\Diamond S$
	faible	Q	W	$\Diamond Q$	$\Diamond W$

TAB. 1.1 – Classes de détecteurs de pannes dans [CT91]

et $D' \rightarrow D$, on dit que les détecteurs sont équivalents, on le note $D' \approx D$. Les relations $P \rightarrow S$, $\Diamond P \rightarrow \Diamond S$, $Q \rightarrow W$ et $\Diamond Q \rightarrow \Diamond W$ suivent évidemment des définitions.
[CT91] montre qu'il y a équivalence entre complétude forte et complétude faible. Donc que $P \approx Q$, $S \approx W$, $\Diamond P \approx \Diamond Q$ et $\Diamond S \approx \Diamond W$. Il ne reste donc plus que 4 classes : P, $\Diamond P$, S et $\Diamond S$.
Dans [CT91] les auteurs démontrent que le consensus est possible dans un environnement avec pannes franches ; en utilisant le détecteur S, ils prouvent que le consensus est obtenu, quel que soit le nombre de pannes obtenues $f < n$ dans un réseau complet (n est le nombre de processus du réseau). Dans le même contexte, ils montrent qu'avec un détecteur $\Diamond S$ le consensus est solvable si et seulement si $f < \frac{n}{2}$. Dans [CT91] ils prouvent que $\Diamond W$ est le détecteur le plus faible pour résoudre le consensus.
Une autre approche est l'introduction du détecteur Ω [CHT92]. Ce détecteur retourne un processus leader avec la sûreté qu'à un moment tous les processus corrects ont le même leader et que le processus leader est correct. Le détecteur Ω est équivalent au détecteur $\Diamond S$.

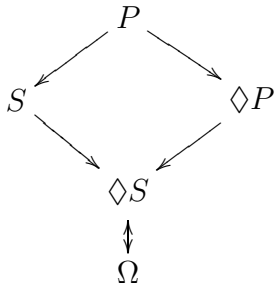


FIG. 1.1 – Hiérarchie entre détecteurs de pannes classiques

2 La synchronisation

Comme nous venons de le voir, les différents modèles de communication et d'ordonnancement peuvent s'interpréter en termes de synchronisme du réseau. Les transformateurs qui permettent de passer d'un modèle à un autre sont en quelque sorte des "synchroniseurs". Pouvoir implanter un détecteur de pannes revient à synchroniser le réseau, ou à faire une hypothèse sur le synchronisme du modèle. En pratique, implanter un détecteur de pannes

revient souvent à implanter un "time-out" éventuellemement *adaptatif*. Faire l'hypothèse que ce détecteur de pannes est ultimement complet et exacte, revient à faire l'hypothèse que les canaux sont fiables et que les temps de communication sont bornés, c'est-à-dire que le réseau est synchrone.

Une forme élémentaire de synchronisation est la réalisation du consensus. Prenons l'exemple de la gestion d'une base de données dupliquées. L'objectif est que toutes les copies de cette base de données B soient identiques. Deux primitives doivent être implantées : l'ajout d'une donnée et la suppression d'une donnée. Ces ajouts et suppressions doivent être faits de manière cohérente et de sorte que toutes les copies soient identiques à B. Pour assurer cela, il suffit de faire en sorte que les mises à jour se fassent sur toutes les copies et dans le même ordre. Cela revient itérer une opération de consensus (ou de diffusion). On peut implémenter cela grâce à la circulation d'un jeton sur un anneau virtuel contenant l'ensemble des sites [Ray90].

La conception et l'analyse d'algorithmes résolvant un problème donné dans les modèles asynchrones sont souvent plus délicates que l'étude de leurs analogues dans un environnement synchrone. D'où l'intérêt de développer une technique générale de synchronisation des réseaux asynchrones. L'idée est de développer un transformateur qui simule un réseau synchrone, et ainsi, en composant ce transformateur avec un protocole qui fonctionne dans un environnement synchrone, obtenir un protocole correct sous un ordonnanceur asynchrone. Un tel transformateur est appelé un synchroniseur [Awe85].

Le but d'un synchroniseur est de générer des pulsations sur chaque site d'un réseau asynchrone comme si celles-ci étaient générées par une horloge globale dont chaque site aurait la valeur instantanément [RH88]. Cela signifie que tous les sites sont à la même pulsation de calcul en même temps, on parle de *synchronisation de phase forte*. Il est facile de voir qu'on peut relâcher la contrainte de simulation de l'horloge globale temps réel en autorisant un processus à passer à la pulsation $i+1$ une fois que tous ses voisins et lui-même ont fini la phase i. On parle alors de *synchronisation de phase faible*.

L'existence d'un tel synchroniseur montre que les tâches que l'on peut faire dans un environnement synchrone peuvent être faites aussi dans un environnement asynchrone. Dans un environnement synchrone, on ne résout pas plus de problèmes que dans le cas asynchrone. Cette équivalence algorithmique n'est plus vraie quand il y a possibilité de pannes dans le réseau [FLP85]. Awerbuch présente en 1985 les synchroniseurs α, β et γ [Awe85].

Le synchroniseur principal introduit par Awerbuch est le $\gamma-$synchroniseur, il est un compromis entre l'α-synchroniseur, qui synchronise faiblement le réseau par des contrôles locaux, et le β-synchroniseur, qui est une barrière de synchronisation utilisant un arbre couvrant. Le α-synchroniseur est bon en temps mais mauvais en nombre de communications ; le β-synchroniseur demande l'existence d'une racine dans le réseau et le précalcul d'un arbre couvrant. Le β-synchroniseur est bon en nombre de communications mais mauvais en temps à cause de la barrière de synchronisation. Le $\gamma-$synchroniseur est un compromis qui exige la construction d'une forêt couvrante : sur chaque arbre est implanté un $\beta-$synchroniseur et un $\alpha-$synchroniseur synchronise les racines des arbres entre elles.

2.1 L'$\alpha-$synchroniseur

L'idée de l' $\alpha-$synchroniseur est extrêmement simple :

1. un site P est sûr relativement à la pulsation i si :
   (a) tous les messages qu'il a émis lors de cette pulsation sont arrivés
   (b) il a terminé le traitement de la pulsation courante i.
2. Le site P peut passer à la pulsation $i+1$ si :
   (a) Il a terminé le traitement de la pulsation i
   (b) Tous les voisins de P sont sûrs pour la pulsation i

Pour ce faire, chaque site exécute les trois phases suivantes [Awe85, RH88] :

1. Traiter et émettre les éventuels messages de P vers ses voisins.
2. Attendre les accusés de réception de tous les messages envoyés.
3. Diffuser à tous les voisins que P est sûr.
4. Attendre que tous les voisins de P soient sûrs.

On peut remarquer que dans une synchronisation de phase faible, un processus dans la phase i peut recevoir des messages d'un processus dans la phase $i+1$, que ce soit un message de l'application qui est synchronisée ou un message de sûreté pour la phase $i+1$. On retrouvera ce phénomène dans les chapitres 7 et 8 à propos de calculs d'infima à distance δ et à propos d'exclusion mutuelle à distance δ.

2.2 Horloge de phase et unisson

L' $\alpha-$synchroniseur est très simple, les phases sont implicites, déterminées par des échanges de messages. Si le système est mal initialisé, il se peut que, malgré la synchronisation, l'exécution ne soit pas en "phase". On peut ajouter à chaque processus un compteur appelé horloge de phase qui s'incrémente à chaque changement de phase. Quand le système démarre et que les horloges ne sont pas à l'unisson, malgré la pulsation apportée par l'alpha synchroniseur, le système peut ne jamais être coordonné.

> **Définition 1** Dans un environnement synchrone, on dit que l'*unisson* est atteint quand tous les processus ont leur horloge de phase à la même valeur (barrière de synchronisation forte). Dans un système réparti asynchrone on dit que l'*unisson* est atteint quand les horloges de deux voisins quelconques ne diffèrent pas d'au plus une unité (barrière de synchronisation faible).
> Dans ce chapitre et seulement dans ce chapitre, en suivant la littérature [Her01], on appellerons *unisson* tout algorithme d'horloge de phase auto-stabilisant vers l'unisson. Nous donnerons une définition définitive de l'unissons dans le chapitre 3, page 94.

L'unisson synchrone ou asynchrone apparaît donc comme un système plus contraint que le synchroniseur. En particulier, il a du sens en environnement synchrone, ce qui peut

paraître paradoxal à première vue. L'élaboration d'un protocole d'unisson peut se faire dans un réseau synchrone ou asynchrone. Le registre de l'unisson peut être non borné ou à valeurs dans l'ensemble $\mathbb{Z}_K$ des entiers modulo K (dans ce cas, nous disons que l'horloge est d'ordre K).
Dans la suite de ce chapitre nous emploierons indifféremment les expressions "unisson" et "horloge de phase auto-stabilisante".
La synchronisation de phase et l'unisson ont été très étudiés dans les années 1990 -2000 :

1. Misra [Mis91] pose clairement le problème de la synchronisation de phase forte. Le réseau est complet, la question d'un registre de phase borné est abordée. L'auto-stabilisation n'est pas abordée.
2. Dans [GH90], Gouda et Herman proposent la première horloge de phase auto-stabilisante, autrement dit le premier unisson. Le registre de l'horloge de phase est non borné. Le réseau est synchrone.
3. Dans [CFG92], Couvreur, Francez et Gouda proposent le premier unisson avec une horloge de phase bornée dans un environnement asynchrone. Un lemme difficile est sans démonstration et il n'y a pas d'analyse de complexité. L'auto-stabilisation utilise un procédé de stabilisation par réinitialisation locale. On trouvera une analyse des algorithmes de ce papier dans les chapitres 2 et 3 de ce mémoire.
4. Dans [HG95], Herman et Ghosh proposent plusieurs algorithmes d'unisson dans un environnement synchrone. Notamment un algorithme à trois états sur l'arbre. On y trouvera également un algorithme probabiliste, malheureusement le temps de convergence de cet algorithme peut être exponentiel.
5. Plusieurs papiers étudient la question de l'implantation d'un unisson sur un anneau [LS95, HL98, HL01].
6. Dans [KA97], Kulkarni et Arora introduisent la tolérance simultanée aux fautes détectables et indétectables. Suite à une faute détectable, on peut engager un processus de réinitialisation globale du système et finallement on peut "masquer la faute" ; dans le cas d'une faute indétectable, on demande que le système regagne un fonctionnement normale au bout d'un certain temps (auto-stabilisation). Dans ce contexte, les auteurs proposent une barrière de synchronisation multitolérante dans un graphe complet asynchrone. [KA98a] propose une solution sur l'anneau, le tout pour différents niveaux de raffinement du modèle de communication.
7. Dans [NV01], Nolot et Villain proposent, en environnement asynchrone, un synchroniseur global instantanément stabilisant sur l'arbre pour une horloge de phase d'ordre quelconque supérieur à trois.
8. Dans [Dol00], Dolev présente l'unisson à horloge de phase non bornée de [CFG92] comme un α synchroniseur auto-stabilisant. Il présente un unisson à horloge de phase bornée d'ordre $K > n^2$ où n est le nombre de processus du réseau. Il utilise une méthode de stabilisation différente de [CFG92], à savoir un procédé de réinitialisation globale. Aucune analyse de complexité n'est proposée.

Remarque

Dans ce mémoire, nous ne définirons pas l'unisson comme barrière de synchronisation asynchrone (resp. synchrone) auto-stabilisante. Au chapitre 3 (voir page 94) et au chapitre 4 (voir page 114), nous donnerons une spécification légèrement plus contrainte de l'unisson dans un environnement asynchrone (respectivement synchrone). L'idée est d'imposer une contrainte de synchronisation dés la convergence vers la barrière de synchronsation asynchrone (respectivement synchrone). Cela permettra d'obtenir dans certaines applications des protocoles qui seront instantanéments stabilisants en utilisant un unisson. Dans ces applications, l'unisson assure la synchronisation quelque soit l'état de départ, la convergence assurant le maximum de concurrence. Il va sans dire qu'un protocole qui satisfera la nouvelle spécification restera une barrière de synchronisation auto-stabilisante, la réciproque pouvant être fausse. On trouvera dans le tableau 1.2 les différentes notions qui seront définies et utilisées dans ce mémoire, ainsi que les pages où ces notions sont spécifiées.

	Barrière	Barrière auto-stabilisante	Unisson
Réseau synchrone	Barrière de synch. forte, page 114	B. de synch. forte auto-stabilisante	Unisson synchrone page 114
Réseau asynchrone	Barrière de synch. faible, page 93	B. de synch. faible auto-stabilisante	Unisson asynchrone page 94
			Unisson asynchrone silencieux, page 159
	Barrière de synch. forte, page 93	B. de synch. forte auto-stabilisante	
	Barrière de synch. forte à distance ρ, page 167	B. de synch. forte à distance ρ auto-stabilisante	

TAB. 1.2 – Tableau des différentes notions de synchronisation avec horloge de phase

3 Le modèle

Un système réparti (ou distribué) est un ensemble S de processus. Chaque processus p ne peut communiquer qu'avec une partie des processus, appelée ensemble des voisins de p, notée $\mathcal{N}_p$. On suppose que les communications sont bidirectionnelles, c'est-à-dire que si le processus p est voisin du processus q alors q est voisin de p. Le réseau ainsi constitué est représenté par un graphe $G = (V, E)$ où V est l'ensemble des processus et E est l'ensemble des arêtes : $E = \{\{p, q\}, q \in \mathcal{N}_p\}$. Le réseau est donc représenté par un 1-graphe non orienté simple (sans boucle).

3.1 Système de transitions sur un ensemble de processus

Soit $G = (V, E)$ un graphe d'interconnexion de processus.
On note n le nombre de processus, et C_p l'ensemble des états du processus p. Une configuration γ d'un système distribué est une instance de l'état de chacun des processus, donc un élément du produit cartésien $\Gamma = \prod_{p \in V} C_p$. Γ est donc l'ensemble des configurations possibles du système. On note $\gamma.p$ l'état du processus p dans la configuration γ. On a bien sûr $\#(\Gamma) = \prod_{p \in P} \#C_p$.

Définition 2 Un système de transitions sur le graphe G est un triplet (Γ, δ, I) où :

1. δ est une relation binaire sur $\Gamma \times \Gamma$.
2. I est une partie non vide de Γ (ensemble des états initiaux).

Une *étape du calcul* est un couple de configurations (γ_1, γ_2) tel que $(\gamma_1, \gamma_2) \in \delta$. On notera la transition $\gamma_1 \xrightarrow{\delta} \gamma_2$ ou plus simplement $\gamma_1 \rightarrow \gamma_2$ s'il n'y a pas d'ambiguité.
On dit que la transition $\gamma_1 \xrightarrow{\delta} \gamma_2$ *active* p si $\gamma_1.p \neq \gamma_2.p$.
Un processus est *activable* pour la configuration γ s'il existe une transition $\gamma \xrightarrow{\delta} \gamma'$ qui active p.

3.2 Graphe des transitions

Soit un système de transitions (Γ, δ, I). δ définit un graphe orienté sur Γ noté (Γ, δ), c'est le graphe des transitions du système de transitions (Γ, δ, I). L'une des tâches de l'analyse d'un système de transitions est l'étude de la topologie de (Γ, δ). Introduisons un peu de vocabulaire ; un système de transitions pouvant être vu comme un système dynamique, nous empruntons le vocabulaire à cette théorie :

Définition 3

On dit qu'une configuration γ est *finale* ou *terminale* s'il n'y a aucun arc sortant de γ.

On dit qu'une configuration γ est *sans retour* si γ n'est pas *final* et qu'aucun circuit ne passe par γ.

On dit qu'un sous-graphe induit de (Γ, δ) est une composante *stable* ou *close* si il est sans arc sortant (le *cocycle* associé ne contient que des arcs entrants).

On appelle *composante récurrente* du graphe toute composante *fortement connexe stable*.

Définition 4

On dit qu'une configuration γ est *attirée* par une *composante stable* C *si* et seulement si tout *chemin maximal* (fini ou infini) issu de γ passe par C.

On appelle *bassin d'attraction* d'une *composante stable* C le sous-graphe de (Γ, δ) induit par les configurations *attirées par* C. On le note $Bassin(C)$.

3.3 Exécution d'un système de transitions

A partir d'une condition initiale donnée, l'ensemble des exécutions possibles est un arbre, éventuellement infini.
Ainsi une séquence d'exécution maximale est un mot $e = \gamma_{t_0}\gamma_{t_1}....\gamma_{t_n}...$(fini ou infini) sur l'alphabet Γ satisfaisant :

$$\gamma_{t_0} \rightarrow \gamma_{t_1} \rightarrow .... \rightarrow \gamma_{t_n} \rightarrow ...$$

Si la séquence est finie, alors le dernier état du mot e est un état final. La suite des indices est une suite (finie ou infinie) strictement croissante d'entiers. Dans la transition $\gamma_{t_0} \rightarrow \gamma_{t_1}$: t_1 est la date d'exécution de la transition $\gamma_{t_0} \rightarrow \gamma_{t_1}$. Si t est un indice, $t \in [t_1, t_2]$ signifie t est un indice compris entre t_1 et t_2.

3.4 Fermeture, équité, convergence, et interblocage

Soit L une partie de Γ, L est en général définie par un prédicat récursif sur Γ. On confond L et le sous-graphe induit par L.
Un système de transitions est *équitable pour* L ssi pour toute exécution maximale $\gamma_{t_0}\gamma_{t_1}....\gamma_{t_n}...$ il existe t tel que $\gamma_t \in L$, ou encore, dans le langage des systèmes dynamiques :

L est une composante périodique de Γ.

En particulier toute exécution infinie passe une infinité de fois par L.
Un système de transitions est *convergent vers* L ssi il est *équitable* pour L et L est *fermé*, ou, dans le langage des systèmes dynamiques :

L est une composante stable de bassin d'attraction Γ.

Un système de transitions est *sans interblocage* si et seulement si toute exécution maximale est infinie, ou encore :

Γ ne contient pas de configuration finale

Il arrive souvent que l'ensemble des configurations finales soit séparé en deux parties, l'ensemble F des configurations finales correctes et l'ensemble des états finaux incorrects. F vide signifie que l'algorithme ne s'arrête jamais.
Si F est non vide, le système de transitions termine (correctement) en partant de L si et seulement si

$$L \subset Bassin\,(F)\,.$$

3.5 Temps de convergence et temps de service en nombre d'étapes en pire cas

Soit L une partie de Γ telle que le système de transitions soit auto-stabilisant pour L, on appelle *temps de convergence* le nombre maximal de transitions nécessaires pour atteindre L. C'est la longueur du plus long chemin de Γ_L +1.

Soit L une partie de Γ telle que le système de transitions soit équitable pour L, on appelle *temps de service* le nombre maximal de transitions nécessaires entre deux passages dans L.

3.6 Systèmes répartis

On modélise un système réparti (ou système distribué) par la donnée d'un graphe des processus $G = (V, E)$. L"ensemble des états d'un processus p est noté C_p. L'ensemble des états du système est noté Γ. Chaque processus p exécute un programme local qui est une suite d'actions conditionnelles du type

$$\langle étiquette \rangle : \langle garde \rangle \rightarrow \langle action \rangle$$

1. L'*étiquette* identifie une règle.
2. Chaque *garde* est une fonction booléenne mettant en jeu les variables et les constantes de p et de ses voisins.
3. L'*action* est un calcul utilisant les variables et les constantes de p et de ses voisins. Les actions d'affectation se font sur les variables de p.

Si plusieurs gardes sont simultanément vraies, alors elles s'exécutent séquentiellement dans l'ordre de leur écriture.
Les actions d'évaluation des gardes et d'exécution de l'action correspondante se font de manière atomique. L'ensemble des processus munis chacun d'un algorithme local constitue un *système réparti*. L'ensemble des algorithmes locaux constitue un *algorithme réparti*, ou *protocole réparti*. Un processus p est activable pour la configuration γ quand le programme de p a une garde vraie, l'action correspondante est alors *activable* pour p. Si p n'a aucune garde activable alors p n'est pas activable. On note $\mathfrak{E}_\gamma$ l'ensemble des processus activables dans l'état γ. L'ensemble des transitions définit un système de transitions $(\Gamma, \rightarrow)$.

Définition 5
Un système réparti (ou plus simplement un réseau) est :

1. *anonyme* si tous les processus de même degré sont identiques (même registres, même programme) ; en particulier, les processus n'ont pas d'identitificateur unique ;
2. *uniforme* si le réseau est anonyme et que tous les processus commencent dans le même état [BV97] ;
3. *enraciné* (ou *semi-anonyme*) si un processus se distingue des autres par un programme différent, il est appelé la racine du système.

3.6.1 Exécution et ordonnanceur

On suppose que chaque transition d'une configuration à une autre est choisie par un ordonnanceur distribué (ou démon). L'ensemble des processus qui feront une action à partir

de l'état γ est noté $\mathfrak{D}$, cet ensemble est choisi par l'ordonnanceur de manière que : $\mathfrak{D} \subset \mathfrak{E}_\gamma$ avec $\mathfrak{D} \neq \emptyset$, si cela est possible. Si un tel choix n'est pas possible, cela signifie que l'état γ est terminal. Lors de la transition $\gamma \rightarrow \gamma'$, les processus appartenant à $\mathfrak{D}$ exécutent leurs actions activables. On pourra noter la transition par $\gamma \xrightarrow{\mathfrak{D}} \gamma'$ quand cela sera utile (remarquer que cette notation est redondante puisque la transition définit l'ensemble $\mathfrak{D}$). Une séquence d'états $e = \gamma_0, \gamma_1, \ldots, \gamma_i, \gamma_{i+1}, \ldots$ est appelée une *exécution* si $\forall i \geq 0, \gamma_i \rightarrow \gamma_{i+1}$.
L'ordonnanceur ordonne l'exécution. De manière générale, il est défini en fonction de deux critères, un critère de répartition spatiale et un critère d'équité dans le temps.

3.6.1.1 Répartition spatiale du choix de l'ordonnanceur .

La répartition du choix d'un démon spécifie le niveau de parallélisme. Avec $\mathfrak{E}_\gamma$, l'ensemble des processus activables dans l'état courant γ, les hypothèses les plus courantes sont :

1. **l'ordonnanceur distribué** (asynchrone) (D_a) : à chaque étape de calcul, l'ordonnanceur choisit pour $\mathfrak{D}$ n'importe quelle partie non vide de $\mathfrak{E}_\gamma$;
2. **l'ordonnanceur global** (synchrone) (D_s) : à chaque étape de calcul, l'ordonnanceur choisit systématiquement $\mathfrak{D} = \mathfrak{E}_\gamma$;
3. **l'ordonnanceur central** (séquentiel) : à chaque étape de calcul, l'ordonnanceur choisit systématiquement un processus unique dans $\mathfrak{E}_\gamma$;
4. **l'ordonnanceur ρ-central** : à chaque étape, l'ordonnanceur choisit un sous-ensemble non vide des processus activables, satisfaisant la propriété que deux processus choisis ne sont pas à distance inférieure ou égale à ρ l'un de l'autre ;

Les ordonnanceurs centraux et distribués sont des cas particuliers de l'ordonnanceur ρ-central : prendre $\rho \geq d$ pour l'ordonnanceur central et $\rho = 0$ pour l'ordonnanceur distribué. Si on prend $\rho = 1$, on trouve l'**ordonnanceur localement central** [Tix06]. Pour construire un transformateur qui simule un démon **ρ-central** sous un ordonnanceur distribué quelconque, il suffit de disposer d'un protocole d'exclusion mutuelle. L'inconvénient d'une telle solution est qu'elle ramène l'exécution à une exécution séquentielle (ordonnanceur global). Une question importante est de trouver un tel transformateur qui conserve un certain parallélisme de l'exécution. Dans le chapitre 8 nous donnons un transformateur efficace au sens du parallélisme, qui simule un démon **ρ-central.**

3.6.1.2 Equité de l'ordonnanceur :

Sur l'ordonnanceur asynchrone il peut être réaliste d'imposer une contrainte de temps sur le choix des processus par l'ordonnanceur, c'est-à-dire une condition d'*équité* sur les processus :

1. **ordonnancement inéquitable** (arbitraire) : le choix de l'ordonnanceur est quelconque, la seule hypothèse est la progression (au moins un noeud activable est choisi, s'il existe) ;

2. **ordonnancement faiblement équitable** : toute exécution maximale ne contient aucun suffixe dans lequel un processus continuement activable n'exécute jamais d'action ;
3. **ordonnanceur fortement équitable** : toute exécution maximale ne contient aucun suffixe dans lequel un processus infiniment souvent activable n'exécute jamais d'action ;

Il est clair que l'ordonnancement synchrone est un ordonnancement fortement équitable. Pour passer d'un modèle faiblement équitable vers un modèle inéquitable, il suffit de disposer d'un protocole d'exclusion mutuelle locale. Nous proposons un tel protocole au chapitre 8.

3.6.2 Complexité en temps et notion de ronde

Un objectif important est d'évaluer la complexité en temps d'un protocole. Si l'ordonnanceur est synchrone alors la complexité en temps est évaluée en nombre de transitions (ou pulsations). Si l'ordonnanceur est asynchrone, alors le nombre de transitions n'est pas significatif. Nous utiliserons la notion de ronde introduite dans [DIM97]. Cette définition a été remaniée dans [CDPV02] avec l'introduction de la notion de neutralisation de règle.

> **Définition 6** Un processus p subit une *neutralisation* durant la transition $\gamma \xrightarrow{\mathfrak{D}_\gamma} \gamma'$ si p est activable dans l'état γ ($p \in \mathfrak{E}_\gamma$) et n'est plus activable dans l'état γ' ($p \notin \mathfrak{E}_{\gamma'}$) sans avoir exécuté aucune règle durant la transition ($p \notin \mathfrak{D}_\gamma$).

En utilisant la notion de neutralisation, la notion de ronde se définit par :

> **Définition 7** Etant donné une exécution maximale $e = \gamma_0, \gamma_1, \ldots, \gamma_i, \gamma_{i+1}.....$, la première ronde de e est le préfixe minimal de e, noté e', contenant, pour chaque processus de $\mathfrak{E}_{\gamma_0}$, l'exécution d'une de ses règles ou la neutralisation du processus. Si $e = e'e''$, la seconde ronde de e correspond à la première ronde de e'', et ainsi récursivement.

La ronde évalue le temps d'exécution par rapport au processus le plus lent. Dans notre modèle, c'est l'ordonnanceur qui détermine la lenteur. Le fait qu'un processus non activable ne puisse pas agir ne dépend pas de l'ordonnanceur. Il est donc "naturel" qu'un processus qui n'est plus activable soit exclu de la ronde.

3.6.3 Attracteur.

Soit X un ensemble. Un *prédicat* P sur X est une fonction booléenne définie sur X, on le confond avec son support : $P = \{x \in X,\ P(x)\}$. Un prédicat P, défini sur Γ, est *clos* pour le graphe des transitions sur Γ si et seulement si pour toute transition $\gamma \rightarrow \gamma'$, si $\gamma \in P$ alors $\gamma' \in P$. Un prédicat Q est un *attracteur* pour le prédicat P, noté : $P \rhd Q$, si et seulement si :

1. Q est clos pour Γ ;
2. pour toute exécution maximale $e = \gamma_0, \gamma_1, \ldots, \gamma_i, \gamma_{i+1}.....$ avec $\gamma_0 \in P$, alors il existe une configuration γ_j de cette exécution telle que $\gamma_j \in Q$.

Un graphe de transitions (Γ, δ) est *auto-stabilisant* pour le prédicat P si et seulement si P est un attracteur du prédicat $Vrai$, i.e. $Vrai \triangleright P$.

4 L'auto-stabilisation

4.1 Définition

En général les spécifications d'un problème ne se définissent pas en termes d'*état* correct ou incorrect mais seulement en termes de *comportement* correct ou incorrect. Prenons les spécifications de l'*exclusion mutuelle* :

> **Spécification 2**
>
> 1. **Sûreté** : deux processus différents n'exécutent pas leur section critique simultanément.
> 2. **Vivacité** : dans toute exécution maximale, si un processus demande à entrer en section critique alors il finira par entrer en section critique.

On voit sur cet exemple que la spécification ne porte pas sur un état, mais sur la suite des transitions d'une exécution.

> **Définition 8** Un système est dit *auto-stabilisant* pour une spécification S si, démarrant dans un état arbitraire, il finit par avoir le *comportement attendu*.

Dans cette définition nous entendons *comportement attendu* dans le sens de *comportement correspondant aux spécifications données*. La spécification S définit un ensemble L d'exécutions légales du protocole. Ainsi un protocole auto-stabilisant pour la spécification S est un système réparti tel que toute exécution maximale contient un suffixe non vide qui est dans L [Dol00].
Une configuration est *légitime* si à partir de cette configuration le comportement du système est conforme aux spécifications. Un état n'est pas légitime si à partir de cet état le système peut ne pas se comporter légitimement par rapport aux spécifications. Nous utilisons l'expression "peut" car le système n'est pas déterministe à cause des choix du démon. Montrer l'auto-stabilisation d'un système pour une spécification revient souvent à déterminer un ensemble d'états légitimes EL_1, de montrer que cet ensemble est clos pour le protocole et qu'il admet pour bassin d'attraction l'ensemble Γ, ou en d'autres termes que $Vrai \triangleright EL_1$. Remarquons que EL_1 n'est pas nécessairement l'ensemble des états légitimes EL, mais seulement une partie close ayant pour bassin d'attraction Γ.
Le *temps d'auto-stabilisation* est le temps de convergence vers EL, il est majoré par le temps de convergence vers EL_1. Dans le cas d'un système synchrone, le temps de stabilisation est compté en nombre de *pulsations*. Il est compté en nombre de *rondes* dans le cas des systèmes asynchrones.

> **Définition 9** Un protocole auto-stabilisant est silencieux s'il converge vers un état global légitime où les valeurs des variables locales des processus ne changent plus, tant que le code applicatif d'aucun processus ne fasse une demande de "démarrage".

Remarquons que dans le cadre de l'auto-stabilisation, un protocole silencieux n'est pas silencieux en messages, puisqu'il y a nécessairement un contrôle ad libitum par chaque processus de la cohérence de l'état de ses voisins avec lui-même.

4.1.1 Stabilisation instantanée

Le concept de *stabilisation instantanée* apparaît en 1999 dans [BDPV99].

> **Définition 10** Quel que soit l'état initial du système, un protocole *instantanément stabilisant* garantit que le système satisfait toujours ses spécifications (le temps de stabilisation est nul).

4.2 Composition d'algorithmes auto-stabilisants.

Quand on analyse une tâche à réaliser, il est souvent efficace de décomposer cette tâche en une séquence de tâches plus simples, chacune tirant parti des précédentes. Supposons qu'une tâche spécifiée par T soit décomposée en deux tâches spécifiées respectivement par T_1 et T_2. Supposons que nous ayons un protocole auto-stabilisant satisfaisant T_1 et un protocole auto-stabilisant satisfaisant T_2 sachant que T_1 est satisfait. En algorithmique distribuée non auto-stabilisante, il serait nécessaire de détecter la terminaison du premier protocole pour lancer le deuxième. Dans le cas de protocoles auto-stabilisants, cette détection de terminaison n'est pas nécessaire, peu importe ce que fait le deuxième protocole tant que le premier n'est pas stabilisé.
La composition de deux protocoles a été introduite dans [Her91, DIM93], voir aussi [Dol00].

> **Définition 11** Soient P_1 et P_2 deux protocoles, les variables de P_1 sont utilisées par P_2 commes données. Les variables de P_2 ne sont pas utilisées par P_1. On désigne par $P_2 \circ P_1$ la composition de ces deux protocoles. Cette composition est définie de la manière suivante :
>
> 1. $P_2 \circ P_1$ contient toutes les variables de P_1 et de P_2 ;
> 2. les constantes de P_1 sont des constantes de $P_2 \circ P_1$;
> 3. certaines variables de P_1 sont identifiées comme constantes de P_2 ;
> 4. les régles gardées de $P_2 \circ P_1$ sont les règles de P_2 et de P_1 , dans cet ordre.

En général, les gardes des actions de P_1 et de P_2 ne s'excluent pas mutuellement.
Il faut prendre garde au fait qu'un protocole peut être auto-stabilisant sous un démon inéquitable, et ne plus l'être dans une composition. Imaginons, par exemple, que le deuxième protocole soit une tâche simple sans condition : un "do forever". Sous un démon non équitable, un processus particulier peut être choisi systématiquement par le démon. Dans ce cas, le premier peut ne jamais se stabiliser. La contrainte d'une exécution équitable devient alors nécessaire :

> **Définition 12** Une exécution maximale e de la composition de P_1 et P_2 est *équitable* pour P_i si e contient une infinité d'exécutions de règles de P_i ou contient un suffixe (fini

ou infini) dans lequel aucune règle de P_i n'est activable.

Par projection des exécutions, on a immédiatement le théorème :

Théorème 1 [Tel00, Dol00] Si les quatre conditions suivantes sont vérifiées :

1. le programme P_1 est auto-stabilisant pour la spécification S_1 ;
2. le programme P_2 est auto-stabilisant pour la spécification S, si la spécification S_1 est satisfaite ;
3. les variables de P_2 ne sont pas utilisées par P_1 ;
4. l'exécution est équitable pour P_1 et P_2 ;

alors : $P_2 \circ P_1$ est auto-stabilisant pour la spécification S.

5 Nos contributions et le plan de la thèse.

Il est facile de voir que même dans un univers sans panne, il n'est pas toujours vrai qu'un "unisson" synchronisé soit sans interblocage ou sans famine, au sens où tous les processus exécuteraient le code de l'application une infinité de fois (voir page 59). Ce type de pathologie exige une étude approfondie de l'unisson.

Nos contributions s'articulent autour de trois grands axes :

– comprendre l'unisson et sa convergence vers un état synchronisé sans interblocage dans un graphe quelconque, en déduire des protocoles efficaces sur le graphe général ;
– améliorer les protocoles dans deux cas particuliers : les réseaux synchrones et les réseaux à topologie en arbre ;
– appliquer l'unisson à la synchronisation locale, et plus généralement à la synchronisation à distance ρ.

Ces trois axes motivent les trois parties de cette étude.

5.1 Première partie : stabiliser

Elle se compose des chapitres 2 et 3 et des annexes A et B.

Cette première partie est un développement approfondi de [BPV04]. A part le premier algorithme "naïf" d'unisson (algorithme 1 , page 52) dont les registres sont à valeurs dans $\mathbb{Z}$, les chapitres de cette partie sont entièrement originaux.

5.1.1 Chapitre 2

Une analyse détaillée de l'algorithme "naïf" amène à poser 5 questions. La dernière question, portant sur la convergence, sera abordée au chapitre suivant (Le chapitre 3). Le chapitre 2 apporte des réponses positives et constructives aux quatre premières questions et dégage plusieurs concepts simples, dont on fera un usage important par la suite :

1. la notion d'incrémentation bornée ;
2. la notion de retard sur un réseau ;

3. la notion de résiduel sur un cycle du graphe ;
4. la notion de caractéristique cyclomatique d'un graphe G, notée C_G ;
5. la notion de relèvement.

Ces outils cachés, car triviaux ou inutiles, dans le cas de registres à valeurs dans $\mathbb{Z}$, prennent toute leur importance dans le cas de registres bornés (essentiellement des registres à valeurs dans le groupe modulaire additif $\mathbb{Z}_K$, où K est l'ordre de l'horloge de phase) :
- Une *incrémentation bornée* est une extension simple de l'incrémentation sur $\mathbb{Z}_K$.
- Un unisson synchronisé définit un graphe orienté sur le réseau ; la notion de *résiduel* est une notion algébrique qui caractérise les cycles dans ce graphe orienté.
- La *caractéristique cyclomatique* joue un rôle pivot grâce au résultat suivant : si l'ordre K de l'horloge est strictement supérieur à C_G alors l'incrémentation de phase est sans interblocage et sa correction est localement contrôlable (locally checkable).
- Si $K > C_G$ la notion de *retard* entre deux processus est indépendante du chemin utilisé pour son calcul, on dit que le retard est intrinsèque. Le retard définit alors un préordre total sur l'ensemble des processus.
- La notion de *relèvement* est utilisée dans les chapitres 2,7 et 8. Elle est l'outil théorique qui exprime, entre autres, l'idée simple, mais peut-être surpprenante, que si $K > C_G$, alors l'intuition d'une horloge à valeurs dans $\mathbb{Z}_K$ est la même que l'intuition qu'on peut avoir d'une horloge à valeurs dans $\mathbb{Z}$.

Dans la section 6 nous proposons un algorithme de calcul d'un arbre couvrant en largeur d'abord. Cet algorithme est optimal en occupation mémoire et en temps de construction. Il est le seul, à notre connaissance, qui ait ces deux propriétés d'optimalité. Une version de cet algorithme est présentée dans le modèle à passage de messages, avec l'hypothèse que les communications sont de type FIFO. Cette section présente un double intérêt :

1. elle montre que le vocabulaire introduit dans ce chapitre rend très simple et intuitive l'étude de cet algorithme ;
2. le lemme 34, qui est essentiel à la preuve de correction de l'algorithme, est en fait un lemme très général, il est la clé qui permet de réécrire tout le chapitre dans le modèle à passage de messages.

5.1.2 Chapitre 3

Dans ce chapitre nous proposons une famille d'unissons très générale à trois paramètres dans un réseau anonyme. Cette famille est basée sur un système de réinitialisation locale. Une analyse complète de complexité est effectuée. Un cas très particulier est l'unisson de Couvreur, Gouda et Francez [CFG92] ; notre travail permet de donner une preuve complète de correction de cet unisson, ainsi qu'une analyse de complexité en temps. En particulier, [CFG92] converge en $O(nd)$ en pire cas, tandis que les meilleurs protocoles de cette famille convergent en $O(n)$ en pire cas (rappelons que n est le nombre de processus du réseau et d le diamètre de celui-ci).
Deux notions jouent un rôle important :

1. la notion de *Dag de réinitialisation* ;
2. la notion de *trou* dans un graphe.

5.1.3 Annexe A

Dans cette annexe, et suivant [AG94], nous proposons un unisson original, muni d'une procédure de réinitialisation globale. C'est un algorithme efficace d'unisson sur un réseau avec identités et sous un démon faiblement équitable. Cet unisson converge en $O(d)$ rondes. Cet algorithme est basé sur une réinitialisation globale utilisant un arbre couvrant enraciné. Il utilise le fait que si $K > C_G$, la désynchronisation est localement contrôlable. Une fois qu'une telle désynchronisation est localement repérée par un processus, ce processus envoie à la racine un ordre de réinitialisation globale. La racine gère alors cette tâche de manière classique. C'est le caractère classique de la méthode qui nous a poussé à proposer cet algorithme seulement en annexe. Cet algorithme n'est pas publié.

5.1.4 Annexe B

Nous y montrons que, dans le cas général, trouver la longueur d'un plus grand trou dans un graphe est un problème NP-difficile.

5.2 Deuxième partie : optimiser

Cette partie se compose de deux chapitres, les chapitres 4 et 5.

5.2.1 Chapitre 4

Dans ce chapitre nous abordons l'unisson synchrone :

1. nous montrons que si $K > C_G$, alors, dans un réseau synchrone, un protocole d'unisson asynchrone converge vers un unisson synchrone.
2. On en déduit que l'unisson asynchrone $GAU(K, 0, 1)$ (voir programme 7 page 96) fournit un unisson synchrone sur l'arbre qui est optimal en espace. Cet unisson fonctionne avec n'importe quelle taille d'horloge $K > 2$ et avec un nombre d'états $S = K$. Il converge en au plus $2d$ pulsations. Cette solution donne une réponse positive à une question posée par Nolot et Villain [Nol02] : *"existe-t-il un unisson synchrone sur l'arbre dont le nombre d'états est indépendant de toute information locale ou globale sur l'arbre, c'est-à-dire le nombre de processus, le diamètre, ou le degré du réseau ?"*. On peut remarquer aussi que pour le cas $K = 3$, le temps de convergence est majoré par d pulsations, cette instance du protocole précédent définit un protocole qui atteint les performances de [HG95] ; curieusement, ce nouveau protocole est différent de [HG95], il est plus *uniforme* dans sa définition et son exécution.
3. Le troisième résultat de ce chapitre est une nouvelle solution générale qui utilise les avantages des deux approches proposées dans [ADG91] et dans [BPV04]. Ce protocole, appelé SS_MinSU, demande $K \geq 2$ seulement, avec $S \geq K + d$.

Son temps de convergence est majoré par $2d$ seulement. C'est la meilleure solution actuelle pour un réseau quelconque synchrone, aussi bien en temps de convergence qu'en occupation mémoire.

Ce chapitre a été présenté à la conférence *SSS 2004* [BPV05] et fait l'objet d'une publication dans la revue *Algorithmica* (à paraître, tapuscript accepté).

5.2.2 Chapitre 5

Ce chapitre est un petit peu plus technique. Nous y construisons un unisson asynchrone et un unisson synchrone qui convergent sur l'arbre en moins de d rondes (resp. pulsations) . Ces deux protocoles s'écrivent avec une seule règle, et l'ordre de l'horloge est un entier impair quelconque supérieur à 2. Les preuves de convergence utilisent de manière fine les outils développés précédemment. Ce chapitre a fait l'objet d'une conférence à *SSS 2006* [BPV06].

5.3 Troisième partie : synchroniser

Cette partie comprend trois chapitres : les chapitres 6, 7 et 8.
L'exclusion mutuelle locale est à l'origine de l'ensemble de ce travail. Le problème est d'assurer que deux processus voisins n'entrent pas en section critique simultanément, et que pour toute exécution maximale, tous les processus entrent en section critique une infinité de fois. Donner une solution à ce problème permet par exemple de construire un transformateur permettant de passer d'un démon central à un démon distribué. Si l'exclusion mutuelle locale est développée dans le modèle à registres [NA02, HG05], elle permet de fabriquer un transformateur permettant de passer du modèle à états au modèle à registres, tout en raffinant éventuellement le démon (passage d'un démon faiblement équitable à un démon inéquitable).

5.3.1 Chapitre 6

Le chapitre 6 propose un algorithme d'exclusion mutuelle locale auto-stabilisant dans le modèle à états, et sur le graphe général. Il utilise l'unisson. En fait, ce chapitre aborde le problème un peu plus général de l'allocation locale de ressources , et propose aussi une version silencieuse de ces algorithmes. L'essentiel de ce chapitre a été présenté à la conférence *PODC 2004* [BPV04].

5.3.2 Chapitre 7

Le chapitre 7 montre qu'on peut voir l'unisson comme un courant de vagues sur un réseau, ce réseau pouvant être anonyme. Nous introduisons deux généralisations de la notion de vague : les $\rho-$vaguelettes et les vagues fortes. Nous montrons comment ces deux notions permettent de calculer respectivement un infimum à distance ρ et un problème global paramétré par des $r-$opérateurs.

Nous montrons comment l'unisson permet de définir une barrière de synchronisation autostabilisante à distance ρ, et comment la structure de ses communications permet de définir des vaguelettes, des vagues et des vagues fortes.

5.3.3 Chapitre 8

Le chapitre 8 utilise les vaguelettes pour faire de la synchronisation à distance ρ. En particulier, cet outil permet de résoudre le problème de l'exclusion mutuelle locale à distance ρ, et en particulier de simuler un $\rho-$démon central. De ce travail on peut déduire, par exemple, un transformateur qui permet de transformer un protocole défini dans le modèle à réécriture de graphe à distance k [GMM00] en un protocole défini dans le modèle à états.

5.4 Conclusion et annexes

Les deux dernières parties concernent la conclusion et les annexes. Les trois premières annexes ont déjà été présentées, la dernière expose des notions élémentaires d'algèbre, nécessaires à la compréhension du texte principal. On y trouvera aussi quelques rappels de théorie des graphes, les autres notions utiles de théorie des graphes sont introduites dans le corps du texte. A la fin de ce travail, un index propose les renvois aux définitions.

Première partie

Stabiliser

Chapitre 2

Comprendre

...de conduire par ordre mes pensées, en commençant par les plus simples et les plus aisées à connaître...

Descartes, *Discours de la méthode* [Des37]

Sommaire

1 L'unisson sur les entiers relatifs

La question de l'unisson est une question très simple. Il s'agit d'organiser sur un réseau réparti, une $\alpha-$synchronisation avec une compatibilité de phase entre chaque couple de processus voisins. Pour ce faire, chaque processus p maintient un registre $p.r$ à valeurs dans un ensemble χ (ensemble des numéros de phase). L'ensemble χ est muni d'une incrémentation φ. Un exemple simple d'un tel système d'incrémentation est l'ensemble des entiers relatifs $\mathbb{Z}$ muni de la fonction successeur. Un autre exemple est l'ensemble des entiers modulaires $\mathbb{Z}_K$ (noté aussi $\mathbb{Z}/K\mathbb{Z}$) muni de l'incrémentation $x \to \overline{x+1} \bmod K$. Nous verrons d'autres exemples plus loin.

Le premier exemple pose un problème pour l'informaticien, car l'ensemble des états de l'horloge n'est pas fini. Le deuxième pose un problème pour le mathématicien, car il n'a pas de bonnes propriétés d'ordre. Nous allons voir que l'on peut dépasser ces problèmes grâce à quelques concepts bien choisis. Mais pour cela, il nous faut d'abord bien comprendre les difficultés. Nous proposons de développer le cas de $\mathbb{Z}$ muni de la fonction successeur, notée encore φ. Nous allons voir que ce cas particulier simple et intuitif correspond à un comportement "universel", en un certain sens que nous préciserons plus tard dans ce chapitre.

1.1 Une définition dans le cas d'une horloge à valeurs dans $\mathbb{Z}$.

Donnons une définition provisoire de l'unisson dans le cas d'une horloge à valeurs dans $\mathbb{Z}$: chaque processus p maintient un registre $p.r$ à valeurs dans $\mathbb{Z}$ muni de la fonction successeur φ. Soit Γ l'ensemble des états γ du système, c'est-à-dire la séquence des valeurs des registres r de chacun des processus, dans un certain ordre fixé. Nous définissons sur l'ensemble des états du système le prédicat :

$$\Gamma_1(\gamma) \equiv_{def} \forall p \in V, \forall q \in \mathcal{N}_p : |q.r - p.r| \leq 1$$

Nous confondrons Γ_1 avec l'ensemble des états γ qui satisfont le prédicat Γ_1.
Le problème de l'unisson asynchrone est de définir un protocole qui satisfasse, pour toute exécution, les propriétés suivantes :

Spécification 3 [Unisson : spécification provisoire pour $\chi = \mathbb{Z}$]

1. **sûreté** :
   (synchronisation) un processus p peut incrémenter son horloge $p.r$ seulement si la valeur de $p.r$ est inférieure ou égale aux valeurs des horloges des processus voisins de p ;
2. **vivacité** :
   (a) (absence de famine) tout processus p incrémente son horloge une infinité de fois ;
   (b) (convergence) $\Gamma \triangleright \Gamma_1$, c'est-à-dire qu'à partir de n'importe quel état du système, au bout d'un nombre fini d'étapes, le système est dans un état de Γ_1.

La condition de sûreté garantit que l'ensemble Γ_1 est clos. Pour prouver cela, supposons qu'un protocole satisfasse les spécifications ci-dessus, alors la proposition suivante est vraie :

Proposition 2 Si $\gamma \in \Gamma_1$ et $\gamma \rightarrow \gamma'$, alors $\gamma' \in \Gamma_1$

Preuve : Supposons que $\gamma' \notin \Gamma_1$ alors il existe deux processus voisins p et q, tels que $|p.r - q.r| > 1$. Quitte à échanger les rôles de p et q, supposons que $q.r > p.r + 1$. Alors nécessairement dans l'état γ : $q.r \geq p.r + 1$ et donc le processus p ne peut avoir exécuté l'action à cause de la condition de sûreté. Cela conduit à une contradiction. On en déduit la clôture de Γ_1.

□

Nous donnerons une spécification définitive de l'unisson dans le chapitre 3, section 1.6.
Comme annoncé, nous allons commencer par développer l'exemple dans lequel les registres sont à valeurs dans $\mathbb{Z}$. Dans ce cas, un algorithme très simple s'impose, l'idée est de faire une preuve complète et détaillée de la correction de cet algorithme, afin de dégager précisément les raisons de cette correction, et ainsi de se convaincre que, déjà dans ce cas simple, il est nécessaire d'utiliser des outils fins, même s'ils sont bien connus donc souvent implicites,

donc parfois invisibles. Dans le cas de $\mathbb{Z}$, les raisons de la correction de l'algorithme que nous proposons sont essentiellement l'ordre total sur $\mathbb{Z}$ et le caractère monogène (pour l'incrémentation) de l'ensemble des entiers relatifs. Nous étudierons ensuite comment il est possible de conserver de telles méthodes dans le cas de registres bornés, où un ordre total sur les registres est a priori perdu.

1.2 Définition d'un protocole.

L'algorithme, pour chaque processus, est défini par une seule instruction gardée notée (NA) :

$$NA : \forall q \in N_p \; p.r \leq q.r \rightarrow p.r := p.r + 1$$

Algorithme 1 Unisson avec une horloge non bornée définie sur $\mathbb{Z}$.

Constante et variable :

$\mathcal{N}_p$: l'ensemble des voisins de p ;

$p.r \in \mathbb{Z}$;

Action :

NA : $\quad (\forall q \in \mathcal{N}_p \; p.r \leq q.r) \quad \longrightarrow \quad p.r := p.r + 1$;

Soit $\gamma_0\gamma_1$.... une exécution asynchrone du protocole réparti défini par l'algorithme 1. L'état γ_0 de démarrage étant un état quelconque de Γ. La correction de l'algorithme revient à prouver :

que la condition de synchronisation est toujours satisfaite ;

qu'il y a l'absence de famine ;

et que la convergence vers Γ_1 est assurée.

Pour prouver la correction de l'algorithme, nous disposons d'un outil simple : la relation d'ordre total sur l'ensemble des entiers naturels $\mathbb{Z}$. Nous allons prouver les trois propriétés en insistant bien, lorsqu'il y a lieu, sur le rôle de la relation d'ordre.

1.3 Correction du protocole

1.3.1 Sûreté et clôture de Γ_1.

La condition de sûreté est garantie par la garde de l'action (NA). Il suit par la proposition 2 que Γ_1 est clos.

1.3.2 Vivacité

Pour montrer qu'il n'y a pas de famine, nous allons utiliser l'ordre. Pour cela, nous construisons un outil qui sera utile pour la suite. Reprenons l'exécution $\gamma_0\gamma_1$..... A la date t, donc dans l'état γ_t, notons $\perp_t$ la valeur la plus petite des registres $p.r$, et notons $\top_t$ la plus

grande de ces valeurs.

Lemme 3 Pour tout n entier naturel, au bout de n rondes on a : $\perp_t \geq \perp_0 + n$

Preuve : On fait une récurrence sur le nombre de rondes.
Si $n = 0$ le lemme est clair.
Supposons que le lemme soit vrai après la ronde n. Donc à la fin de la ronde n, $\perp_t \geq \perp_0 + n$. Les processus dont les registres ont la valeur minimale peuvent exécuter l'action (NA), donc font au moins une fois l'action (NA) durant la ronde suivante, donc à la fin de la ronde $n+1$: $\perp_{t'} \geq \perp_0 + n + 1$.

□

De ce lemme, on déduit immédiatement que chaque processus exécutera une infinité de fois l'acion (NA), ce qui prouve la vivacité.

Proposition 4 L'algorithme 1 est sans famine.

1.3.3 Convergence vers Γ_1

Posons Λ_t l'ensemble des processus dont la valeur du registre r est supérieure strictement à $\top_0$ dans l'état γ_t.

Lemme 5 Pour tout $t \in \mathbb{N}$, pour tout $p \in \Lambda_t$ et tout $q \in \mathcal{N}_p$: $|q.r - p.r| \leq 1$ dans l'état γ_t, on dit que p est correcte avec q.

Preuve : Nous prouvons ce lemme par récurrence sur t.

Pour $t = 0$, la proposition est vraie car $\Lambda_0 = \emptyset$.

Supposons que la proposition soit vraie pour $t \in \mathbb{N}$. Supposons que le processus p incrémente son registre dans la transition $\gamma_t \rightarrow \gamma_{t+1}$ de sorte : que $p \in \Lambda_{t+1}$. Deux cas sont possibles :

- si $p \in \Lambda_t$ alors p est correct avec tous ses voisins par hypothèse de récurrence. Le lemme suit de la définition de l'action (NA).
- Si $p \notin \Lambda_t$ alors à la date t, p peut exécuter l'action (NA) et $p.r = \top_0$. Donc, pour tout $q \in \mathcal{N}_p$, $q.r \geq p.r$ et :
  - si $q.r = p.r$ alors à la date $t+1$ on a $|q.r - p.r| \leq 1$;
  - si $q.r > p.r$ alors $q \in \Lambda_t$. Donc le processus q est dans Λ_t, donc par hypothèse de récurrence : $q.r = p.r + 1$. Par conséquent, le processus q ne peut effectuer l'action (NA) et donc dans l'état suivant γ_{t+1}, on a : $|q.r - p.r| = 0$.

On en déduit le lemme.

□

On déduit des lemmes 3 et 5 la proposition suivante :

Proposition 6 Au bout d'au plus $\top_0 - \perp_0$ rondes, le système a convergé dans Γ_1.

On obtient comme corollaire le théorème attendu :

Théorème 7 Le protocole 1 défini par l'action (NA) résout le problème de l'unisson asynchrone.

2 Problèmes à résoudre dans le cas d'une incrémentation bornée

La résolution du problème de l'unisson est donc simple si les registres sont à valeurs dans $\mathbb{Z}$. La correction provient essentiellement de l'ordre total sur $\mathbb{Z}$ et du caractère monogène de $\mathbb{Z}$. Le projet est maintenant de faire la même chose mais avec des registres bornés. Plusieurs questions se posent :

1. Qu'est-ce qu'une incrémentation bornée ?
2. Dans un système fini, l'incrémentation est nécessairement ultimement périodique. Il n'existe donc pas d'ordre total compatible avec cette incrémentation. La question est alors : comment faire des comparaisons locales entre processus voisins dans ces conditions ?
3. Dans une situation où tout couple de processus voisins est un couple de processus comparables, peut-on garantir l'absence d'interblocage ? Remarquons que dans le cas de $\mathbb{Z}$, l'absence d'interblocage est prouvée en utilisant l'ordre total sur $\mathbb{Z}$.
4. L'utilisation de l'ordre total est essentielle pour prouver la convergence dans le cas de $\mathbb{Z}$. Peut-on redéfinir un ordre global une fois un ordre local défini ?
5. Comme il n'y a pas d'ordre global compatible avec l'incrémentation, il existera donc des configurations où deux processus voisins ne seront pas comparables ; que faire de ce genre de situation nouvelle.

Nous allons donner successivement des réponses à chacune des quatre premières questions. Nous allons définir ce qu'est un système à incrémentation bornée et définir un ordre local sur celui-ci. Nous considérerons alors, sur l'ensemble des états Γ, les états où deux processus voisins quelconques sont comparables. Ces états constituent un sous-ensemble de Γ qui sera noté Γ_M. On montrera alors que dans Γ_M, bien que les registres de processus voisins soient toujours comparables, il peut y avoir encore des situations inadmissibles : des interblocages, des ralentissements (exécutions en exclusion mutuelle globale par exemple) et bien d'autres phénomènes. Nous donnerons une condition suffisante sur la taille des horloges afin que ces pathologies disparaissent. Sous cette condition, nous définirons deux outils importants, le *retard* et le *relèvement*, qui nous permettront de prouver la convergence $\Gamma_M \triangleright \Gamma_1$. La réponse à la cinquième question, et la convergence générale $\Gamma \triangleright \Gamma_1$ seront étudiées dans le chapitre 3, qui porte sur la tolérance aux fautes. Le chapitre 3 utilisera de manière importante les outils développés dans le présent chapitre.

3 Incrémentation bornée et ordre local

3.1 Etude générale

En s'inspirant de la théorie des modèles [CH73], définissons un langage composé par une constante $\overline{0}$ et une fonction unaire φ, appelée incrémentation. Il y a de nombreux modèles $\left(\chi, \overline{0}, \varphi\right)$ correspondant à ce langage :
$(\mathbb{N}, 0, x \to x+1)$, $(\mathbb{Z}, 0, x \to x+1)$ ou encore $\left(\mathbb{Z}/K\mathbb{Z}, 0, x \to \overline{x+1}\,[K]\right)$ sont des exemples, mais il y en a beaucoup d'autres. Nous nous limiterons dans la suite à ceux qui vérifient les trois conditions supplémentaires suivantes :

1. χ est fini.
2. Pour tout $u \in \chi$, il existe un entier n tel que $\varphi^n(u) = \bar{0}$.
3. Il existe $v \in \chi$ tel que $\chi = \{\varphi^n(v), n \in \mathbb{N}\}$.

La deuxième condition impose que toute itération passe par la constante $\overline{0}$; cela impose la connexité du système d'incrémentation. La dernière condition impose que le système soit monogène (Il existe une valeur dont l'ensemble des itérés par φ est l'ensemble χ). Si on oublie la dernière condition, on peut construire des modèles très curieux, comme le montre la figure 2.1. Ces modèles sont peut-être utiles mais nous n'en savons rien.

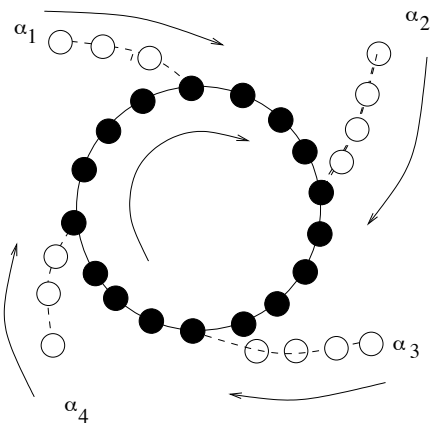


FIG. 2.1 – Une incrémentation exotique $(\mathcal{X}, \varphi)$.

Nous appellerons *incrémentation bornée* tout modèle satisfaisant les trois conditions supplémentaires. Dans ce cas, la forme générale est facile à définir par une simple analyse :
en fait, on a alors $\chi = \left\{\varphi^k(v), k \in \mathbb{N}\right\} = \left\{\varphi^k(v), k \in \{0, .., M-1\}\right\}$ où M est le cardinal de χ, et l'application :

$$\begin{array}{rcl} \{0, .., M-1\} & \to & \chi \\ k & \to & \varphi^k(v) \end{array}$$

est bijective.

Posons $\bar{0} = \varphi^M(\upsilon)$, il existe un unique $\alpha \in \{0, .., M-1\}$ tel que $\bar{0} = \varphi^\alpha(\upsilon)$.
Cela nous permet un système général de codage d'une incrémentation :
posons $K = M - \alpha$, prenons $\chi = \{-\alpha, ..., 0, ..., K-1\}$, on a $\bar{0} = 0$. φ est alors définie par :

$$\begin{array}{rcl} x & \rightarrow & x+1 \text{ si } x \neq K-1 \\ K-1 & \rightarrow & 0 \end{array}$$

Dans la suite, nous appellerons *représentation modulaire* ce modèle d'incrémentation.
Nous le représenterons comme une cerise , voir la figure 2.2.

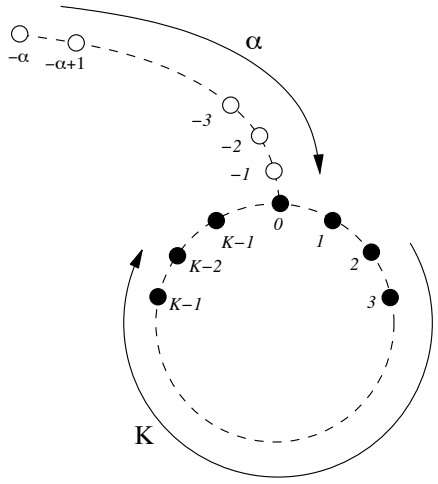


FIG. 2.2 – Système d'incrémentation fini $(\mathcal{X}, \varphi)$.

Introduisons un peu de vocabulaire.

Définition 13
On appelle $-\alpha$ l'élément initial de (χ, φ).
On appelle période du système l'entier : K.
On note $tail_\varphi = \{-\alpha, ..., 0\}$, c'est l'ensemble des valeurs de réinitialisation (y compris $0 = \varphi^\alpha(-\alpha)$)
On pose $ring_\varphi = \{0, ..., K-1\}$, φ est bijective sur $ring_\varphi$.
On note $tail^*_\varphi = \emptyset$ si $\alpha = 0$, sinon $tail^*_\varphi = \{-\alpha, ..., -1\}$
Le couple (α, K) caractérise l'incrémentation, on l'appelle la signature de l'incrémentation.
Nous disposons d'un ordre total naturel sur χ et donc sur $tail_\varphi$. Nous noterons cette relation d'ordre par $\leq$ sur χ et, sur $tail_\varphi$, par $\leq_{tail}$.

La fonction φ n'est pas compatible avec l'ordre total naturel sur χ. Cette remarque fait l'objet du paragraphe suivant.

3.2 Ordre local sur un système à incrémentation bornée

Rappelons rapidement quelques notions élémentaires sur les congruences. Soit $\mathbb{Z}$ l'ensemble des entiers relatifs. Soit K un entier strictement positif.

1. On dit que a et b sont congruents modulo K et on écrit $a \equiv b\,[K]$, si et seulement si il existe λ dans $\mathbb{Z}$ tel que $b = a + \lambda K$.
2. On note $\overline{a}$ l'unique entier dans $[0, K[$ tel que $a \equiv \bar{a}\,[K]$.
3. $d_K(a,b) = \inf(\overline{a-b}, \overline{b-a})$ est une *distance* sur $[0, K-1]$.

Définition 14 Soit M un entier positif ou nul tel que $2M < K$. Deux entiers a et b sont dits *M-localement comparables* (ou simplement *localement comparables*) si et seulement si $d_K(a,b) \leq M$. On définit la relation d'*ordre local* $\leq_l$ de la manière suivante :

$$a \leq_l b \overset{\text{def}}{\Leftrightarrow} 0 \leq \overline{b-a} \leq M$$

Le cas $M = 0$ peut sembler artificiel, car dans ce cas le préordre est l'égalité. Nous l'introduisons par souci d'uniformité, il sera utile lorqsque nous étudierons les barrières de synchronisation forte (synchrones). Remarquons que si $M > 0$ alors $\leq_l$ définit un préordre qui n'est pas l'égalité ni l'identité de tous les objets. Dans la suite nous supposons toujours $K > 2$. Remarquons que dans le cas où $K = 2M + 1$, alors K est impair et tout couple d'entiers est un couple d'entiers M-localement comparables. Dans le cas de systèmes asynchrones, nous supposerons toujours $M > 0$. $M = 0$ a un intérêt seulement dans le cas synchrone (voir chapitre 4).

Définition 15 Etant donné deux entiers localement comparables a et b, on définit la *différence locale* $b \ominus_l a$ de la manière suivante :

$$b \ominus_l a \overset{\text{def}}{=} \begin{cases} \overline{b-a} & \text{si } a \leq_l b \\ -\overline{a-b} & \text{sinon (et alors } b \leq_l a) \end{cases}$$

Quand il n'y aura pas d'ambiguité, nous omettrons l'indice "l" dans $b \ominus_l a$.

Il est important de remarquer que $b \ominus_l a \equiv b - a[K]$.

Définition 16 si $a_0, a_1, a_2, \ldots a_{p-1}, a_p$ est une séquence d'entiers tels que $\forall i \in \{0, \ldots, p-1\}$, a_i est localement comparable à a_{i+1}, alors on définit la *variation locale* S de cette séquence de la manière suivante : $S = \sum_{i=0}^{p-1} (a_{i+1} \ominus_l a_i)$.

Clairement, $S \equiv a_p - a_0[K]$, et $S \equiv 0[K] \Leftrightarrow a_p - a_0 \equiv 0[K]$.

Récapitulons les différentes notions d'ordre sur la cerise (voir définition page 56) :

1. $\leq_{tail}$ est l'ordre naturel sur $tail_\varphi$,
2. $\leq$ est l'ordre naturel sur $\mathcal{X}$,
3. $\leq_l$ est l'ordre local sur $ring_\varphi$.

4 Protocole d'incrémentation de phases et pathologies

4.1 Graphes à nœnds étiquetés

Maintenant que nous avons défini ce qu'est un système à incrémentation bornée, étudions les configurations γ possibles.

> On se donne donc une incrémentation (χ, φ) de signature (α, K) avec $K > 2$.
> On définit un *ordre local* en choisissant un entier M satisfaisant les inégalités : $M \geq 0$ et $2M < K.$; on peut par exemple prendre $M = 1$, puisque $K > 2$.

Chaque processus p est donc muni d'un registre $p.r$, c'est l'étiquette du nœnd p. Chaque configuration apparaît comme un graphe à nœnds étiquetés, les étiquettes sur les nœnds étant à valeurs dans χ. Définissons formellement la notion de *graphe à nœnds étiquetés*, ou plus simplement *graphe étiqueté*.

> **Définition 17** [Graphe à nœnds étiquetés] Soit $\mathcal{X}$ un ensemble. Un $\mathcal{X}$-*graphe étiqueté* est un couple (G, r) où $G = (V, E)$ est un graphe, et r est une application de V vers $\mathcal{X}$.

Pour être cohérent avec le contexte dans lequel les graphes étiquetés interviennent, à chaque nœnd p du graphe, nous noterons la valeur de son étiquette $p.r$. Nous confondrons la notion de configuration avec celle de graphe étiqueté, ainsi chaque graphe étiqueté est une configuration γ du système. On note enfin Γ l'ensemble des configurations du système (ensemble des graphes à nœuds étiquetés). Nous définissons maintenant quelques notions globales sur les graphes à nœuds étiquetés.

> **Définition 18** [Correction locale]
> Un graphe étiqueté (G, r) est M-localement correct si et seulement si les étiquettes de deux nœnds adjacents quelconques de G sont M -localement comparables.
> On note Γ_M le sous-ensemble de Γ des configurations (graphes étiquetés) γ qui sont M-localement corrects. Γ_0 est le sous-ensemble de Γ_M des configurations à l' unisson, c'est-à-dire celles où toutes les étiquettes sont identiques, et à valeurs dans $ring_\varphi$.

On peut remarquer que dans le cas où $\chi = \mathbb{Z}$, les processus sont toujours comparables ; on peut alors considérer M comme égal à $+\infty$ et dans ce cas : $\Gamma = \Gamma_\infty$.

4.2 Synchronisation de phase avec horloges à valeurs dans un système à incrémentation bornée

En général, on a seulement $\Gamma_M \subset \Gamma$ et pas d'égalité, car deux éléments de χ peuvent ne pas être comparables. Il existe des situations où il y a égalité : par exemple, si on prend $M = 1$, $K = 3$ et $\alpha = 0$, alors $\Gamma_1 = \Gamma$. On trouvera un exemple important d'une telle situation dans le chapitre 5.
On peut se demander s'il n'est pas possible de faire fonctionner un analogue de l'algorithme 1 en partant dans Γ_M.

4.2.1 Protocole d'incrémentation de phase

Dans cette section, on définit par l'algorithme 2 un nouveau protocole, appelé protocole de synchronisation de phase. Cet algorithme est défini par une seule action gardée notée NA :

$$(NA) : \forall q \in N_p,\ p.r \leq_l q.r \rightarrow p.r := \varphi\,(p.r)$$

Algorithme 2 Protocole d'incrémentation de phase du processus p

Constante :
$\mathcal{N}_p$: l'ensemble des voisins de p ;
Variable :
$p.r \in \mathcal{X}$;
Fonctions booléennes :
$NormalStep_p \equiv \forall q \in \mathcal{N}_p : p.r \leq_l q.r$;
Action :
$NA : \quad NormalStep_p \longrightarrow p.r := \varphi(p.r)$;

Proposition 8 Si $0 < m \leq M$, alors l'ensemble Γ_m est clos pour l'algorithme 2.

Preuve : Soit $\gamma \rightarrow \gamma'$ une transition avec $\gamma \in \Gamma_m$. Supposons que p et q soient deux processus voisins. Quatre cas sont possibles :

1. Si p et q font l'action (NA) alors $p.r = q.r$ dans γ et γ', on a donc $d_K(p.r, q.r) = 0$ et ce qui est inferieur ou égal à m.
2. Si p seul fait l'action (NA) alors $p.r \leq_l q.r$ dans γ et alors $d\,(p.r, q.r) < m$ dans γ', on a donc $d_K(p.r, q.r) \leq m$.
3. Si q seul fait l'action (NA), on retrouve le cas précédent en échangeant les rôles de p et q, ainsi dans $\gamma' : d_K(p.r, q.r) \leq m$.
4. Si p et q ne font pas d'action, on a encore $d_K(p.r, q.r) \leq m$.

□

Dans tout ce qui suit, on supposera que ce protocole démarre dans Γ_M.

4.2.2 Pathologies

En général, ce protocole, même en démarrant dans Γ_M, présente de graves pathologies. On peut en particulier avoir des interblocages. Prenons par exemple un anneau de longueur $n > 2$, noté par la suite des processus $p_0 p_1 ... p_{n-1}$. Prenons $K = n$ et posons pour tout $i \in \{0, ..., n-1\}$ $p_i.r = i$: l'anneau est clairement interbloqué, voir figure 2.3 pour le cas $n = 5$.

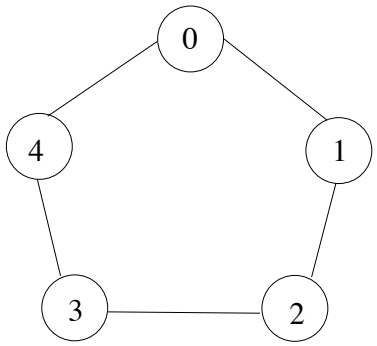


FIG. 2.3 – Un interblocage sur un anneau pour $K = 5$.

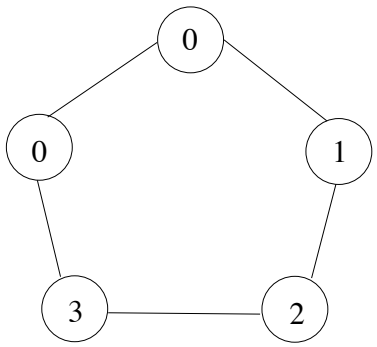


FIG. 2.4 – Exclusion mutuelle pour $K = 4$.

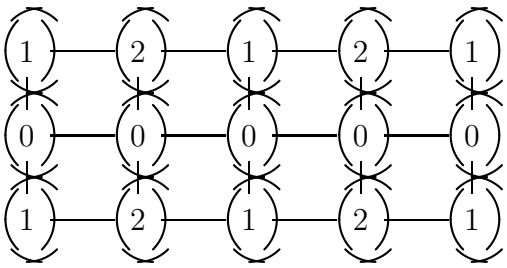


FIG. 2.5 – Une grille à mailles carrées de 15 processus, avec $K = 3$

On peut aussi avoir de curieux ralentissements. Sur la figure 2.4 on peut observer qu'en toute circonstance, un seul processus peut faire l'action (NA). Le fonctionnement est donc en exclusion mutuelle.
Prenons un autre exemple : soit une grille à mailles carrées de 15 processus sur trois rangées, voir figure 2.5. On prend $K = 3$. Ici, l'exécution se fait en exclusion mutuelle sur chacun des carrés de base de quatre processus. Une exécution synchrone est donnée sur la figure 2.6, les processus activables y sont doublement entourés.

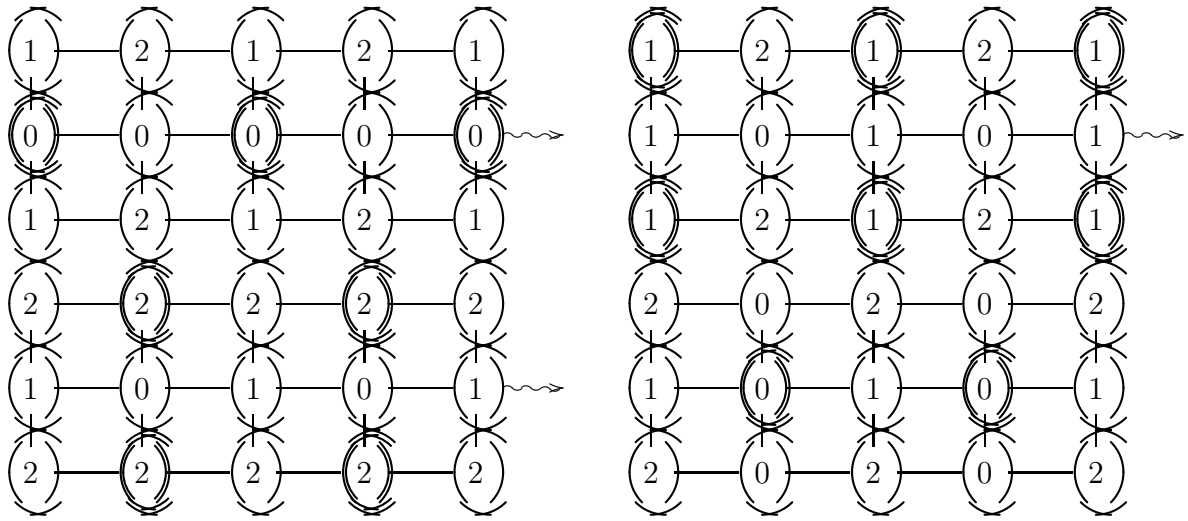


FIG. 2.6 – Une exécution synchrone en exclusion mutuelle sur chaque cellule d'une grille, pour $K = 3$

4.2.3 Interblocages

Introduisons la notion de graphe d'interblocage.

Définition 19 Soit $\gamma \in \Gamma_M$, le graphe d'interblocage associé est le graphe défini sur l'ensemble V des processus par la relation :

$$p \to q \Leftrightarrow q \in \mathcal{N}_p \text{ et } q.r = \varphi(p.r) \tag{2.1}$$

Ce graphe associé à l'état γ est noté $\mathcal{L}(\gamma)$.

Par exemple, le graphe d'interblocage de la première configuration de la figure 2.6 est donné par la figure 2.7.
Remarquons que $\mathcal{L}(\gamma) = \mathcal{L}(\gamma')$ si et seulement si il existe une translation τ telle que $\tau(\gamma) = \gamma'$. On note $\mathcal{L}(0)$ le graphe d'interblocage sans arête ; il correspond à un état où tous les registres r des processus sont à la même valeur.

Proposition 9 Si le graphe d'interblocage d'une configuration contient un circuit, alors ce circuit est invariant et l'exécution de l'algorithme 2 conduit à un interblocage.

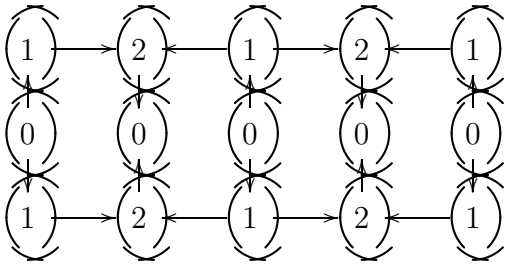


FIG. 2.7 – Graphe d'interblocage de la grille à mailles carrées de la figure 2.6.

Proposition 10 Si on met un processus en famine (il ne fait plus aucune action), alors l'exécution du protocole 2 conduit à un interblocage de l'ensemble du réseau. Une fois l'interblocage réalisé, deux situations sont possibles :

1. le processus est sur un circuit d'interblocage, donc il ne pouvait de toute façon pas exécuter l'action (NA).
2. le processus est minimal pour la relation de précédence et peut donc exécuter l'action d'incrémentation (NA).

Proposition 11 Si l'algorithme 2 est sans interblocage alors il est sans famine.

Preuve : c'est la contraposée de la proposition précédente.

□

Proposition 12 Dans le cas où une configuration γ est dans Γ_1, alors elle conduit à un interblocage si et seulement si le graphe d'interblocage $\mathcal{L}(\gamma)$ de la configuration contient un circuit.

Par exemple, la configuration de la figure 2.6 est dans Γ_1 et a un graphe d'interblocage (voir figure 2.7) qui ne contient pas de circuit, donc l'algorithme sera vivace à partir de cette configuration.

Dans l'exemple de la figure 2.8, $M = 2$. On présente une configuration dont le graphe d'interblocage ne contient aucun circuit et qui pourtant conduit à un interblocage.

5 Retard et relèvement

On voit sur le dernier exemple de la section précédente que l'étude de l'interblocage en général ne conduit pas à des résultats satisfaisants. On ne peut donc pas rester à ce niveau. Il nous faut faire un effort d'abstraction pour mieux comprendre les phénomènes et espérer obtenir une théorie satisfaisante. Dans cette perspective, nous introduisons les notions de *retard* et de *relèvement*.

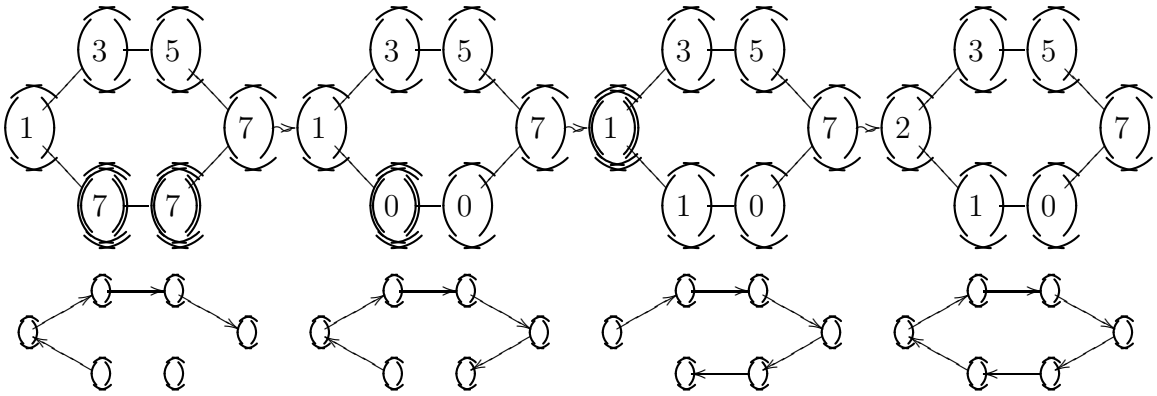


FIG. 2.8 – Exemple d'une exécution conduisant à un interblocage. $M = 2$ et $K = 8$.

5.1 La notion de retard

Dans cette partie nous allons développer la notion de *retard* suivant un chemin, qui va se révéler être un outil très puissant. Le retard suivant un chemin peut s'interpréter comme l'image par une forme linéaire d'un certain vecteur dans un certain espace vectoriel. Cette interprétation va nous permettre de comprendre à quelle condition un retard peut être intrinsèque entre deux processus ; c'est-à-dire être indépendant du chemin suivant lequel on calcule ce retard. Commençons par définir la notion de retard.
Dans la suite de cette partie, on suppose donné un graphe étiqueté localement correct, c'est-à-dire un élément (une configuration) γ de Γ_M.

5.1.1 Retard suivant un chemin

Définition 20 [Retard dans le réseau] Soit $\gamma \in \Gamma_M$. Le retard suivant un chemin $\mu = p_0, p_1, \ldots, p_k$ du graphe γ, noté $\Delta_\mu(\gamma)$, est la variation locale de $r_{p_0}, r_{p_1}, \ldots, r_{p_k}$, c'est-à-dire : $\Delta_\mu(\gamma) = \sum_{i=0}^{k-1} \left(r_{p_{i+1}} \ominus_l r_{p_i}\right)$ si $k > 0$, 0 sinon (i.e. $k = 0$). Nous noterons ce retard Δ_μ s'il n'y a pas de confusion sur l'état considéré.

La notion de retard suivant un chemin est une notion algébrique. Son comportement est comme une intégrale suivant un chemin. Clairement : $\Delta_\mu = -\Delta_{\tilde{\mu}}$, où $\tilde{\mu}$ désigne le chemin miroir de μ. Soit $\mu_1 = p_0, p_1, \ldots, p_j$ et $\mu_2 = p_j, \ldots, p_{k-1}$ deux chemins, la relation de Chasles : $\Delta_{\mu_1\mu_2} = \Delta_{\mu_1} + \Delta_{\mu_2}$ est satisfaite.

Définition 21 [Retard intrinsèque] Le retard entre deux nœnds p et q dans V est pq-intrinsèque s'il est indépendant du choix du chemin de p vers q. Le retard est intrinsèque si et seulement si pour tout $(p, q) \in V^2$ le retard entre p et q est pq-intrinsèque.
On note Γ_M^0 le sous-ensemble de Γ_M des configurations pour lesquelles le retard est intrinsèque.

5.1.2 Relation de précédence

Dans cette partie, on considère un état $\gamma \in \Gamma_M^0$.
Pour cet état, le retard est donc intrinsèque entre deux processus quelconques p et q du réseau. Notons Δ_{pq} la valeur commune des retards le long des différents chemins de p à q.

Définition 22 On dit que p *précède* q dans une configuration $\gamma \in \Gamma_M^0$ si et seulement si $\Delta_{pq} \leq 0$. De même on dit que p et q sont γ-*synchrones*, ou encore *synchrones*, si $\Delta_{pq} = 0$ dans γ. On note la relation de précédence par $\preceq_\gamma$, et la relation de γ-synchronisation par $\equiv_\gamma$.

Remarque

Comme le réseau est connexe, la relation de précédence est un préordre (voir définition page 228) et la relation de γ-synchronisation est la relation d'équivalence associée (voir corollaire 2.4 page 230). Les processus minimaux pour la relation de précédence sont synchrones. De même les processus maximaux pour la relation de précédence sont synchrones. L'ensemble des processus minimaux et l'ensemble des processus maximaux sont non vides, puisque le réseau est fini.

Proposition 13 La relation $\preceq_\gamma$ est une relation de préordre total sur l'ensemble V des processus.

Notons V_0 l'ensemble des processus minimaux. La relation de précédence induit sur le quotient de V par la relation d'équivalence $\equiv_\gamma$ un ordre total noté $\leq_\gamma$. Remarquons que $\frac{V}{\equiv_\gamma}$ est un singleton si et seulement si γ est à l'unisson, c'est-à-dire dans Γ_0. De plus, $\frac{V}{\equiv_\gamma}$ est un ensemble fini totalement ordonné par $\leq_\gamma$.

Soit $\gamma \in \Gamma_M^0$. Soit k le nombre de classes d'équivalence dans $\frac{V}{\equiv_\gamma}$. On représente ces classes de la manière suivante : $V_0, V_1, \ldots, V_{k-1}$ avec $V_0 <_\gamma V_1 <_\gamma \ldots <_\gamma V_{k-1}$.
Nous sommes ainsi capables de construire une application Σ de l'ensemble des configurations Γ_M^0 vers l'ensemble des partitions de Γ_M^0 par : $\Sigma(\gamma) = (V_0, V_1, \ldots, V_{k-1})$
Remarquons que $\Sigma(\gamma) = (V) \Leftrightarrow \gamma \in \Gamma_0$.
Nous venons de répondre positivement à la question 3 page 54 de ce chapitre, pourvu que l'état γ soit dans Γ_M^0. Une question importante maintenant est de savoir si l'ensemble Γ_M^0 est stable sous le protocole 2, car nous avons vu que cette stabilité garantit l'existence d'un préordre durant l'exécution du protocole, si celui-ci démarre dans Γ_M^0.

5.1.3 Retard et exécution du protocole

Une première question importante est de savoir comment le retard évolue durant l'exécution du protocole 2 cadencé par un démon inéquitable. Après avoir répondu à cette question, on peut espérer répondre à la question de savoir si Γ_M^0 est stable, et donc répndre positivement à la question posée dans la section précédente. La proposition suivante répond à la première question.

Proposition 14
Pour tout état $\gamma_i \in \Gamma_M$, pour toute transition $\gamma_i \rightarrow \gamma_{i+1}$ cadencée par un démon asynchrone, pour tout chemin $\mu = p_1 p_2 ... p_l$ on a l'égalité :

$$\triangle_\mu(\gamma_{i+1}) = \triangle_\mu(\gamma_i) + p_l^{i+1}.r \ominus p_l^i.r - p_1^{i+1}.r \ominus p_1^i.r$$

Preuve : La preuve se fait par récurrence sur la longueur du chemin.
Si la longueur du chemin est 0 alors $\Delta_\mu(\gamma_i) = \Delta_\mu(\gamma_{i+1}) = 0$ et $p_0^{i+1}.r \ominus p_0^i.r - p_0^{i+1}.r \ominus p_0^i.r = 0$, la proposition est donc vraie.
Supposons la proposition vraie au rang l, montrons-la au rang $l+1$. Soit $\mu = p_0 p_1 .... p_l p_{l+1}$ un chemin et $\mu_1 = p_0 p_1 .... p_l$; par l'hypothèse de récurrence :

$$\Delta_{\mu_1}(\gamma_{i+1}) = \Delta_{\mu_1}(\gamma_i) + p_l^{i+1}.r \ominus p_l^i.r - p_0^{i+1}.r \ominus p_0^i.r$$

et :

$$\begin{aligned} \Delta_\mu(\gamma_i) &= \Delta_{\mu_1}(\gamma_i) + p_{l+1}^i.r \ominus p_l^i.r \\ \Delta_\mu(\gamma_{i+1}) &= \Delta_{\mu_1}(\gamma_{i+1}) + p_{l+1}^{i+1}.r \ominus p_l^{i+1}.r \end{aligned}$$

donc :

$$\begin{aligned} \Delta_\mu(\gamma_{i+1}) &= \Delta_{\mu_1}(\gamma_i) + p_l^{i+1}.r \ominus p_l^i.r - p_0^{i+1}.r \ominus p_0^i.r + p_{l+1}^{i+1} \ominus p_l^{i+1} \\ &= \Delta_\mu(\gamma_i) - p_{l+1}^i.r \ominus p_l^i.r + p_l^{i+1}.r \ominus p_l^i.r - p_0^{i+1}.r \ominus p_0^i.r + p_{l+1}^{i+1}.r \ominus p_l^{i+1}.r \end{aligned}$$

Il ne reste plus qu'à montrer que :

$$p_{l+1}^{i+1}.r \ominus p_{l+1}^i.r = -p_{l+1}^i.r \ominus p_l^i.r + p_l^{i+1}.r \ominus p_l^i.r + p_{l+1}^{i+1}.r \ominus p_l^{i+1}.r$$

Posons $R = p_{l+1}^i.r \ominus p_l^i.r$, alors $R \in [-M, M]$.

1. Si $R > 0$ alors $p_{l+1}^i.r >_l p_l^i.r$ et seul p_l peut s'incrémenter : s'il ne fait rien, alors $p_l^i.r = p_l^{i+1}.r$ et $p_{l+1}^{i+1}.r = p_{l+1}^i.r$ et l'égalité est vérifiée.
   S'il s'incrémente, alors $p_l^{i+1}.r \ominus p_l^i.r = 1$, $p_{l+1}^i.r \ominus p_l^i.r = R$ et $p_{l+1}^{i+1}.r \ominus p_l^{i+1}.r = R - 1$ et comme $p_{l+1}^{i+1}.r \ominus p_{l+1}^i.r = 0$ on en déduit l'égalité.
2. Si $R < 0$, on fait la même étude en échangeant les rôles de p_l et p_{l+1}.
3. Si $R = 0$ alors quatre cas sont possibles suivant que les deux processus p_l et p_{l+1} s'incrémentent ou non :

   (a) Si les deux processus s'incrémentent alors : $p_{l+1}^{i+1}.r \ominus p_{l+1}^i.r = 1$, $p_l^{i+1}.r \ominus p_l^i.r = 1$ et $p_{l+1}^{i+1}.r \ominus p_l^{i+1}.r = 0$, et comme $p_{l+1}^i.r \ominus p_l^i.r = 0$, on en déduit l'égalité.

   les trois autres cas se traitent de la même manière et dans chacun des cas il y a égalité.

□

Le lemme suivant résulte de la proposition précédente.

Lemme 15 Soit μ un chemin $p_0 p_1 \dots p_k$. Considérons la transition $\gamma \to \gamma'$ avec γ dans Γ_M. Soient Δ et Δ' les retards sur le chemin μ pour les états γ et γ' respectivement. Durant la transition $\gamma \to \gamma'$ on a :

1. Si p_0 est incrémenté et p_k ne l'est pas, alors $\Delta' = \Delta - 1$.
2. Si p_0 et p_k ensemble sont incrémentés ou non incrémentés, alors $\Delta' = \Delta$.
3. Si p_k est incrémenté et p_0 ne l'est pas, alors $\Delta' = \Delta + 1$.

Corollaire 1 Clairement, si $\mu = p_0, p_1, \dots, p_0$ est un cycle, alors Δ_μ est un invariant congru à 0 modulo K.

Définition 23 Si $\mu = p_0, p_1, \dots, p_0$ est un cycle, alors l'invariant $|\Delta_\mu|$ est appelé le *résiduel* de μ.

On obtient le théorème important suivant :

Théorème 16 Le retard sur le réseau est intrinsèque si et seulement si les résiduels sur tous les cycles sont égaux à 0.
Si le retard est intrinsèque, alors il le demeure durant toute l'exécution du protocole 2, et alors la relation de préordre $\preceq_\gamma$ sera toujours définie.

Corollaire 1 Si $\gamma \in \Gamma_M^0$ alors γ n'est pas interbloqué.

Preuve : Les processus minimaux pour la relation $\preceq_\gamma$ ne sont pas bloqués. Ces processus existent car le réseau est fini.

□

On obtient la proposition importante :

Proposition 17 Si $\gamma \in \Gamma_M^0$, γ n'est pas interbloqué,et si $\gamma \to \gamma'$ est une transition alors $\gamma' \in \Gamma_M^0$ et l'exécution est potentiellement infinie.

Ces résultats sont fondamentaux, il reste maintenant à trouver une bonne condition suffisante pour que les résiduels soient tous nuls.

5.1.4 Le retard est une forme linéaire.

5.1.4.1 Codage des cycles dans un espace vectoriel. Il est souvent intéressant de généraliser une notion, ou au moins de la voir dans un cadre plus général pour mieux la comprendre. On peut étudier la notion de retard en la considérant comme une forme linéaire un $\mathbb{R}-$espace vectoriel. On pourrait ici utiliser la notion de $\mathbb{Z}-$module libre, mais comme cette notion est moins bien connue en général et que cette restriction n'apporte rien de plus, nous n'utiliserons que des espaces vectoriels sur le corps des réels.
Soit m le nombre des arêtes du graphe G. Numérotons ces arêtes de 1 à m, notons l'ensemble de ces arêtes $A = \{a_1, ..., a_m\}$. Considérons l'espace vectoriel $\mathbb{R}^m$et sa base canonique

$(\vec{e}_1, ..., \vec{e}_m)$. Orientons chacune des arêtes de G arbitrairement, on obtient un ensemble $\vec{A} = \{\vec{a}_1, ..., \vec{a}_m\}$.

Exemple

Sur la figure 2.9, nous avons représenté un réseau de 6 processus :

1. La numérotation des arêtes est arbitraire, ainsi que l'orientation de chacune d'elles.
2. Le codage du cycle 1245 est donné par le vecteur $(1, 1, -1, 0, 0, 1, 0, 0, 0)$.
3. Le codage du cycle 126345 est donné par le vecteur $(1, 1, -1, 0, -1, 0, 1, 1, 0)$.

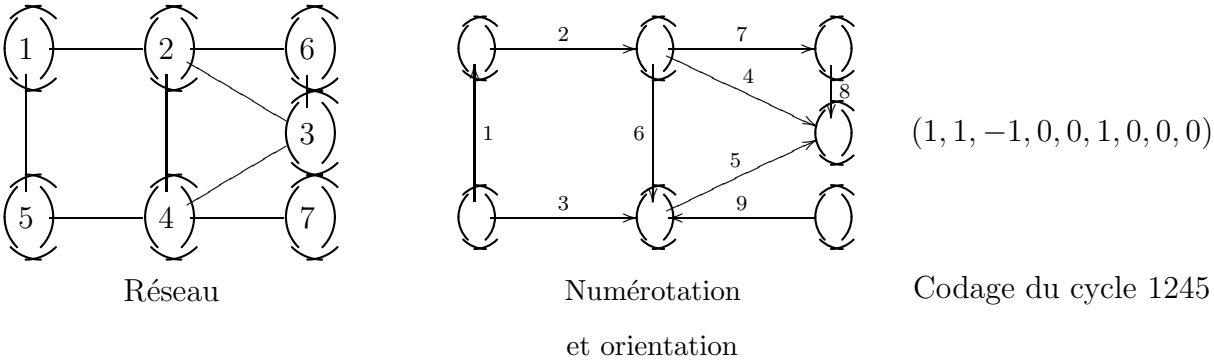


FIG. 2.9 – Exemple de codage.

5.1.4.2 Bases de cycles. Par ce codage, l'ensemble des cycles élémentaires forme un ensemble de vecteurs de l'ensemble $\mathbb{R}^m$, éventuellement vide. Notons cet ensemble σ. Il représente l'ensemble des cycles du réseau. σ est une partie de $\mathbb{R}^m$, on peut donc considérer le sous-espace vectoriel engendré par cette partie, noté $vect\,(\sigma)$. Par définition $vect\,(\emptyset) = \left\{\vec{0}\right\}$.

Exemple

Reprenons l'exemple de la figure 2.9.

Pour simplifier la notation, posons :

$$\vec{e}_1 = (1, 1, -1, 0, 0, 1, 0, 0, 0)$$
$$\vec{e}_2 = (1, 1, -1, 0, -1, 0, 1, 1, 0)$$
$$\vec{e}_3 = (0, 0, 0, 1, 0, 0, -1, -1, 0)$$

.

Le vecteur $\vec{e}_3$ code le cycle 236.

Le cycle 12345 est représenté par le vecteur $\vec{\epsilon} = (1, 1, -1, 1, -1, 0, 0, 0, 0)$, on peut remarquer que $\vec{\epsilon} = \vec{e}_2 + \vec{e}_3$.
Le cycle 234 est représenté par $\vec{\epsilon'} = (0, 0, 0, 1, -1, -1, 0, 0, 0)$ et $\vec{\epsilon'} = \vec{\epsilon} - \vec{e}_1 = -\vec{e}_1 + \vec{e}_2 + \vec{e}_3$.
Le cycle 2634 est représenté par $\vec{\epsilon''} = (0, 0, 0, 0, 0, -1, -1, 1, 1)$ et $\vec{\epsilon''} = \vec{\epsilon'} - \vec{e}_3 = -\vec{e}_1 + \vec{e}_2$.
Le cycle 1263245 n'est pas simple, il n'est pas "dans" σ, mais il est généré par des vecteurs de σ. Il se code par $\vec{e}_1 + \vec{e}_3$.
L'ensemble des cycles simples est :

$$\sigma = \left\{\vec{e}_1, -\vec{e}_1, \vec{e}_2, -\vec{e}_2, \vec{e}_3, -\vec{e}_3, \vec{\epsilon}, -\vec{\epsilon}, \vec{\epsilon'}, -\vec{\epsilon'}, \vec{\epsilon''}, -\vec{\epsilon''}\right\}$$

$\Sigma = vect\,(\sigma)$ est un sous-espace vectoriel de $\mathbb{R}^m$. La dimension de ce sous-espace vectoriel est le *nombre cyclomatique* de G [Ber83]. Une famille génératrice de Σ est l'ensemble σ. Si Σ n'est pas réduit au vecteur nul, c'est à dire si σ n'est pas vide, ou encore si G n'est pas un arbre (G est supposé connexe), alors on peut extraire de σ une base de Σ.

Définition 24 Si G contient au moins un cycle, on appelle base de cycle du graphe G toute famille de cycles simples de G dont le codage est une base de $\Sigma = vect\,(\sigma)$. La dimension du sous-espace vectoriel Σ est le *nombre cyclomatique* de G.

Exemple

Reprenons l'exemple de la figure 2.9.
Une base est $B = (\vec{e}_1, \vec{e}_2, \vec{e}_3)$. Cette famille est clairement génératrice, comme nous l'avons montré plus haut. C'est une famille libre puisque :

$$\begin{aligned} \alpha\vec{e}_1 + \beta\vec{e}_2 + \gamma\vec{e}_3 &= \vec{0} \Leftrightarrow \\ (\alpha+\beta, \alpha+\beta, -\alpha-\beta, \gamma, -\beta, \alpha, \beta-\gamma, \beta-\gamma) &= (0,0,0,0,0,0,0,0,0) \\ &\Leftrightarrow \alpha = \beta = \gamma = 0 \end{aligned}$$

Le nombre cyclomatique de G est donc 3.
Voir la base de cycle sur la figure 2.10.

5.1.4.3 Caractéristique cyclomatique d'un graphe. L'étude du retard nous conduit à introduire une nouvelle constante topologique sur les graphes, la *caractéristique cyclomatique*. Celle-ci va jouer un grand rôle dans la suite de cette étude. Si le graphe contient des cycles, à chaque base de cycles B on peut associer la longueur d'un plus long cycle de cette base. C'est un entier qui prendra une valeur entière strictement plus grande que 2. Il est alors intéressant de connaître la plus petite de ces valeurs, qui existe clairement, on appelle cette valeur la caractéristique cyclomatique du graphe.

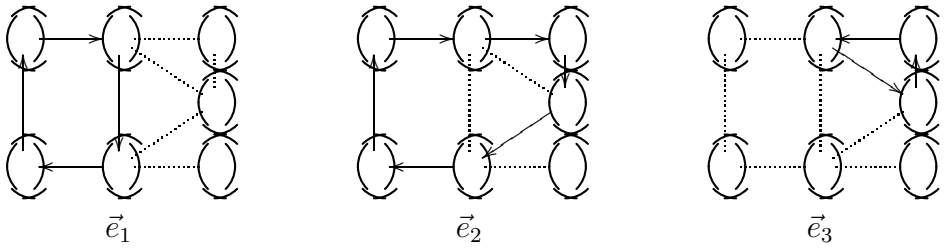


FIG. 2.10 – Base de cycles $B = (\vec{e}_1, \vec{e}_2, \vec{e}_3)$.

Définition 25 [Caratéristique cyclomatique] Si G est un graphe sans cycle, alors sa caractéristique cyclomatique C_G est égale à 2. Si G contient des cycles, soit B une base de cycles, la longueur d'un plus grand cycle de B est noté $\lambda(B)$. La caractéristique cyclomatique de G, notée C_G, est égale au minimum des $\lambda(B)$ quand B parcourt l'ensemble des bases de cycles de G.

Exemples

1. Sur l'exemple de la figure 2.9, $\lambda(B) = 6$. Peut-on trouver une base avec une constante plus petite ? La base $(\vec{e}_1, \vec{\epsilon}', \vec{e}_3)$ répond à la question avec une constante $\lambda((\vec{e}_1, \vec{\epsilon}', \vec{e}_3)) = 4$. Il est facile de montrer qu'on ne peut dscendre en dessous, par exemple en épuisant les cas, et que donc la constante cyclomatique du graphe G est 4. Voit figure 2.11.

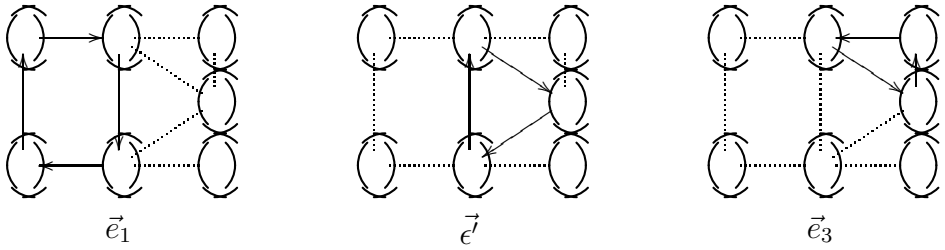


FIG. 2.11 – Base de cycles $\left(\vec{e}_1, \vec{\epsilon}', \vec{e}_3\right)$, ici $C_G = 4$.

2. Sur les réseaux réguliers, on obtient :

   La *caractéristique cyclomatique* d'un réseau carré est 4
   La *caractéristique cyclomatique* d'un réseau hexagonal est 6
   La *caractéristique cyclomatique* d'un graphe complet est 3
   La *caractéristique cyclomatique* d'un graphe triangulé est 3
   La *caractéristique cyclomatique* d'un arbre est 2
   La *caractéristique cyclomatique* d'un anneau à N nœnds est N

La *caractéristique cyclomatique* d'un hypercube d'ordre $n \geq 2$ est 4
La *caractéristique cyclomatique* d'un réseau apériodique de Penrose est 4

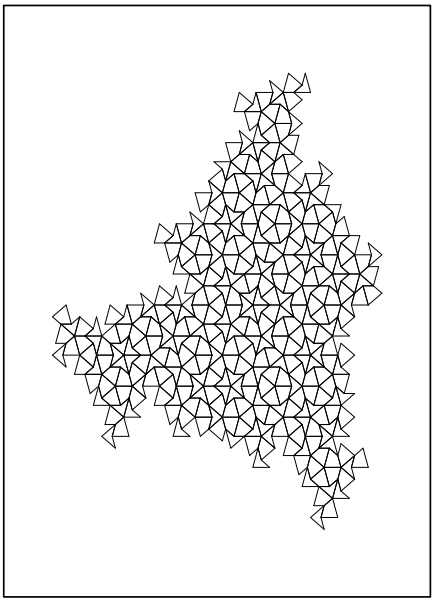


FIG. 2.12 – Un réseau apériodique de Penrose.

Ces exemples montrent que la caractéristique cyclomatique d'un réseau régulier peut ne pas dépendre de la taille du réseau.
Le problème de trouver une base de cycles B d'un graphe G avec la propriété que la longueur d'un plus grand cycle soit égale à la caractéristique cyclomatique du graphe a été étudié dans [CGH95]. Les auteurs prouvent que ce problème a une complexité en temps dans $O(m^3n)$. Ainsi, C_G peut être calculé en temps polynomial dans $O(m^3n)$. Bien entendu, la question de faire ce calcul de manière distribuée est une autre question, que nous n'aborderons pas ici.

5.1.4.4 Formes linéaires et retard. Reprenons notre identification des arêtes du graphe G avec des vecteurs de $\mathbb{R}^m$. m est le nombre des arêtes du graphe G. Elles sont numérotées de 1 à m, l'ensemble des arêtes est noté :$A = \{a_1, ..., a_m\}$. La base canonique de $\mathbb{R}^m$ est $(\vec{e}_1, ..., \vec{e}_m)$. Les arêtes de G ont été orientées arbitrairement, $\vec{A} = \{\vec{a}_1, ..., \vec{a}_m\}$. On associe à chaque arête $\vec{a}_i$ le vecteur $\vec{e}_i$, cette relation a défini notre codage des cycles. On peut ainsi coder les chemins, mais aussi les marches et bien d'autres choses.
Considérons maintenant le graphe G en tant que graphe étiqueté appartenant à Γ_M, ce qui nous garantit que les valeurs associées à deux nœnds voisins sont comparables. A chaque nœnd p on associe une valeur : la valeur du registre $p.r$, qui est un élément de $\mathbb{Z}_K$. Considérons la forme linéaire ψ de $\mathbb{R}^m \to \mathbb{R}$, définie sur la base $(\vec{e}_1, ..., \vec{e}_m)$ par :

$$\psi(\vec{e}_i) = q.r \ominus p.r$$

où $(p,q) = \vec{a}_i$. Pour simplifier les notations, nous noterons indifféremment $\psi(\vec{e}_i) = \psi(\vec{a}_i)$. Soit $\mu = p_0 p_1 .... p_r$ un chemin, ce chemin se code en un vecteur $\vec{u}$ de $\mathbb{R}^m$. On a

$$\begin{aligned} \Delta_\mu &= \sum_{k=0}^{r-1} r_{p_{k+1}} \ominus r_{p_k} \\ &= \sum_{k=0}^{r-1} \psi((p_k, p_{k+1})) \\ &= \psi(\vec{u}) \end{aligned}$$

Le retard suivant le chemin μ n'est autre que l'image par la forme linéaire ψ du vecteur $\vec{u}$. Le retard, donc ψ, est intrinsèque si et seulement si il est nul sur tous les cycles, donc si et seulement si ψ est nul sur σ, donc, par linéarité de la forme linéaire ψ, nul sur Σ. On obtient la proposition suivante :

Proposition 18 Le retard est intrinsèque si et seulement si $\Sigma \subset Ker(\psi)$

On déduit la condition nécessaire et suffisante suivante :

Corollaire 1 Si G contient des cycles, le retard est intrinsèque si et seulement si il est nul sur une base de cycles.

Théorème 19
Si K est strictement plus grand que $M.C_G$, alors pour tout état γ dans Γ_M, le retard est intrinsèque pour toute paire de nœnds x et y dans V.

Preuve : Si G est un arbre, alors le théorème est évident, puisqu'il n'y a pas de cycle, donc il suffit que $K > 2M$. Supposons que G contient des cycles et notons B une base de cycles de G avec $\lambda(B) = C_G$. Soit $\mu = p_0, p_1 \ldots p_k$ un cycle de B. On sait que $\left| \sum_{i=0}^{k-1} (p_{i+1}.r \ominus_l p_i.r) \right| \leq M.C_G$, et que le résiduel sur μ est congru à 0 modulo K. Donc si $K > MC_G$, les résiduels sur les cycles de B sont nuls. Donc, par le corollaire précédent, le retard est intrinsèque sur V.

□

Comme corollaire nous obtenons le théorème important suivant :

Théorème 20 $K > M.C_G \Rightarrow \Gamma_M = \Gamma_M^0$.

Nous conjecturons que la réciproque de ce corollaire est vraie.

Nous venons de voir que si $K > M.C_G$, alors $\Gamma_M = \Gamma_M^0$. On en déduit donc, par la proposition 17, que si $K > M.C_G$ toute configuration de Γ_M est sans interblocage ; et par la proposition 11 l'exécution du protocole est sans famine. On peut énoncer le théorème important :

Théorème 21 Si $K > M.C_G$ alors le protocole (démarrant dans Γ_M) est sans famine et, en particulier, il est sans interblocage.

5.2 Les relèvements d'une exécution

5.2.1 Relèvement d'un état

L'idée est de montrer que, quand tous les résiduels sont nuls, alors l'intuition de l'ordre total défini dans cas des registres à valeur dans $\mathbb{Z}$ reste correcte dans le cas de registres à valeur dans $\mathbb{Z}_K$. Pour cela nous allons associer à chaque processus p un registre virtuel supplémentaire noté $p.R$ et à valeurs dans $\mathbb{Z}$. Le contenu de ce registre sera appelé le *relèvement* dans $\mathbb{Z}$ du contenu du registre $p.r$.
L'objectif du *relèvement* étant de montrer que l'on peut interpréter une configuration par une autre dont les valeurs des registres sont prises dans l'ensemble des entiers naturels, ce *relèvement* doit être compatible avec les relations d'inégalités pour l'ordre local dans Γ_M et pour l'ordre naturel dans $\mathbb{Z}$. Les notions de retard pour les registres $p.r$ et les registres $p.R$ doivent coïncider. Ainsi, si un relèvement existe pour une configuration $\gamma \in \Gamma_M$, la relation de précédence deviendra l'ordre naturel sur les entiers pour les registres R, et le retard entre deux processus p et q satisfera $\Delta_{p,q} = q.R - p.R$.
La question du relèvement peut s'exprimer formellement de la manière suivante :
Etant donné une distribution :

$$\gamma = (p.r)_{p \in V} \in \Gamma_M$$

existe-t-il une distribution :

$$\delta = (p.R)_{p \in V} \in \mathbb{Z}^n$$

telle que (condition de compatibilité) :

$$\forall p \in V, \forall q \in \mathcal{N}_p : (p.r \leq_l q.r \Leftrightarrow p.R \leq q.R\ ) \text{et}\ \ (d_K\,(p.r, q.r) = |p.R - q.R|)$$

Une telle distribution est appelée un relèvement de γ.
Comme l'ordre est conservé et les écarts aussi, le lemme suivant est immédiat :

Lemme 22 Si δ est un relèvement de γ alors :

$$\forall p \in V, \forall q \in \mathcal{N}_p : q.r \ominus p.r = q.R - p.R$$

On en déduit, puisque le retard est toujours intrinsèque sur δ, qu'il l'est donc aussi sur γ, d'où le lemme :

Lemme 23 Si $\gamma \in \Gamma_M$ admet un relèvement, alors $\gamma \in \Gamma_M^0$

Lemme 24 Si δ est un relèvement de γ, alors si $p \in V$, on a pour tout $q \in V$: $q.R = p.R + \Delta_{pq}\,(\gamma)$

On voit que la définition de δ ne dépend que du choix arbitraire du contenu de l'un des registres, ici $p.r$. Le relèvement est défini à une constante près. Ainsi, si $\gamma \in \Gamma_M^0$ alors il y a une infinité de relèvements de l'état γ, on en déduit une réciproque :

Proposition 25 $\gamma = (p.r)_{p\in V} \in \Gamma_M$ admet un relèvement si et seulement si $\gamma \in \Gamma_M^0$. Dans ce cas, il existe une infinité de relèvements de γ, ils sont définis à une constante additive près :
pour $p \in V$, δ est un relèvement de γ ssi $\forall q \in V,\ q.R = p.R + \Delta_{pq}(\gamma)$.
Dans ces conditions : $\forall p \in V, \forall q \in V, q.r = \overline{p.R + q.R - p.r}\,[K]$.

Définition 26 On note ρ la projection :

$$\begin{array}{rl} \rho : & \mathbb{Z}^V \to (\mathbb{Z}/K\mathbb{Z})^V \\ & (q.R)_{q\in V} \to (q.r)_{q\in V} \end{array}$$

Remarquez que cette projection est indépendante du choix de p.

5.2.2 Relèvement d'une exécution

Reconsidérons maintenant le protocole 1 sur des processus à registres à valeurs dans $\mathbb{Z}$. Une question intéressante est de savoir si, par le relèvement, on peut interpréter une exécution du protocole 2, par une exécution du protocole 1.
Concrètement : considérons une transition $\gamma_0 \stackrel{\mathfrak{D}_0}{\to} \gamma_1$, où $\mathfrak{D}_0$ représente l'ensemble des processus qui font une action lors de la transition ; si $\gamma_0 \in \Gamma_M^0$ alors $\gamma_1 \in \Gamma_M^0$.
Soit maintenant δ_0 un relèvement de γ_0. Les processus qui peuvent faire une action dans γ_0, pour le protocole 2 et les processus qui peuvent faire une action dans δ_0 pour le protocol 1 sont les mêmes.
Considèrons donc la transition $\delta_0 \stackrel{\mathfrak{D}_0}{\to} \delta_1$, que peut-on dire de δ_1 ?

Proposition 26 Avec les notations ci-dessus, δ_1 est un relèvement de γ_1. Voir figure 2.13.

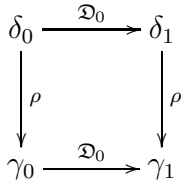


FIG. 2.13 – Relèvement d'une transition.

Considérons maintenant une exécution $\gamma_0 \stackrel{\mathfrak{D}_0}{\to} \gamma_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \gamma_k \stackrel{\mathfrak{D}_k}{\to} ...$ Si δ_0 est un relèvement de γ_0 alors, par récurrence, l'éxécution :

$$\delta_0 \stackrel{\mathfrak{D}_0}{\to} \delta_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \delta_k \stackrel{\mathfrak{D}_k}{\to} ... \tag{2.2}$$

est une exécution du protocole 1, et, de plus, pour tout $k \in \mathbb{N}$, δ_k est un relèvement de γ_k. Nous somme maintenant en mesure de définir ce qu'est le relèvement d'une exécution.

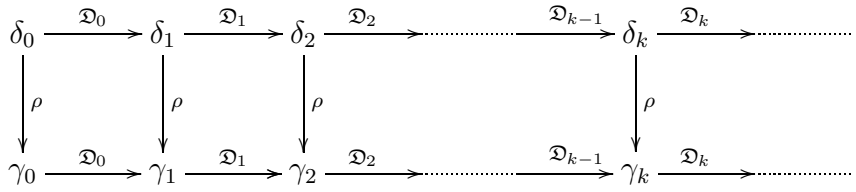


FIG. 2.14 – Relèvement d'une exécution.

Définition 27 Avec les notations ci-dessus, on appelle $\delta_0 \stackrel{\mathfrak{D}_0}{\to} \delta_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \delta_k \stackrel{\mathfrak{D}_k}{\to} ...$ un relèvement de $\gamma_0 \stackrel{\mathfrak{D}_0}{\to} \gamma_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \gamma_k \stackrel{\mathfrak{D}_k}{\to} ...$.

Voir la figure 2.14

Proposition 27 Soit $\gamma_0 \stackrel{\mathfrak{D}_0}{\to} \gamma_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \gamma_k \stackrel{\mathfrak{D}_k}{\to} ...$ une exécution du protocole 2, cette exécution admet un relèvement si et seulement si $\gamma_0 \in \Gamma_M^0$.
Ce relèvement $\delta_0 \stackrel{\mathfrak{D}_0}{\to} \delta_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \delta_k \stackrel{\mathfrak{D}_k}{\to} ...$ est uniquement déterminé par le choix du relèvement δ_0 de l'état γ_0.

5.2.3 Théorème fondamental

A la fin de cette quête, revenons à la question initiale : a-t-on $\Gamma_M \triangleright \Gamma_1$?
En général, la réponse est non. Prendre par exemple $M = 2$, $K = 10$ et un anneau à 5 éléments. Mettre dans les registres respectivement $0, 2, 4, 6, 8$. L'anneau est interbloqué, il n'y a pas convergence.

Théorème 28
$\Gamma_M^0 \triangleright \Gamma_1^0$.
Si $\gamma \in \Gamma_M^0$ est l'état de départ, alors la convergence se fait en au plus ϖ rondes, où ϖ est le retard maximal dans le réseau pour l'état γ.
De plus $\varpi \leq Md$ où d est le diamètre du réseau.

Preuve : Le protocole est vivace d'après le corollaire 1 du théorème 16.
Soit $\gamma_0 \stackrel{\mathfrak{D}_0}{\to} \gamma_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \gamma_k \stackrel{\mathfrak{D}_k}{\to} ...$ une exécution et $\delta_0 \stackrel{\mathfrak{D}_0}{\to} \delta_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \delta_k \stackrel{\mathfrak{D}_k}{\to} ...$ un relévement de cette exécution. La notion de ronde est la même pour les exécutions des protocoles 1 et 2. D'aprés la proposition 6, l'exécution $\delta_0 \stackrel{\mathfrak{D}_0}{\to} \delta_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \delta_k \stackrel{\mathfrak{D}_k}{\to} ...$ converge vers Γ_1 (pour $\chi = \mathbb{Z}$ et dans Γ_∞) en au plus ϖ rondes, il en est de même pour $\gamma_0 \stackrel{\mathfrak{D}_0}{\to} \gamma_1 \stackrel{\mathfrak{D}_1}{\to} ... \stackrel{\mathfrak{D}_{k-1}}{\to} \gamma_k \stackrel{\mathfrak{D}_k}{\to} ...$ vers Γ_1^0. Il reste que ϖ est clairement majoré par Md puisque l'état de départ est dans Γ_M.

□

En utilisant le théorème 20, on déduit du théorème précédent celui-ci :

Théorème 29 Si $K > M.C_G$ alors $\Gamma_M \triangleright \Gamma_1$.

5.2.4 Visualisation du préordre total

Ce paragraphe a pour seul objet de visualiser le préordre total défini par une synchronisation de phases démarrant dans Γ^0_M, ou encore dans Γ_M à la condition que $k > M.C_G$. Le retard est alors intrinsèque et définit un *préordre total* sur l'ensemble des processus. Tous les processus qui sont minimaux pour ce préordre, ou généralement localement minimaux, peuvent s'incrémenter.
Comme de plus Γ^0_M est stable sous l'action (NA), il n'y aura jamais d'interblocage ni de famine.
Une autre manière de dire les choses est d'utiliser le théorème de relèvement. Celui-ci dit simplement que dans Γ^0_M l'intuition est la même que dans le cas de registres à valeurs dans $\mathbb{Z}$, où la notion de retard est très visuelle. On peut imager cela trés bien par les deux figures 2.15 et 2.16, où un axe vertical repésente un relèvement de la configuration. Sur ces figures est représenté un réseau en forme de grille, les processus qui peuvent exécuter l'action (NA) sont en blanc, ceux qui sont bloqués sont en noir. La première figure montre le réseau à l'unisson strict et donc en situation de concurrence maximale, chaque processus a son horloge à la même valeur que ses voisins. La deuxième figure montre un état plus général dans Γ^0_1. On voit clairement ce que signifie *pouvoir exécuter l'action* (NA).

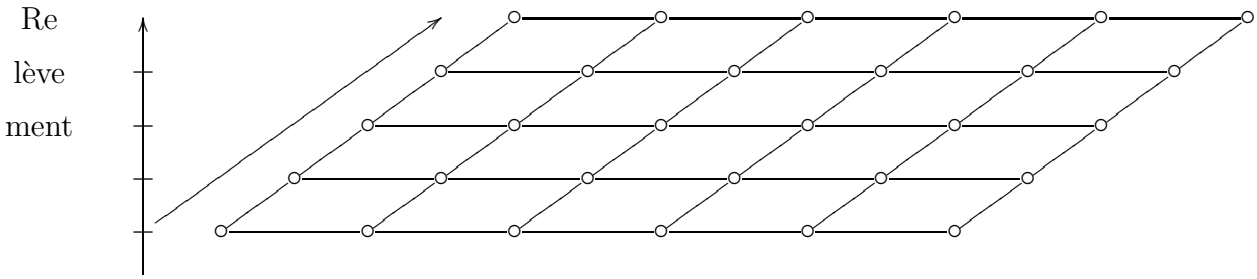


FIG. 2.15 – Unisson : synchronisation forte

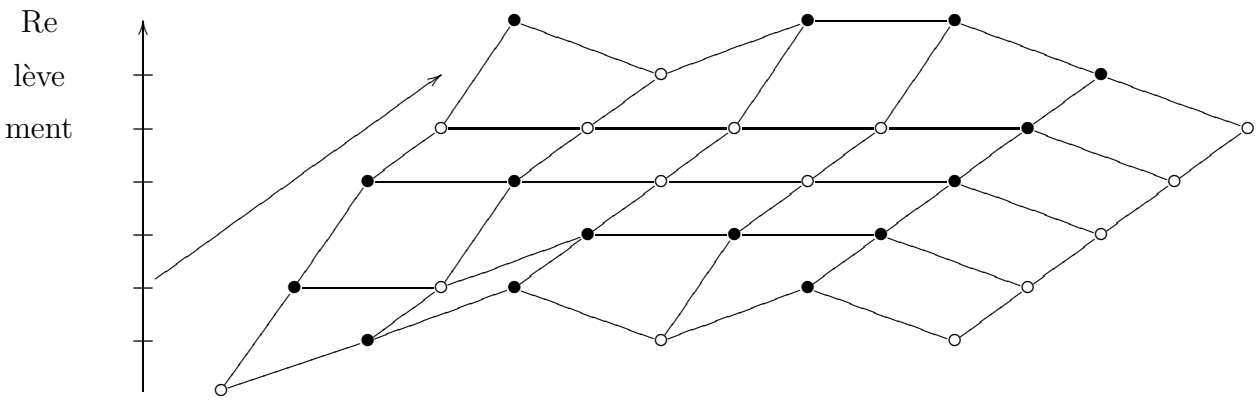


FIG. 2.16 – Unisson : visualisation du préordre

6 Exemple d'application : construction optimale d'un arbre couvrant en largeur d'abord

Dans cette partie, nous donnons un exemple d'application de ce que nous venons de développer. Ce travail est en cours de rédaction avec Larry Larmor, Ajoy Datta et Franck Petit. Etant donné une racine *Root* du réseau, nous présentons une construction d'un arbre de recherche couvrant en largeur d'abord, optimale en temps et en espace. En fait nous proposons trois algorithmes. Le premier construit le Dag de recherche en largeur d'abord enraciné en *Root* ; puis, en utilisant trois couleurs, nous donnons un algorithme de construction d'un arbre de recherche en largeur d'abord avec détermination de terminaison. Troisièmement, nous proposons une version de ce deuxième algorithme dans le modèle à passage de messages.
Cet exemple montre combien le modèle à états, même s'il est un modèle abstrait, parfois considéré comme loin de la réalité informatique, n'en demeure pas moins un modèle fécond en ce sens que, dans un premier temps, il permet une étude fine de questions générales (sans se préoccuper des problèmes de synchronisation dans les communications). Ensuite, le passage à un modèle moins abstrait devient une question cruciale, question qui peut être simple, comme c'est le cas dans l'exemple que nous allons développer, mais qui peut être aussi très difficile : par exemple : l'unisson à réinitialisation local (voir algorithme 7) que nous proposons dans le chapitre 3 ne passe pas tel-quel dans le modèle à registres.

6.1 Introduction

Les arbres de recherche en largeur d'abord sont une composante de base dans la conception de nombreux algorithmes distribués, comme par exemple les algorithmes de diffusion, de routage, de calcul global, d'élection d'un chef, ou de synchronisation. On trouvera un autre exemple d'application important, celui de la conception d'un unisson avec un temps de

convergence en $O(d)$ (voir algorithmes 24 et 25 de l'annexe A).
La littérature sur la construction d'un arbre couvrant en largeur d'abord dans un environnement réparti sans faute est importante [Gal82, Awe85, AG85, AG87, Awe89, AP90]. Comme le montre [AGLP89], sur un réseau avec n nœnds, m arêtes (canaux) et un diamètre d, la complexité intrinsèque en messages et en temps de la construction d'un arbre couvrant en largeur est respectivement dans $\Omega(m)$ et dans $\Omega(d)$. Dans [PV00], les auteurs établissent formellement la borne inférieure en terme de nombre d'états par processus, pour construire une circulation de jetons en profondeur d'abord. Comme conséquence de ce résultat, le nombre d'états nécessaires pour construire un arbre couvrant est dans $\Omega(\Delta)$, où Δ est le degré maximum du réseau. Cette borne inférieure est la même pour la construction des arbres en largeur d'abord.
Les constructions d'arbres couvrants en largeur $AsynchBFS$ et $LayeredBFS$ sont bien connues, et proposées dans [Lyn96a] ; elles présentent le problème du compromis entre la complexité en espace et la complexité en temps dans la conception d'un algorithme performant. $AsynchBFS$ est dans $O(d)$ en complexité en temps, il est optimal, mais son coût en espace est dans $O(d)$. Par contre, $LayeredBFS$ est optimal en espace, sa complexité est en $O(\Delta)$ états par processus , mais sa complexité en temps est dans $O(d^2)$. La meilleure solution en complexité en messages est, à notre connaissance, dans $O(m+n\log^3 n)$ [AP90]. Cependant, la complexité en temps de cette solution est dans $O(d\log^3 n)$ et sa complexité en espace est dans $\Omega(d)$. Il n'existe pas, à notre connaissance, de protocole autre que $LayeredBFS$ qui soit optimal en espace. Nous proposons dans ce qui suit le premier protocole de construction d'arbre couvrant en largeur, qui soit à la fois optimal en temps et en espace.

6.2 Construction du Dag couvrant en largeur

L'idée est très simple : on définit une synchronisation de phase sur le graphe G, chaque processus p maintient un registre $p.r$ à valeurs dans $\mathbb{Z}_3 = \{0, 1, 2\}$. Ici $K = 3$ et $\alpha = 0$. φ désigne l'incrémentation dans $\mathbb{Z}_3$. On note $d(p, q)$ la distance entre p et q dans le réseau défini par G. On appelle δ l'écartement de la racine $Root$, c'est-à-dire le maximum, sur l'ensemble des $p \in V$, de la longueur d'un plus court chemin de $Root$ vers p.
L'algorithme démarre dans l'état γ_0 où tous les registres sont à 0. On est donc dans Γ_1^0. Nous ne faisons aucune hypothèse sur la constante cyclomatique du graphe. Cela n'est pas nécessaire, puisque la synchronisation de phase démarre dans Γ_1^0, où tous les résiduels sont nuls, ils le resteront durant l'exécution de l'algorithme. Cela veut dire que l'outil fondamental qu'est le retard est, et restera, bien défini (c'est-à-dire intrinsèque, indépendant du chemin choisi). On note toujours Δ_{pq} le retard entre les processus p et q. La racine ne fait aucune action, les autres processus effectuent l'action gardée d'incrémentation de phase (NA). Le protocole est donné par l'algorithme 3.
Dans la suite, on note $Inc(p)$ le retard de la racine par rapport au processus p ; ainsi $Inc(p) = \Delta_{Root,q}$. Comme la racine ne fait aucune incrémentation, $Inc(p)$ est aussi le nombre d'incrémentations effectué par registre r du processus p dans l'état γ courant. On notera, quand cela sera nécessaire, $Inc(p)(\gamma)$ la valeur de $Inc(p)$ dans l'état γ.

Algorithme 3 Algorithme de construction du BFS_dag pour p,

Constante et variable :
$\mathcal{N}_p$: : l'ensemble des voisns de p ;
$p.r \in \mathbb{Z}_3$: initialisé à 0 ;
Fonctions booléennes :
$NormalStep_p \equiv \forall q \in \mathcal{N}_p : ((p.r = q.r) \vee (q.r = \varphi(p.r))$;
Action :
Si p n'est pas la racine :
$NA : NormalStep_p \longrightarrow p.r := \varphi(p.r)$;

Soit γ un état du système, accessible par notre algorithme à partir de γ_0, ce qui signifie que $\gamma \in \Gamma_1^0$. On note $Haut(p)$ la distance de la racine $Root$ à p, c'est-à-dire la quantité $dist(Root, p)$.
On note :

$$\Psi(\gamma) = \sum_{p \in V} Inc(p)(\gamma)$$

Il est clair que :

$$\Psi(\gamma) \leq \sum_{p \in V} Haut(p)$$

Rappelons la définition du graphe d'interblocage (page 61) : Soit $\gamma \in \Gamma_1^0$, le graphe d'interblocage associé à l'état γ est le graphe défini sur V par la relation :

$$p \rightarrow q \Leftrightarrow q \in \mathcal{N}_p \text{ et } q.r = \varphi(p.r) \tag{2.3}$$

Ce graphe est noté $\mathcal{L}(\gamma)$. Nous savons que $\mathcal{L}(\gamma)$ est sans circuit car $\gamma \in \Gamma_1^0$.

Définition 28 Le $BFS - Dag$ est le graphe d'interblocage de l'état $\tilde{\gamma}$ défini par la distribution :

$$\forall p \in V, p.r \equiv Haut(p) \bmod 3 \tag{2.4}$$

Remarquons que $\tilde{\gamma}$ est l'unique état de Γ_1^0 tel que $\forall p \in V, Inc(p) = Haut(p)$; c'est pourquoi nous appelons ce Dag : le BFS-Dag. Remarquons que $\Psi(\tilde{\gamma}) = \sum_{p \in V} Haut(p)$.
La racine ne fait aucune incrémentation, et Ψ est strictement croissant à chaque transition, à valeurs entières et majoré. On en déduit que l'exécution de notre protocole aboutit en un nombre fini d'étapes à un interblocage. Ainsi toute exécution maximale est finie.

Lemme 30 $\gamma \in \Gamma_1^0$ est un interblocage si et seulement si $\mathcal{L}(\gamma)$ =BFS-Dag.

Preuve : Soit γ un interblocage dans Γ_1^0. Supposons que la proposition n'est pas vraie, alors la quantité $Haut(p) - Inc(p)$ est positive ou nulle, mais pas toujours nulle, son maximum sur V est donc strictement positif. Soit p un plus proche processus de la racine, qui rend maximale la quantité $Haut(p) - Inc(p)$, alors le processus p peut s'incrémenter, en effet, si q est un voisin de p :

1. si $Haut(q) < Haut(p))$ alors $q.r = p.r$ car p est un plus proche processus de la racine, qui maximise la quantité : $Haut(p) - Inc(p)$.

2. Si $Haut(q) \geq Haut(p)$ alors $q.r \geq_l p.r$, sinon $Haut(p) - Inc(p)$ ne serait pas maximal.

On en déduit que le processus p peut s'incrémenter et que la configuration n'est pas interbloquée.

□

Par récurrence on obtient :

Lemme 31 Soit p un processus : après k rondes, si $k \leq Haut(p)$ alors $Inc(p) \geq k$ sinon $Inc(p) = Haut(p)$. Après δ rondes, $\Psi(\gamma) = \sum_{p \in V} Haut(P)$ et le BFS-Dag est construit.

Rappelons que δ est l'écartement de la racine *Root*, clairement $\delta \leq d$.

6.3 Construction de l'arbre couvrant en largeur

L'algorithme précédent, optimal en temps et en espace, présente deux inconvénients : il ne construit pas un arbre, mais seulement le Dag ; de plus, la racine ne sait pas quand la construction est finie. On remédie facilement à ces petits manques en introduisant un deuxième registre appelé *statut*. Celui-ci contient une des trois couleurs U, F et S. La racine *Root* commence avec la couleur F (finish), les autres commencent avec la couleur U (unfinish). Un processus p sait qu'il est sur le BFS-Dag en construction une fois qu'un de ses voisins q a le statut F avec la valeur de son registre $q.r$ strictement inférieure à celle du registre de p. Alors p prend le statut F. Une fois qu'une feuille prend le statut F et que ses voisins ont le statut F ou S (silent), la feuille prend le statut S et choisit un père en utilisant la procédure $choose\_parent()$. Si les voisins q d'un processus p qui satisfont $q.r = \varphi(p.r)$ ont le statut S et que les autres voisns ont le statut F, alors le processus p prend le satut S et, s'il n'est pas la racine, choisit un père en utilisant $choose\_parent_p()$. Quand la racine prend la couleur S, alors tous les autres processus ont la couleur S et l'arbre couvrant en largeur est construit. La terminaison est connue par la racine.

Algorithme 4 ($BFS0$) pour le processus p

Constantes et variables :
- $\mathcal{N}_p$: : l'ensemble des voisins de p ;
- $p.r \in \mathbb{Z}_3$: initialisé à 0 ;
- $p.statut \in \{U, F, S\}$: initialisation de Root par : F, initialisation de $p \neq Root$ par : U

Fonctions booléennes :
- $NormalStep_p \equiv \forall q \in \mathcal{N}_p : (p.r = q.r \vee (q.r = \varphi(p.r)))$;
- $Finish \equiv (p.statut = U) \wedge (\exists q_1 \in \mathcal{N}_p : (q_1.r <_l p.r) \wedge (q_1.statut = F))$;
- $Silent \equiv (p.statut = F) \wedge (\forall q \in \mathcal{N}_p : (q.statut \neq U) \wedge (q.r >_l p.r \Rightarrow q.statut = S))$;

Actions :

Si P est la racine :
- SA : $\forall q \in \mathcal{N}_{Root} : q.statut = S \quad \rightarrow \quad Root.statut := S$;

Si P n'est pas la racine :
- NA : $NormalStep_p \longrightarrow p.r := \varphi(p.r)$;
- FA : $Finish \longrightarrow p.statut := F$; $p.parent := choose\_parent_p()$;
- SA : $Silent \longrightarrow p.statut := S$

Suivant la section 6.2, après δ rondes le BFS-Dag est construit. Il est facile de montrer qu'une ronde plus tard, donc après $\delta + 1$ rondes, le statut de chaque processus est dans $\{F, S\}$. Après $\delta + 2$ rondes, au moins un processus a pour statut S et après $2\delta + 2$ rondes, le statut de tous les processus est S, en particulier le statut de la racine est S. Comme $\delta \leq d$, on conclut :

Proposition 32 Après au plus $2d + 2$ rondes, le protocole 4 définit un arbre couvrant en largeur d'abord (d est le diamètre du réseau).

6.4 Version dans le modèle à passage de messages

6.4.1 Le modèle

Pour bien distinguer le modèle à passage de messages du modèle à états, nous noterons les processus par une majuscule P dans le modèle à passage de messages. Les communications sont bidirectionnelles. Les messages envoyés d'un nœud vers un de ses voisins sont délivrés en un temps fini mais non borné. Les communications sont de type $FIFO$ sur chaque lien. Les actions du processus P mettent à jour une ou plusieurs variables de P, et soit lisent tous les messages disponibles soit envoient un message à tous les voisins de P, mais pas les deux à la fois. Une action n'est exécutée que si la garde est évaluée à vrai.
Le réseau est asynchrone, les actions sont exécutées atomiquement, c'est-à-dire que l'évaluation d'une garde et l'exécution de l'action correspondante, si elle est exécutée, sont effectuées en une étape atomique.
Ainsi, il y a deux sortes d'actions qu'un processus P peut effectuer :
– dans une action de lecture, P lit tous les messages de tous ses voisins
– dans une action locale, P peut changer une ou plusieurs de ses propres variables. Après avoir exécuté une action locale, P envoie immédiatement, dans la même action, tous les messages disponibles.

Dans notre algorithme, il est possible pour un processus Q d'envoyer jusqu'à quatre messages consécutifs à un voisin P, avant que P ne lise le premier message. Dans ce cas, P lit les messages dans l'ordre de leur envoi. Un processus ne peut pas à la fois lire et écrire un message en une seule étape.

6.4.2 Algorithme $\mathcal{BFS}$

Rappelons brièvement le principe de l'algorithme, dans le contexte du modèle à passage de messages. On suppose que chaque nœud P maintient une variable de niveau r : $P.r \in Z_3$. Dans la configuration initiale γ_0, tous les nœuds P ont leur registre à 0 : $P.r = 0$. On suppose aussi que chaque nœud P maintient une variable de niveau $P.r_Q$ pour chaque $Q \in \mathcal{N}_P$; la valeur, initialisée à 0, est ensuite égale à la plus récente valeur du registre r du processus Q a envoyé à P, les envois se font de manière relative, c'est-à-dire sous la forme d'un ordre d'incrémentation. Chaque nœud maintient aussi une variable *statut*, dont les trois valeurs possibles : U, F, ou S. On ordonne ces valeurs de la manière suivante : $U < F < S$. Au départ, tous les nœuds ont le satut U. Chaque nœud P contient aussi

une copie des valeurs des variables *statut* de chacun de ses voisins Q. Cette copie est la valeur de la variable $P.statut_Q$. Cette variable est mise à jour chaque fois qu'un message aproprié est reçu du voisin Q. Un nœud P peut incrémenter la valeur de son registre r si pour tout $Q \in \mathcal{N}_P$, on a $P.r \leq_l P.r_Q$. Un nœud prend le satut F quand il détermine qu'il ne peut plus s'incrémenter. Les nœuds prennent le statut S dans une vague partant des feuilles. Quand *Root* prend le statut S, tous les autres nœuds sont S, et plus aucune incrémentation n'est possible.

Variables du nœud P :

1. $P.r \in Z_3$, initialisé à 0 pour chaque P.
2. $P.r_Q \in Z_3$ pour tout $Q \in \mathcal{N}_P$, initialisé à 0 pour tout P, Q.
3. $P.statut \in \{U, F, S\}$, initialisé à U pour tout P.
4. $P.statut_Q \in \{U, F, S\}$ initialisé à U.
5. $P.parent \in \mathcal{N}_P + \{\perp\}$, initialisé à $\perp$ pour tout P.

Prédicats du nœud P :

1. $CanIncrement(P) \equiv (P \neq Root) \wedge (\forall Q \in \mathcal{N}_P P.r \leq_l P.r_Q)$,
2. $CanLock(P, Q) \equiv Q \in \mathcal{N}_P \wedge P.statut_Q = F \wedge P.r_Q <_l P.r$,
3. $CanBeSilent(P) \equiv (P.statut = F) \wedge (\forall Q \in \mathcal{N}_P P.statut_Q \geq F) \wedge$
   $(\forall Q \in \mathcal{N}_P P.r <_l P.r_Q \Rightarrow P.statut_Q = silent)$

Les messages envoyés par un nœud P vers $Q \in \mathcal{N}_P$ sont des constantes :

1. INCREMENT,
2. LOCKED,
3. SILENT.

6.4.3 Complexité en temps et en messages

Comme précédemment, $Haut(P)$ est la distance du nœud P à la racine, et $Inc(P)$ est le nombre de fois que P a exécuté l'action S1, i.e., le nombre de fois que $P.r$ a été incrémenté. De la même manière, $Inc_Q(P)$ est le nombre de fois que $P.r_Q$ a été incrémenté. On notera t le nombre de rondes exécutés depuis le début de l'exécution de l'algorithme. denote L'algorithme défini en message passing s'éxecute comme attendu grâce à l'hypothèse que les canaux sont FIFO.
On a clairement le lemme suivant :

Lemme 33
$P.r = Inc(P)$ **mod** 3.
Si $Q \in \mathcal{N}_P$ alors : $P.r_Q = Inc_Q(P)$ **mod** 3.

Du fait que les communications sont FIFO, on déduit :

Algorithme 5 Actions de $\mathcal{BFS}$

Actions :

	Etiquette	*Garde*		*Action*
$S1$	(*Increment*)	$CanIncrement(P)$	$\longrightarrow$	$P.r \leftarrow \varphi(P.r)$ Envoyer INCREMENT à tous les voisins
$S2$	(*Lock*)	$(P.statut = U) \wedge ((P = Root) \vee (\exists Q \in \mathcal{N}_P : CanLock(P, Q)))$	$\longrightarrow$	$P.statut \leftarrow F$ Envoyer LOCKED à tous les voisins
$S3$	(*Sleep*)	$CanBeSilent(P)$	$\longrightarrow$	$P.statut \leftarrow S$ Envoyer SILENT à tous les voisins
$S4$	(Receive INCREMENT)	INCREMENT reçu de $Q \in \mathcal{N}_P$ Ni S1 ni S2 ni S3 ne sont exécutables.	$\longrightarrow$	$P.r_Q \leftarrow \varphi(P.r_Q)$
$S5$	(Receive LOCKED)	LOCKED reçu de $Q \in \mathcal{N}_P$ Ni S1 ni S2 ni S3 ne sont exécutables.	$\longrightarrow$	$P.statut_Q \leftarrow F$
$S6$	(Receive SILENT)	SILENT reçu de $Q \in \mathcal{N}_P$ Ni S1 ni S2 ni S3 ne sont exécutables.	$\longrightarrow$	$P.statut_Q \leftarrow S$

Lemme 34 Si $Q \in \mathcal{N}_P$ alors :

1. $Inc(Q) - 1 \leq Inc_Q(P) \leq Inc(Q)$.
2. Si $Inc(Q) - 1 = Inc_Q(P)$ alors $Inc(P) \in \{Inc(Q), Inc(Q) - 1\}$.
3. Si $Inc(Q) = Inc_Q(P)$ alors $Inc(P) \in \{Inc(Q), Inc(Q) + 1, Inc(Q) - 1\}$.

De ce lemme on déduit la proposition importante :

Proposition 35 Durant toute l'exécution, le système reste dans Γ_1^0, et $Inc(P) \leq Haut(P)$.

La quantité :

$$\Psi(\gamma) = \sum_{P \in V} Inc(P)(\gamma)$$

est donc bien définie. Le lemme 30 est donc encore correcte au sens que l'action $S1$ ne sera plus jamais applicable dans le réseau si et seulement si le graphe d'interblocage de l'état courant est le BFS-Dag.

L'annalogue du lemme 31 devient :

Lemme 36 Supposons que $k \leq Haut(P)$, alors

1. Si $t \geq 2k - 1$, alors $Inc_Q(P) \geq k - 1$ pour tout $Q \in \mathcal{N}_P$.
2. Si $t \geq 2k$, alors $Inc(P) \geq k$.

Preuve : Par récurrence sur k. Le cas $k = 0$ est facile étant donné les conditions initiales. Supposons $k > 0$. On commence par prouver (1). On sait que $Haut(Q) \geq k-1$. Considérons la situation après les $2k-2$ premières rondes : par hypothèse de récurrence, $Inc(P) \geq k-1$ et $Inc(Q) \geq k - 1$. Si $Inc_Q(P) \geq k - 1$ la proposition est prouvée. Sinon, P ne peut pas exécuter les actions S1, S2, ou S3 durant l'étape suivante, et doit donc exécuter S4 pour

le voisin Q durant la ronde suivante, ainsi la proposition est vérifiée. (2) découle de (1) et de la définition de l'action S1, puisque P ne peut qu'effectuer l'action S1.

□

On en déduit comme corollaire la proposition suivante :

Proposition 37 A bout d'au plus 2δ rondes, le BFS-Dag est défini par les variables *level* des processus.

Lemme 38 Pour tout nœud P,

1. Si $P \neq Root$ et $t \geq 2 \cdot Haut(P)$, alors il existe $Q \in \mathcal{N}_P$ tel que $P.statut_Q \geq F$.
2. Si $t \geq 2 \cdot Haut(P) + 1$, alors $P.statut \geq F$.

Preuve : par récurrence sur $Haut(P)$. Pour $Haut(P) = 0$, $P = Root$. (1) n'est pas défini, de plus S2 est exécutable initialement et sera donc exécuté durant la première ronde ; donc (2) est satisfait.
Supposons $P \neq Root$, i.e., $Haut(P) \neq 0$. On prouve d'abord (1). Soit $Q \in \mathcal{N}_P$ tel que $Haut(Q) = Haut(P) - 1$. Par l'hypothèse de récurrence, Q envoie le message LOCKED à un moment donné, durant les $2 \cdot Haut(P) - 1$ premières rondes. Par les lemmes 35 et 36, P ne peut pas exécuter S1 durant la ronde $2 \cdot Haut(P) + 1$. Si $P.statut \geq F$ à la fin de la ronde $2 \cdot Haut(P) - 1$, la preuve est terminée. Sinon, P peut exécuter l'action S5 durant la ronde $2 \cdot Haut(P)$, et la proposition est démontrée. (2) découle de la définition de S2 et du fait que (1) a lieu pour un certain $Q \in \mathcal{N}_P$.

□

On obtient comme corollaire, la proposition :

Proposition 39 Au bout d'au plus $2\delta + 1$ rondes, tous les processus P satisfont : $P.statut \geq F$.

Lemme 40 Pour tout nœud P,

1. Si $Q \in \mathcal{N}_P$ et $t \geq 4 \cdot d - 2 \cdot Haut(P) + 2$, alors $P.statut_Q \geq F$.
2. Si $Q \in \mathcal{N}_P$ et $Haut(Q) = Haut(P) + 1$, et si $t \geq 4 \cdot d - 2 \cdot Haut(P) + 2$, alors $P.statut_Q = S$.
3. Si $t \geq 4 \cdot d - 2 \cdot Haut(P) + 3$, alors $P.statut = S$.

Preuve : Nous prouvons d'abord (1) :
Considérons la situation après $4 \cdot d - 2 \cdot Haut(P) + 1$ rondes. Par le lemme 36 : $inc(P) = Haut(P)$, et par le lemme 38 : $Q.statut \geq F$ et $P.statut \geq F$. Donc P ne peut plus exécuter ni S1 ni S2. Si P a déjà exécuté S3, la preuve est finie par le lemme 41 ; sinon P exécutera S6 pour Q dans la ronde prochaine, et la preuve est finie.
Nous prouvons (2) par induction décroissante sur $Haut(P)$:
Si $Haut(P) = \delta$, alors (2) n'est pas défini. Sinon, supposons que $Q \in \mathcal{N}_P$ et $Haut(Q) = Haut(P) + 1$. Considérons la situation après les $4 \cdot d - 2 \cdot Haut(P) + 1$ premières rondes : par les lemmes 36 et 38, $inc(P) = Haut(P)$ et $P.statut \geq F$. Si $P.statut = S$, alors, par

la garde de S3 pour P, c'est fini. Sinon, P exécutera S6 pour Q à la ronde suivante, et la preuve est finie.
De même nous prouvons maintenant (3) :
Considérons la situation après les $4 \cdot d - 2 \cdot Haut(P) + 2$ premières rondes : $inc(P) = Haut(P)$ et $P.statut \geq F$, comme prouvé au-dessus. Si $P.statut = S$, la preuve est finie. Sinon, P exécutera S3 à la ronde suivante, et la preuve est finie.

□

Le lemme suivant est immédiat :

Lemme 41 Si $Q \in \mathcal{N}_P$, alors $P.statut_Q \leq Q.statut$.
Si $Q \in \mathcal{N}_P$ et $Haut(Q) = Haut(P) + 1$, et si $P.statut = S$, alors $P.statut_Q = S$.

On en déduit le théorème :

Théorème 42 Quand $Root.statut = S$, plus aucune action n'est exécutable dans le réseau, et les valeurs de $\{P.parent\}_{P \in V}$ définissent l'arbre couvrant en largeur d'abord enraciné en *Root*.

Preuve : Par le lemme 41, $P.statut = S$ implique que $Q.statut = S$ pour tout $Q \in \mathcal{N}_P$ tel que $Haut(Q) = Haut(P) + 1$. Ainsi, par récurrence sur $Haut(P)$, $Root.statut = S$ implique que $P.statut = S$ pour tout $P \in V$. Puisque tous les nœuds, excepté *Root*, choisissent un père qui est plus proche de *Root*, alors ils exécutent S2 ; l'arbre qui en résulte est un sous-graphe du BFS-Dag, c'est un arbre en largeur d'abord.

□

Théorème 43 Après au plus $4 \cdot d + 3$ rondes, $Root.statut = S$.

Preuve : Immédiate grâce au lemme 40.

□

Théorème 44 l'espace utilisé par chaque nœud P est dans $O(\delta_P)$, où $\delta_P = \|\mathcal{N}_P\|$.

Preuve : Le nombre de variables de P est $2 \cdot \delta_P + 3$.

□

Théorème 45 La complexité en messages est $2 \cdot (d + 2) \cdot \|E\|$, où E est l'ensemble des arêtes de V.

Preuve : Chaque nœud envoie au plus $d + 2$ messages à chacun de ses voisins.

□

Théorème 46 La taille de chaque message est de $O(1)$ bits.

Preuve : Il y a trois types de messages, et aucun message ne transport de donnée.

□

7 Conclusion

Dans ce chapitre, nous avons commencé par étudier le cas particulier d'une horloge à valeurs dans $\mathbb{Z}$. Deux propriétés de $\mathbb{Z}$, son caractère monogène et son ordre total, permettent de résoudre le problème de l'unisson de manière très simple. Pour passer à des registres bornés, nous avons posè cinq questions :

1. Qu'est-ce qu'une incrémentation bornée ?
2. Comment définir un ordre local sur une incrémentation bornée ?
3. Comment éviter les interblocages ?
4. Comment passer d'un ordre local à un ordre global dans de tels systèmes ?
5. Comment gérer les états où l'ordre local n'est pas partout défini entre les registres d'horloge de deux processus voisins ?

Les réponses aux deux premières questions sont très simples et font apparaître des objets nouveaux : les *systèmes d'incrémentation bornée* et des *ordres locaux* définis sur ceux-ci. Deux valeurs quelconques de ces systèmes d'incrémentation bornée ne sont pas nécessairement comparables, d'où la restriction du travail sur les ensembles d'états Γ_M avec $K > 2M$.
Nous avons montré qu'il peut y avoir de nombreuses pathologies dans Γ_M. Nous avons introduit la notion de *caractéristique cyclomatique* du graphe, notée : C_G. Nous avons montré que si $K > MC_G$, alors les pathologies disparaissent, et on peut construire un *ordre global* sur le système grâce à la notion de *retard*, qui est alors *intrinsèque*, c'est-à-dire indépendante du chemin choisi pour le calculer. Nous avons répondu aux questions trois et quatre.
La convergence $\Gamma \triangleright \Gamma_M$ n'est pas traitée dans ce chapitre. Elle le sera dans le chapitre 3.
En un certain sens, ce chapitre est clos pour les quatre premières questions, il culmine avec le théorème de relèvement qui dit, de manière un peu formelle mais essentielle, que l'intuition que nous avons dans le cas des registres à valeurs dans $\mathbb{Z}$, est celle que nous devons avoir dans le cas des registres à valeurs dans le groupe modulaire $\mathbb{Z}_K$, pourvu que l'on démarre bien, c'est-à-dire dans Γ_M^0, ou avec K assez grand, ce qui garantit $\Gamma_M = \Gamma_M^0$.
L'outil technique, pour gérer cette intuition, est le relèvement. Nous retrouverons cette notion de manière essentielle dans les chapitres 7 et 8.
Etre dans Γ_M^0 est difficilement contrôlable quand on fait de l'auto-stabilisation ; en revanche, " K assez grand " ne veut pas dire "monstrueux". Assez grand veut dire : K plus grand que $M.C_G$, où C_G est la caractéristique cyclomatique du graphe. Comme $C_G \leq \min(2d, n)$, si on prend $M = 1$, on sera donc en $O(\ln(n))$ en occupation mémoire par processus.
Si le registre r est défini sur 4 mots de 2 octets, alors $K = 2^{64} > 10^{19}$, ce qui est largement suffisant, même si r est défini sur 2 mots de 2 octets, alors $K = 2^{32} > 10^9$, cela peut être encore bien suffisant.
Une application du théorème de relèvement est le théorème de convergence $\Gamma_M^0 \triangleright \Gamma_1^0$. Sa preuve tient en une ligne : grâce au théorème de relèvement, il suffit d'appliquer le théorème de convergence démontré dans le cas où les registres sont à valeurs dans $\mathbb{Z}$.

Enfin, nous avons donné un exemple d'application : un algorithme de calcul d'arbre couvrant en largeur, dans un environnement sans faute, optimal en espace et en temps de construction. Il est meilleur que tout ce qui existait dans la littérature, où demeurait une sorte d'incompatibilité entre l'optimalité en espace et l'optimalité en temps de construction. Notre algorithme montre qu'il n'est pas nécessaire de faire un compromis entre ces deux objectifs d'optimalité. Nous donnons une version de cet algorithme dans le modèle à passage de messages. Cet exemple montre aussi combien le travail dans le modèle à états peut être fécond, car il permet d'abstraire les problèmes de communication.

Chapitre 3

Tolérer les fautes

> Nous n'existons pas dans la majorité de ces temps ; dans quelques-uns vous existez et moi pas ; dans d'autres, moi, et pas vous ; dans d'autres encore, tous les deux. Dans celui-ci, que m'accorde un hasard favorable, vous êtes arrivé chez moi ; dans un autre, en traversant le jardin, vous m'avez trouvé mort ; dans un autre, je dis ces mêmes paroles, mais je suis une erreur, un fantôme.
>
> Jorge Luis Borges, *Le jardin aux sentiers qui bifurquent* (1941) [Bor93].

Sommaire

1 Position du problème

1.1 Préliminaires

Dans le chapitre précédent, nous avons d'abord étudié l'unisson défini avec une horloge non bornée à valeurs dans $\mathbb{Z}$. Nous avons montré que le protocole ainsi défini est convergent vers Γ_1. Nous avons ensuite étudié les incrémentations de phase définis avec une horloge bornée ; essentiellement, une horloge à valeurs dans $\mathbb{Z}_k$, avec K entier strictement supérieur à 2. Nous avons défini une notion de comparaison locale des registres de deux processus voisins. Cette relation est définie pour tout entier $M > 0$ satisfaisant $K > 2M$. En fait, la relation peut aussi être définie pour $M = 0$, mais cela ne sera utile que dans le cas synchrone (cf. chapitre 4). Dans ce chapitre, on suppose donc que $M > 0$ et $K > 2M$.
Γ est l'ensemble des états du système, Γ_M est l'ensemble des états tels que deux processus voisins sont localement comparables, c'est-à-dire que leurs registres ont un *écart* d'au plus M. A chaque état de Γ_M on peut associer un graphe d'interblocage. Ce graphe peut contenir des cycles, ou pire, être acyclique mais conduire à des cycles lors de l'exécution du protocole. Nous en avons déduit dans le chapitre précédent que l'approche par l'étude des graphes d'interblocage n'était pas satisfaisante. C'est pourquoi nous avons introduit la notion de *retard* et la notion de *résiduel* (voir définition page 66).
Dans notre travail, la notion de résiduel non nul remplace la notion de cycle (théorème 16, page 66). S'il y a un cycle d'interblocage alors le résiduel sur ce cycle est non nul, mais la réciproque est fausse. Par contre, si un résiduel est non nul, il le restera durant toute l'exécution du protocol. Le résiduel est un invariant global pour toute exécution d'une incrémentation de phase dans Γ_M. Comme la notion de cycle, la notion de résiduel n'est pas une notion locale. De même qu'il fallait briser les cycles dans d'autres théories, il faut ici briser les résiduels non nuls.
Nous avons noté Γ_M^0 l'ensemble des configurations de Γ_M où tous les résiduels sont nuls. Nous avons donné une condition suffisante pour que $\Gamma_M = \Gamma_M^0$, à savoir que $K > MC_G$, où C_G est la caractéristique cyclomatique du graphe G définissant le réseau (voir le théorème 20, page 71).

Quand cette condition est vérifiée, il n'y a pas lieu de "briser les résiduels non nuls", puisqu'il n'y en a pas. Le prix à payer est une relative complexité en espace puisque $log(MC_G)$ est dans $O(\log(Md))$ car $C_G \leq 2d$. Si M est fixé , par exemple à 2, alors la complexité en espace est dans $O(log(d))$.

1.2 Fautes et tolérance aux fautes

1.2.1 Les fautes

La notion de faute est définie page 25. Dans ce chapitre, nous formulons l'hypothèse que les seules fautes possibles sont les suivantes :

1. **Faute transitoire** : Le programme d'un processus n'est pas corruptible. Les valeurs des registres d'un processus peuvent être modifiées (faute transitoire), mais à partir d'un moment, il n'y a plus de nouvelle faute transitoire (voir chapitre 1, page 25).
2. **Changement de topologie** : Remarquons que dans un système asynchrone, si le crash d'un processus est indétectable, ou si sa détection peut prendre un temps non borné, alors il est impossible de réinitialiser le réseau [FLP85]. Donc, dans l'esprit du modèle à états, nous faisons l'hypothèse qu'instantanément, après un changement de topologie (dû à l'entrée ou la sortie du réseau d'un processus, ou dû à un crashs d'un processus par exemple), chaque processus actif connaît l'ensemble de ses voisins actifs. Ce qui revient à dire que chaque nœud du réseau est muni d'un détecteur de pannes parfait (cf. paragraphe 1.3.2). C'est pourquoi, plutôt que de parler de panne et de réparation, nous parlerons de changement de topologie. Ainsi nous formulons l'hypothèse que la topologie du réseau peut être modifiée.

1.3 Stabilisation d'une incrémentation de phase

L'idée de stabiliser un protocole par rapport à des changements dynamiques d'un réseau n'est pas une idée nouvelle. [AAG87], par exemple, aborde le problème de savoir comment adapter un protocole défini pour un réseau dont la topologie est invariante sur un réseau dont la topologie change dynamiquement ; les auteurs développent un transformateur simple, appelé procédure de réinitialisation (reset procedure) qui permet cette adaptation. L'idée de rendre un algorithme tolérant aux faute transitoires n'est pas non plus nouvelle [Dij74]. L'idée de combiner les deux préoccupations a été abordé notament par [KA97]. C'est dans cette perspective que notre travail se situe.

1.3.1 Détection locale d'inconsistance

Pour un problème donné, [AKY90] suggère qu'il arrive parfois que l'inconsistance globale d'un état du réseau soit localement détectable, c'est-à-dire détectable par contrôle par chacun des processus de l'état de ses voisins. Cette idée est développée et formalisée dans [APSV91, Var93] sous le nom de *vérification locale* (local checking). Une fois l'inconsistance du réseau détectée, l'idée est de corriger les fautes par une réinitialisation distribuée du

système (reset). Pour ce faire [AKY90] et [AG94] construisent et utilisent un arbre couvrant le réseau.

1.3.2 Inconsistance et incrémentation de phase

Dans le cas d'une incrémentation de phase, si $K > MC_G$, alors on peut localement détecter si le système n'est pas dans une configuration globale correcte (l'état du système n'est pas dans Γ_M). Dans ces conditions, la correction de l'incrémentation de phase est localement contrôlable [Var93] : si on ne connaît qu'un majorant D du diamètre, alors si $K > 2MD$ le réseau est en mesure de détecter localement toute faute dans les registres due soit à une perturbation transitoire du système, soit à un changement de topologie : perte d'un ou plusieurs nœuds, d'une ou plusieurs liaisons, ajout de nœuds ou de liens. La seule restriction est que le diamètre du réseau doit rester inférieur à D. Si une telle perturbation intervient et si elle met le système dans un état incorrect, cette incorrection sera localement repérée. Il restera à organiser une réinitialisation du système à partir de cette information locale. Deux stratégies sont possibles :

1. lancer une réinitialisation globale à partir d'un arbre couvrant enraciné ;
2. lancer un "reset" local à partir de chaque nœud qui a repéré la faute et étudier à quelles conditions cette réinitialisation locale provoquera une réinitialisation du système vers un état correct.

Une question intéressante serait de savoir s'il est possible de corriger localement un réseau sans propagation des corrections sur l'ensemble du réseau, dans le cas où il n'y a qu'un petit nombre de fautes localisées (locally correctable network)[Var93]. Cette troisième approche est possible est appelée : l' endiguement de fautes [GGHP96]. Nous n'aborderons pas cette question ici.

1.3.3 La réinitialisation d'un système

La notion de réinitialisation a été introduite par [Fin79] comme méthode générale permettant à un protocole défini dans un réseau dont la topologie est invariante, de se définir dans un réseau dont la topologie change dynamiquement. Le vocabulaire de cette partie suit celui défini dans [KA98b]. Dans cet article, un protocole avec réinitialisation est constitué de deux modules définis sur chaque processus : le module d'application et le module de réinitialisation. C'est le module d'application qui lance le module de réinitialisation vers un état donné γ_0 du système. Une fois lancée la réinitialisation, les modules de réinitialisation doivent amener le système, en un nombre fini d'étapes, vers un état qui est accessible à partir de γ_0 par le module d'application ; ils doivent ensuite prévenir chaque processus que la réinitialisation est terminée. En d'autres termes plus formels, une réinitialisation doit satisfaire aux spécifications suivantes :

Spécification 4 [Réinitialisation [KA98b]]

1. **sûreté** : toute opération de réinitialisation ne doit pas se terminer prématurément, c'est-à-dire que l'état γ atteint à la fin de la réinitialisation est accessible par le

module d'application à partir de γ_0 ;

2. **terminaison** : toute opération de réinitialisation doit finir au bout d'un nombre fini d'étapes, et informer chaque processus de la terminaison.

On peut remarquer qu'a priori la définition n'exige pas que chaque processus mette en œuvre son module de réinitialisation. Si une faute est localement corrigible, une réinitialisation globale n'est pas nécessaire. La définition ne demande pas non plus que le système reparte dans l'état γ_0, mais seulement qu'il reparte dans un état accessible à partir de γ_0.
Pour l'incrémentation de phase, nous prendrons pour γ_0 l'état où toutes les horloges sont à 0. L'état γ peut être n'importe quel état dans Γ_M^0, en fait dans Γ_M, puisque $K > MC_G$.

1.4 Correction locale ou globale ?

A strictement parler, une réinitialisation est toujours globale, au sens où il faut atteindre un état qui est globalement correct. La méthode employée est en général une méthode "brutale" qui consiste à introduire une structure, souvent un arbre couvrant, et à utiliser cette structure pour réinitialiser le système globalement. Essentiellement il s'agit, par un moyen ou un autre, de construire une barrière de synchronisation forte, puis de faire une diffusion. Par exemple [AG94] donne une procédure générale de réinitialisation : cette procédure commence par l'élection d'une racine, puis construit un arbre couvrant, puis fait une diffusion avec un retour d'information sur cet arbre. Il va sans dire que le réseau est avec identités. L'unisson de [Dol00] utilise une deuxième horloge d'ordre "assez grand" de manière que, une fois que cette horloge a fait un "tour complet", on soit assuré que la procédure de réinitialisation ait inondé l'ensemble du réseau.
Nous appellerons de telles méthodes : des méthodes globales de réinitialisation. Nous proposons en annexe (cf annexe A) un algorithme d'unisson appelé SS_RU , qui utilise cette technique de réinitialisation. Il construit un arbre couvrant en largeur d'abord et fait une opération de diffusion sur cet arbre. Il est trés efficace en temps de convergence vers Γ_1 puisqu'il converge en $O(d)$ rondes. Cela dit, cet algorithme d'unisson est construit grâce à une composition équitable de deux protocoles auto-stabilisants, la composition se faisant sous un démon faiblement équitable et dans un réseau avec identités.
Il se trouve que dans les applications de synchronisation que nous étudions, la détermination de terminaison globale n'est pas utile. Dans ce chapitre, notre ambition est de mettre au point un protocole de réinitialisation efficace n'utilisant pas de structure globale et fonctionnant dans un environnement anonyme sous un démon asynchrone. Nous voulons ne faire que des corrections locales, sans utiliser de structure globale.

1.5 Protocole d'incrémentation de phase avec corrections locales

1.5.1 La question

L'idée de départ est la suivante : lorsqu'une faute est localement détectée par un processus, celui-ci lance une correction locale. Nous considérons le réseau comme anonyme. La ques-

tion est de définir une correction locale qui aboutisse à une réelle réinitialisation globale, c'est-à-dire que :

1. l'opération de correction locale ne doit par se terminer prématurément
2. Le système doit converger vers un état de Γ_M
3. Chaque processus doit savoir quand il peut localement reprendre ses incrémentations normales.

1.5.2 La réponse de Couvreur, Francez et Gouda

Avant notre travail, seul un protocole parvenait à organiser une telle réinitialisation par corrections locales : le protocole de Couvreur, Francez et Gouda [CFG92]. Ce papier est écrit dans le modèle avec atomicité en lecture-écriture. Dans notre modèle, il est défini par l'algorithme 6.

Algorithme 6 (SS_U_CFG) : protocole de Couvreur, Francez et Gouda

Constante :

$\mathcal{N}_p$: l'ensemble des voisins de p ;

Variable :

$p.r \in \mathcal{X}$;

Fonctions booléennes :

$Correct_p(q) \equiv d(p.r, q.r) \leq M$;
$AllCorrect_p \equiv \forall q \in \mathcal{N}_p : Correct_p(q)$;
$NormalStep_p \equiv \forall q \in \mathcal{N}_p : p.r \leq_l q.r$;

Actions :

NA : $NormalStep_p \longrightarrow$ << code de l'application >> ; $p.r := \varphi(p.r)$;
RA : $\neg AllCorrect_p \longrightarrow p.r := 0$ (correction locale) ;

Chaque processus, quand il perçoit une faute locale, met la valeur 0 dans le registre de son horloge. On dit qu'il se corrige ou se réinitialise (correction locale). C'est la seule action de correction. La question devient : à quelle condition cette action locale de réinitialisation garantit que l'opération de réinitialisation ne termine pas prématurément et que le système converge vers un état de Γ_M ?
[CFG92] propose une condition suffisante très simple :

$$M = n \text{ et } K > n^2$$

L'article énonce, en utilisant un lemme sans démonstration, que, dans ces conditions, il n'y a qu'un nombre fini de réinitialisations locales, pas d'interblocage, et que, une fois stabilisé dans Γ_n, le système converge vers Γ_1.
Remarquons que l'étude du chapitre 2 nous permet de dire immédiatement des choses importantes :
puisque $C_G \leq n$, on peut dire que $n^2 \geq nC_G$, et puisque $K > n^2$, on peut énoncer :

1. par le théorème 20 : $\Gamma_n = \Gamma_n^0$, et donc il n'y a pas d'interblocage ;

2. grâce au théorème 29, la convergence $\Gamma_n \rhd \Gamma_1$ est assurée.

Si on veut comprendre l'algorithme de [CFG92], il ne reste plus qu'à étudier le lemme sans preuve établissant l'extinction des actions de réinitialisation locale. [CFG92] ne donne aucune analyse de complexité en temps de réinitialisation (temps de convergence vers Γ_1). Nous montrerons à la fin de ce chapitre que le temps de réinitialisation de ce protocole est dans $O(nd)$ rondes, ce qui est beaucoup.
En tout cas, le résultat de [CFG92] peut s'énoncer par le théorème suivant :

> **Théorème 47** Si N est un majorant du nombre de processus du réseau et si $K > N^2$ alors il existe un protocole d'incrémentation de phase auto-stabilisant, qui est tolérant aux changements de topologie, et n'effectue que des corrections locales.

1.6 Spécification de l'unisson

Dans l'introduction de ce mémoire, suivant Ted Herman [Her01], nous avons défini provisoirement l'unisson comme une barrière de synchronisation faible auto-stabilisante (voir page 31). Donnons pour commencer une spécification formelle de la barrière de synchronisation faible :

> **Spécification 5** [Barrière de synchronisation faible]
>
> 1. **sûreté** :
>
>   (a) (initialisation) initialement, tous les processus ont effectué correctement la phase 0.
>
>   (b) (synchronisation) un processus peut effectuer *l'action de l'application* de la phase $i+1$ *mod* K puis incrémenter son horloge seulement si ses voisins ont fini d'effectuer *l'action de l'application* de la phase i ;
>
> 2. **vivacité** :
>
>   tout processus p exécute *l'action de l'application* puis incrémente son horloge une infinité de fois ;

Nous proposons ici une définition plus contraignante de l'unisson. Il va de soi qu'un unisson à notre sens doit être un unisson au sens de Ted Herman. Commençons par motiver cette nouvelle définition.
L'algorithme 6 est un algorithme d'unisson au sens de Ted Herman, mais il est facile de voir que l'algorithme 6 peut violer la spécification de synchronisation (cf. page 1.1) :

1. (Synchronisation) un processus p peut incrémenter son horloge $p.r$ si et seulement si la valeur de $p.r$ est inférieure ou égale aux valeurs des horloges des processus voisins de p

Cette violation ne peut se faire que pendant la phase de convergence vers Γ_1. Il peut arriver que pour l'utilisateur, le seul phénomène important soit que l'exécution du code de l'application ne viole pas la spécification de synchronisation que nous venons de proposer. C'est le cas, par exemple, dans les applications de l'unisson proposées dans les chapitres 6

et 8. Ces remarques nous amènent à proposer une définition plus contraignante de la spécification de la synchronisation de l'unisson :

1. (Synchronisation) un processus peut incrémenter son horloge pour exécuter le code de l'application si et seulement si la valeur de celle-ci est localement inférieure ou égale à la valeur de celles de ses voisins.

Dans ces conditions, l'algorithme 6 est instantanément stabilisant pour la nouvelle spécification de synchronisation. Ce qui est une information plus forte. Ajoutons la convergence vers Γ_M, on obtient la nouvelle spécification de l'unisson asynchrone :
Soit K un entier strictement plus grand que 1, on suppose que chaque processus p maintient une horloge d'ordre K, définie par le registre $p.r \in \chi = \{-\alpha, ..., 0, 1, ..., K-1\}$. Un *unisson asynchrone*, ou encore un *unisson* est défini par :

Spécification 6 [Unisson]

1. **sûreté** : Pour tout $i_in \{0, 1, ..., K-1\}$:
   (synchronisation) un processus peut incrémenter son horloge pour effectuer *l'action de l'application* de la phase $i+1$ *mod* K seulement si la valeur i de son horloge est localement inférieure ou égale aux valeurs des horloges de ses voisins ;
2. **vivacité** :
   (a) (convergence) $\Gamma \triangleright \Gamma_1$;
   (b) dans Γ_1 tout processus p ne peut qu'exécuter *l'action de l'application* puis incrémenter son horloge et cela une infinité de fois.

La condition de sûreté de l'unisson est exprimée un peu différemment de la condition de sûreté de la barrières de synchronisation faible. La raison vient du fait qu'il n'y a pas d'initialisation et donc au démarrage il est possible qu'un processus fasse l'*action de l'application* sans qu'aucun voisin n'ai fait une seule fois l'*action de l'application*. Remarquons que nous n'avons pas mis comme condition de sûreté : Γ_1 est clos, puisque cette condition est, par définition, une conséquence de la condition de vivacité : $\Gamma \triangleright \Gamma_1$. Remarquons aussi que la condition de synchronisation impose que $i \in \{0, 1, ..., K-1\}$.
Avec cette nouvelle spécification, tout unisson est une barrières de synchronisation auto-stabilisante.

2 Convergence vers Γ_M par corrections locales

2.1 Introduction

Nous nous plaçons dans un cadre plus général que celui de [CFG92] dans le sens suivant : on considère un système d'incrémentation borné, voir la section 3 du chapitre 2. Rappelons que ces systèmes sont schématisés par la "cerise" représentée par la figure 3.1.
Dans cette section, $M > 0$, $K > 2M$, $\alpha \geq 0$ et $\chi = \{-\alpha, ..., 0, ..., K-1\}$. On y considèrera le système d'incrémentation (χ, φ) dont la signature est (α, K). Nous reprenons aussi le vocabulaire présenté dans la définition 3.1 du chapitre 2.

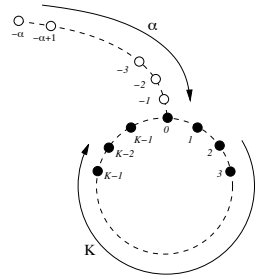


FIG. 3.1 – Système d'incrémentation fini $(\mathcal{X}, \varphi)$.

2.2 Définition du protocole

L'idée est de voir comment utiliser $tail_\varphi$, c'est-à-dire la queue de la cerise, pour réinitialiser le système. Pour que l'unisson soit dans un état correct, il est nécessaire que le contenu de l'horloge de chacun des processus soit dans $ring_\varphi$, c'est à dire positif ou nul. Le prédicat $AllCorrect_p$ signifie que le processus p est localement correct avec ses voisins ; c'est le prédicat de base qui spécifie localement si le réseau est correct. Bien entendu, si on veut que le réseau soit localement contrôlable, on doit avoir $K > MC_G$, voir sous-section 1.3.2. Si un processus détecte localement une inconsistence, alors il se met en retrait du réseau en mettant dans son registre d'horloge la valeur $-\alpha$(réinitialisation locale). L'idée, ensuite, est d'introduire une sorte de temporisation avant que le processus ne rentre à nouveau comme processus correct dans le réseau. Cette temporisation est définie par la remontée dans la queue du système d'incrémentation, c'est-à-dire dans $tail_\varphi = \{-\alpha, .., 0\}$.
La correction locale est donc définie par le prédicat $AllCorrect_p$, lui-même construit à partir des prédicats C_p et $Correct_p$:

$$\begin{array}{lcl} C_p(x,y) & \equiv & x.r \in ring_\varphi \wedge y.r \in ring_\varphi \\ Correct_p(q) & \equiv & C_p(p,q) \wedge d(p.r, q.r) \leq M\,; \\ AllCorrect_p & \equiv & \forall q \in \mathcal{N}_p :\ Correct_p(q)\,; \end{array}$$

Il y a plusieurs modes possibles de remontée dans la queue (voir chapitre 4). Ici nous faisons le choix de la stratégie suivante : un processus peut remonter d'une unité dans $tail_\varphi$ si tous ses voisins sont dans $tail_\varphi$, et en avance au sens large par rapport à lui. Ce qui donne le prédicat :

$$ConvergenceStep_p \equiv p.r \in tail_\varphi^* \wedge (\forall q \in \mathcal{N}_p :\ (q.r \in tail_\varphi) \wedge (p.r \leq_{tail_\varphi} q.r))$$

Remarquons que dans ce prédicat le cas $p.r = 0$ est traité à part, on considère alors que le processus est dans le réseau, et la condition d'incrémentation est la condition de l'incrémentation de phase. En fait, dire qu'il est rentré dans le réseau n'est pas tout à fait juste, il sera rentré quand tous ses voisins seront aussi à 0. C'est pourquoi l'action de réinitialisation locale est déclenchée par le prédicat :

$$ResetInit_p \quad \equiv \quad \neg AllCorrect_p \wedge (p.r \notin tail_\varphi)\,;$$

Notre protocole comporte trois actions : l'action normale (NA) d'incrémentation de phase avec exécution du code de l'application . L'action de réintialisation (RA) et l'action (CA) de remontée dans $tail_\varphi$.
Le protocole est défini sur Γ, ensemble des états du système, par l'algorithme 7. Il est facile de voir que sa restriction à Γ_M est le protocole défini par l'algorithme 2 du chapitre 2.

Algorithme 7 $GAU(K, \alpha, M)$: protocole d'unisson sur réseau anonyme pour le processus p

Constante :

$\mathcal{N}_p$: l'ensemble des voisins de p ;

Variable :

$p.r \in \mathcal{X}$;

Fonctions booléennes :

$C_p(x,y)$	$\equiv$	$x.r \in ring_\varphi \wedge y.r \in ring_\varphi$
$Correct_p(q)$	$\equiv$	$C_p(x,y) \wedge d(p.r, q.r) \leq M$;
$AllCorrect_p$	$\equiv$	$\forall q \in \mathcal{N}_p :\ Correct_p(q)$;
$NormalStep_p$	$\equiv$	$\forall q \in \mathcal{N}_p :\ p.r \leq_l q.r$;
$ConvergenceStep_p$	$\equiv$	$p.r \in tail^*_\varphi \wedge (\forall q \in \mathcal{N}_p :\ (q.r \in tail_\varphi) \wedge (p.r \leq_{tail_\varphi} q.r))$;
$ResetInit_p$	$\equiv$	$\neg AllCorrect_p \wedge (p.r \notin tail_\varphi)$;

Actions :

NA :	$NormalStep_p$	$\longrightarrow$	<< code de l'application >> ; $p.r := \varphi(p.r)$;
CA :	$ConvergenceStep_p$	$\longrightarrow$	$p.r := \varphi(p.r)$;
RA :	$ResetInit_p$	$\longrightarrow$	$p.r := -\alpha$ (réinitialisation locale) ;

On prend $K > MC_G$ afin d'assurer que la cohérence du réseau soit localement vérifiable. Le problème, maintenant, est de trouver une condition suffisante pour que $\Gamma \triangleright \Gamma_M$.

1. Si $M = n$, $\alpha = 0$ et $K > n^2$, on retrouve l'algorithme de Couvreur, Gouda et Francez [CFG92].
2. Si $M = 1$, on retrouve l'algorithme $SSAU$ de [BPV04].

2.3 Correction du protocole

Nous démontrerons en premier lieu qu'il ne peut y avoir d'interblocage en dehors de Γ_M. Ensuite nous introduirons la notion de DAG de réinitialisation locale, plus simplement DAG de réinitialisation. C'est une structure qui contient toute l'information sur la propagation des réinitialisations dans le réseau. L'extinction des réinitialisations assure la convergence vers Γ_M. Notre objectif est donc de trouver une condition qui assure que le DAG de réinitialisation est fini. Un fait très intéressant est le fait que les réinitialisations se propagent le long des trous ; les trous sont les cycles sans corde du graphe G. Cela explique pourquoi les trous jouent un rôle important dans l'étude de la minimisation de la longueur de la queue $tail_\varphi$. Notons T_G la longueur du plus grand trou du réseau G ; dans le cas où G est acyclique alors on pose $T_G = 2$. Nous prouverons les deux résultats suivants :

1. Si $\alpha = 0$, alors le protocole défini par l'algorithme 7 converge vers Γ_M si et seulement si $M \geq T_G - 1$, (Par exemple, on peut prendre $M = 1$ sur un arbre).

2. Si $\alpha > 0$, alors le protocole est convergent vers Γ_M si et seulement si $\alpha \geq T_G - 2$.

Nous prouverons que, respectivement, les inégalités $M \geq T_G - 1$ et $\alpha \geq T_G - 2$ sont les meilleures possibles pour assurer la convergence de $GAU(K, \alpha, M)$. Bien entendu, dans les deux cas, la constante K doit satisfaire la contrainte $K > MC_G$.

2.3.1 Vivacité en dehors de Γ_M

Si $p.r$ est dans $tail^*_\varphi$, on dit que p est en *phase de correction locale*.

Théorème 48 Toute configuration interbloquée est dans Γ_M.

Preuve : Si $n = 1$ alors le théorème est clair puisqu'il ne peut y avoir d'interblocage. Supposons que $n > 1$, et soit γ une configuration interbloquée. Deux cas sont possibles :

1. L'ensemble des processus en phase de correction locale est non vide. Soit p un des processus en phase de correction locale qui a la plus petite valeur dans sont registre. Pour tout $q \in \mathcal{N}_p$, l'interblocage impose $q.r \in tail_\varphi$ et, par hypothèse de minimalité du registre de p, $q.r \geq_{tail_\varphi} p.r$. On en déduit que $ConvergenceStep_p$ est vrai. Ainsi p peut effectuer l'action CA, la configuration n'est donc pas interbloquée.
2. Supposons qu'aucun processus n'est en phase de correction locale, ainsi tous les registres sont à valeur dans $ring_\varphi$. Supposons qu'il existe un processus p tel que $AllCorrect_p$ est faux. Alors il existe un processus $q \in \mathcal{N}_p$ tel que $\neg Correct_p(q)$, et par symétrie $\neg Correct_q(p)$ est aussi vrai. Cela signifie, puisque l'action RA ne peut pas être exécutée, que $p.r = \bar{0}$ et $q.r = \bar{0}$. Ainsi $AllCorrect_p(q)$ est vrai, ce qui contredit l'hypothèse.

Donc on vient de montrer que $\gamma \in \Gamma_M$.

□

Nous savons que si $K > MC_G$ il n'y a pas d'interblocage dans Γ_M (voir théorème 21, chapitre 2). On obtient comme corollaire le théorème suivant :

Théorème 49 Si $K > MC_G$ alors le protocole est sans interblocage.

On ne peut pas dire que le protocole soit sans famine, car il est possible que les réinitialisations se propagent indéfiniment, bloquant certains processus. Nous n'avons pas encore donné de condition sur α pour que le protocole soit convergent. Cette condition assurera alors la convergence vers Γ_M en un temps fini.

2.3.2 Les trous et le DAG de réinitialisation

Nous commençons par définir la notion de *réinitialisation locale*, ou plus simplement de *réinitialisation*. Ensuite nous étudierons la propagation de ces objets.

Définition 29 [Réinitialisation] Soit $e = \gamma_0\gamma_1 \dots \gamma_k \dots$ une exécution maximale de l' algorithme GAU. Une *réinitialisation locale*, ou *réinitialisation*, est un couple (p, t) où p est un processus et $t > 0$ tels que :

1. p exécute l'action RA lors de la transition $\gamma_{t-1} \mapsto \gamma_t$

Il est clair que si (p,t) est une réinitialisation locale alors $p.r \neq \alpha$ dans γ_{t-1} et $p.r = -\alpha$ dans γ_t. On dit que p se réinitialise à $-\alpha$ à la date t.

Définition 30 [Relation de génération] Soient (p_1,t_1) et (p_2,t_2), deux réinitialisations locales. On dit que (p_1,t_1) *génère* (p_2,t_2), si et seulement si les deux conditions suivantes sont vérifiées :
1. $t_1 < t_2$
2. $p_2 \in \mathcal{N}_{p_1}$ et $\forall t \in [t_1, t_2-1]$:
   si $\alpha = 0$: $d(p_1.r, p_2.r) > M$
   si $\alpha > 0$: $p_2.r \notin tail_\varphi$

On note cette relation binaire sur l'ensemble des réinitialisations :

$$(p_1,t_1) \overset{r}{\rightsquigarrow} (p_2,t_2)$$

Avec les notations de la définition, la relation $(p_1,t_1) \overset{r}{\rightsquigarrow} (p_2,t_2)$ signifie que $\forall t \in [t_1, t_2-1]$ le prédicat $\neg Correct_{p_2}(p_1)$ est satisfait et p_2 se réinitialise à cause de p_1.
Puisque $(p_1,t_1) \overset{r}{\rightsquigarrow} (p_2,t_2)$ implique $t_1 < t_2$, la relation $\overset{r}{\rightsquigarrow}$ définit un graphe orienté acyclique (Directed Acyclic Graph ou DAG), appelé *DAG de réinitialisation*. Si la réinitialisation (p_1,t_1) n'est pas générée par une autre réinitialisation, on dit que (p_1,t_1) est une *réinitialisation initiale* . Nous appellerons *hélice de réinitialisation* tout chemin $(p_0,t_0),(p_1,t_1)\ldots(p_k,t_k)$ sur le DAG de réinitialisation tel que sa projection $p_0p_1\ldots p_k$ sur G est un cycle. Remarquons qu'à priori la projection est une "marche" et non un chemein, nous verrons qu'en fait l'absence de bégaiement implique que la projection est bien un chemin non nécessairement simple.

Lemme 50 [Décomposition] Si $p_0p_1\ldots p_k$ est un cycle, alors il existe une paire $i,j \in \{0,\ldots,k\}$ telle que $i < j-1$ *et* $p_ip_{i+1}\ldots p_jp_i$ est un trou.

Lemme 51 Dans toute exécution, un processus p étant donné, il existe au plus une date t telle que (p,t) soit une réinitialisation initiale, et c'est la première réinitialisation locale de p (si elle existe).

Preuve : Supposons que le processus p ait une réinitialisation initiale t_2 après une autre réinitialisation t_1. On a donc $t_1 < t_2$ et on fait l'hypothèse qu'il n'y ait pas d'autre réinitialisation de p dans l'intervalle $[t_1+1, t_2-1]$. Considérons $\gamma_{t_1}\gamma_{t_{1+1}}\ldots\gamma_{t_2}$, l'exécution entre les dates t_1 et t_2. Puisque (p,t_1) et (p,t_2) sont des réinitialisations, $p.r = -\alpha$ dans γ_{t_1} et $p.r \notin tail_\varphi$ dans γ_{t_2-1}. Donc l'action NA est exécutée au moins une fois par le processus p. Donc l'ensemble $\{t \in [t_1, t_2-1] :\ Correct_p \text{ dans } \gamma_t\}$ n'est pas vide et est majoré par t_2. Cet ensemble contient donc un plus grand élément $t_3 < t_2 - 1$. A la date t_3+1, $Correct_p$ est faux. A cause de l'action NA à la date t_3, il existe au moins un voisin q de p tel que $\neg Correct_p(q)$ à la date t_3+1, ce qui n'est possible que si q exécute l'action RA (une réinitialisation locale) à la date t_3. Donc, comme il n'y a pas d'initialisation de p entre t_3 et t_2-1, on obtient $(q,t_{3+1}) \overset{r}{\rightsquigarrow} (p,t_2)$, ce qui est contraire à l'hypothèse.

□

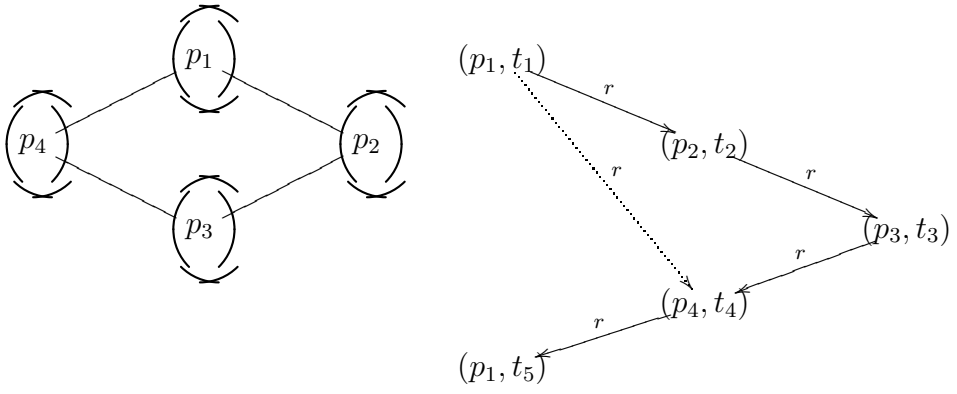
Définition 31 On appelle *bégaiement* tout chemin sur le DAG de réinitialisation de la forme suivante $(p_1, t_1)(p_2, t_2)(p_1, t_3)$.

Lemme 52 Tous les DAG de réinitialisation sont sans bégaiement.

Preuve : Si $(p_1, t_1) \overset{r}{\rightsquigarrow} (p_2, t_2)$ alors si $\alpha > 0 : \forall t \in [t_1, t_2[\ \ p_2.r \notin tail_\varphi$ (respectivement si $\alpha = 0 : d(r_{p_2}, r_{p_1}) > M$) alors $\forall t \in [t_1, t_2]\ \ p_1.r = \alpha$. On en déduit que $(p_2, t_2) \overset{r}{\rightsquigarrow} (p_1, t_3)$ est impossible.

□

Définition 32 [Trou-transitivité] Un DAG de réinitialisation est trou-transitif si et seulement si : $(p_1, t_1) \overset{r}{\rightsquigarrow} (p_2, t_2) \overset{r}{\rightsquigarrow} \ldots \overset{r}{\rightsquigarrow} (p_i, t_i)$ et $p_1 p_2 \ldots p_i p_1$ est un trou, entraine $(p_1, t_1) \overset{r}{\rightsquigarrow} (p_i, t_i)$.



$(a)Un\ trou$ $(b)Une\ hélice$

FIG. 3.2 – Trou-transitivité et bégaiement $(p_1, t_1)(p_4, t_4)(p_1, t_5)$

Théorème 53 Tout DAG de réinitialisation trou-transitif est fini. En particulier, si G est un graphe acyclique, alors tout DAG de réinitialisation sur G est fini.

Preuve : S'il existe une hélice dans un DAG de réinitialisation alors il existe une hélice le long d'un trou. En effet, prenons une hélice de longueur minimale et utilisons le lemme 50 de décomposition : il implique que cette hélice est construite sur un trou, sinon on pourrait en extraire une sous-hélice, ce qui contredirait la minimalité de sa longueur. Par la propriété de la trou-transitivité, le DAG de réinitialisation contient alors un bégaiement, ce qui est impossible par le lemme 52. Ainsi tout DAG de réinitialisation qui est trou-transitif est sans hélice. Donc, comme, par le lemme 51, le DAG de réinitialisation ne contient qu'un nombre fini de réinitialisations initiales (au plus une par processus), on en déduit que le DAG de réinitialisation est fini. Voir la figure 3.2 qui présente un cas qui est impossible dans le cadre de ce théorème.
Si G est acyclique, le DAG de réinitialisation est sans hélice, il est donc fini.

□

2.3.3 Convergence vers Γ_M

Théorème 54 Si le DAG de réinitialisation est fini, alors le système converge vers Γ_M.

Preuve : Soit e une exécution maximale. On distingue 3 cas :

1. Si e est finie alors, par le théorème 48, la dernière configuration est dans Γ_M, ce qui prouve le théorème. Remarquons que le cas fini est possible si le système converge vers un interblocage ; dans ce cas, bien sûr, $K \leq MC_G$.
2. Supposons que e est infinie et qu'il existe un ensemble non vide R de processus qui restent en phase de correction locale pour toujours. Soit e' un suffixe de e sans réinitialisation locale. Cela signifie qu'il y a un suffixe e'' de e' sans incrémentation dans la phase de correction locale, puisque le nombre d'incrémentations dans la phase de correction est fini. Cela veut dire que les processus à distance 1 de l'ensemble non vide R sont bloqués. Les processus à distance 2 ne peuvent faire qu'au plus 2 incrémentations, et ainsi de suite. Par récurrence, les processus à distance k ne peuvent pas faire plus de $2(k-1)$ incrémentations. Puisque G est fini, e' ne peut pas être infini, donc R est vide. Ce cas est donc impossible.
3. On suppose maintenant que e est infinie et qu'il n'existe pas de processus qui reste en phase de correction locale. Soit e'' un suffixe de e sans réinitialisation locale et sans processus en phase de correction locale. Supposons que l'ensemble R' des processus p satisfaisant $\neg AllCorrect_p$ dans le premier état de e' n'est pas vide ; alors R' est invariant dans toute configuration de e'' parce que si $AllCorrect_p$ est faux pour p, alors il existe $q \in \mathcal{N}_p$ tel que $AllCorrect_q$ est aussi faux. Dans ce cas p et q ne peuvent que se réinitialiser. Maintenant, en utilisant le même argument que dans le cas (2), les processus à distance 1 de R' ne peuvent faire qu'au plus 2 incrémentations, les processus à distance 2 ne peuvent faire qu'au plus 4 incrémentations, et ainsi de suite. Par récurrence, les processus à distance k ne peuvent faire qu'au plus $2k$ incrémentations. Puisque G est fini, e'' ne peut pas être infini et donc R' est vide. Ainsi $AllCorrect_p$ est vrai pour tous les processus p et toute configuration de e'' satisfait Γ_M.

□

Lemme 55 Soit $(p_0, t_0), (p_1, t_1), \ldots, (p_i, t_i)$ un chemin dans le DAG de réinitialisation, où $i \in \mathbb{N}^*$ alors, $\forall t \in]t_{i-1}, t_i] : \ p_0.r \in \{\varphi^j(-\alpha), j \in \{0, \ldots, i-1\}\}$.

Preuve : On fait une preuve par récurrence sur la longueur du chemin :
Si $i = 1$ le lemme est trivialement vérifié par définition de la relation de génération de réinitialisation.
Supposons que le lemme est vérifié pour tout chemin de longueur dans $[1, \ldots, i]$. Soit $(p_0, t_0), (p_1, t_1), \ldots, (p_{i+1}, t_{i+1})$ un chemin de longueur $i+1$. On sait pas hypothèse que

$\forall t \in]t_{i-1}, t_i],\ p_0.r \in \{\varphi^j(-\alpha),\ j \in \{0, \ldots, i-1\}\}, p_1.r \in \{\varphi^j(-\alpha), j \in \{0, \ldots, i-2\}\} \ldots$ et $p_{i-1}.r \in \{\varphi^j(-\alpha),\ j \in \{0\}\}$. A la date t_i, p_i se réinitialise et $p_i.r$ reste à la valeur α au moins jusqu'à t_{i+1} ; donc p_{i-1} peut effectuer l'action NA au plus une fois, ou peut appliquer l'action RA. Donc : $\forall t \in]t_i, t_{i+1}]$, $p_{i-1}.r \in \{\varphi^j(-\alpha),\ j \in \{0, 1\}\}$ et successivement $p_{i-2}.r \in \{\varphi^j(-\alpha),\ j \in \{0, 1, 2\}\} \ldots, p_0.r \in \{\varphi^j(-\alpha),\ j \in \{0, \ldots, i\}\}$ et le lemme est prouvé par récurrence.

□

Nous prouvons maintenant la convergence, pour cela nous examinerons deux cas : $\alpha = 0$ et $\alpha > 0$.

Définition 33 [La constante T_G] Soit T_G la constante définie par : $T_G = 2$ si G est un arbre sinon, T_G est égale à la longueur du plus grand trou du réseau.

Théorème 56 [Convergence dans le cas $\alpha = 0$]
Si $M \geq T_G - 2$, alors l'algorithme $SSGAU(K, 0, M)$ converge vers Γ_M et si $K > MC_G$, il est donc instantanément stabilisant pour les spécifications de l'unisson.

Preuve : La définition de l'action (NA) garantit la synchronisation.
Si le réseau est un arbre alors, par le théorème 53, le DAG de réinitialisation est fini. Alors, dans ce cas, par le théorème 54, l'algorithme $SSGAU(K, 0, M)$ est convergent vers Γ_M, il est donc instantanément stabilisant pour les spécifications de l'unisson si $K > MC_G$.
Supposons maintenant que G contient des cycles. Considérons le chemin de réinitialisations $(p_0, t_0), \ldots, (p_i, t_i)$ tel que $p_0, p_1, \ldots, p_i, p_0$ est un trou.
Puisque $i < T_G$ et $i \geq 2$, alors $i - 2 < T_G - 2 \leq M$ et par le lemme 55 on a

$$\forall t \in]t_{i-2}, t_{i-1}],\ p_0.r \in \{\varphi^j(-\alpha), j \in \{0, \ldots, i-2\}\} \subset \{0, ..., M-1\}$$

Soit β la valeur de $p_i.r$ à la date t_{i-1}. p_i se réinitialise à la date t_i et puisque $(p_{i-1}, t_{i-1}) \rightsquigarrow (p_i, t_i)$, on a $\forall t \in [t_{i-1}, t_i[,\ p_{i-1}.r = 0$ et $p_i.r \notin \{-M, ..., M\}$. Mais $\forall t \in [t_0, t_{i-1}]\ p_0.r \in \{0, ..., M-1\}$, donc p_i , qui est un voisin de p_0, ne peut pas effectuer l'action NA pour prendre la valeur β durant cette période. En d'autres termes, durant $[t_0, t_i]$, p_0 ne peut pas incrémenter son registre $p_0.r$ et donc $p_0.r = 0$.
Conclusion : $\forall t \in [t_0, t_{i-1}],\ p_0.r = 0$ et $p_i.r = \beta \notin \{-M, ..., M\}$, et la seule action pour p_0 et p_i est que p_i se réinitialise ; c'est ce qu'il fait à la date t_i.
Cela implique que $(p_0, t_0) \rightsquigarrow (p_i, t_i)$. Donc le DAG de réinitialisation est trou-transitif (définition page 99). Par le théorème 53, le DAG de réinitialisation est fini. Ainsi, par le théorème 54, la convergence vers Γ_M est prouvée. Si $K > M.C_G$ l'algorithme est auto-stabilisant pour les spécifications de l'unisson. Si on remarque que l'exécution du code de l'application ne se fait que dans les conditions de la synchronisation, on en déduit que l'algorithme est instantanément stabilisant.

□

Proposition 57
Si $T_G = 3$, alors le choix de $M = 1$ est optimal pour la convergence de l'algorithme

$SSGAU(K, 0, M)$.
Si $T_G > 3$ alors le choix $M = T_G - 2$ est optimal pour la convergence de l'algorithme $SSGAU(K, 0, M)$.

Preuve : Le cas $T_G = 3$ est facile à prouver.
Supposons maintenant que $T_G > 3$. Soit $p_1 \dots p_{T_G} p_1$ un plus long trou de G. Supposons que $0 < M < T_G - 2$. Considérons la configuration :

$$(p_1.r, p_2.r, \dots p_k.r, p_{k+1}.r, p_{k+2}.r, \dots, p_{T_G}.r) = (1, 2, ..., M, M+1, 0, ..., 0)\,,$$

tous les autres registres contenant la valeur 1.
Considérons l'exécution suivante pour $p_1, ...p_{T_G}$, les autres processus demeurant inchangés :
$(1, 2, \dots, M, M+1, 0, \dots, 0) \to (1, 2, \dots, M, M+1, 0, \dots, 1) \to$
$(2, 2, \dots, M, M+1, 0, \dots, 1) \to (2, 2, \dots, M+1, 0, 0, \dots, 1)$.
Alors, étape par étape, le système finit par atteindre la configuration suivante :

$$(2, 3, \dots, M+1, M+1, 0, \dots, 1)$$

Et maintenant p_{M+1} se réinitialise et on obtient : $(2, 3, \dots, M+1, 0, 0, \dots, 1)$.
Ainsi, le système n'atteindra jamais une configuration dans Γ_M, et donc ne stabilisera jamais.

□

Théorème 58 [Convergence dans le cas $\alpha > 0$]
Si $\alpha \geq T_G - 2$, alors l'algorithme $GAU(K, \alpha, M)$ est convergent vers Γ_M ; si de plus $K > MC_G$, alors il est instantanément stabilisant pour les spécifications de l'unisson.

Preuve : la preuve est similaire à celle du théorème 56.
Si le réseau est un arbre, alors, par le théorème 53, le DAG de réinitialisation est fini. Ainsi, dans ce cas, par le théorème 54, l'algorithme $SSGAU(K, \alpha, M)$ est convergent.
Maintenant on suppose que G contient des cycles. Considérons un chemin de réinitialisations $(p_0, t_0), \dots, (p_i, t_i)$tel que $p_0, p_1, \dots, p_i, p_0$ est un trou.
Puique $i < T_G$ et $i \geq 2$, alors $i - 2 < T_G - 2 \leq M$ et par le lemme 55 on a

$$\forall t \in]t_{i-2}, t_{i-1}],\ p_0.r \in \{\varphi^j(\alpha), j \in \{0, \dots, i-2\}\} \subset tail^*_\varphi$$

Soit β la valeur de $p_i.r$ à la date t_{i-1}. p_i se réinitialise à la date t_i et puisque $(p_{i-1}, t_{i-1}) \rightsquigarrow (p_i, t_i)$, on a $\forall t \in [t_{i-1}, t_i[\ p_{i-1}.r = \alpha$ et $p_i.r \notin tail_\varphi$, donc $\beta > 0$. Mais $\forall t \in [t_0, t_{i-1}],\ p_0.r \in tail^*_\varphi$, donc p_i , qui est un voisin de p_0, ne peut pas faire l'action NA pour prendre la valeur β durant cette période. En d'autres termes, pendant $[t_0, t_i]$, p_0 ne peut pas incrémenter $p_0.r$ et donc $p_0.r = -\alpha$.
Conclusion : $\forall t \in [t_0, t_{i-1}]\ p_0.r = -\alpha$ et $p_i.r = \beta > 0$, et la seule action possible de $\{p_0, p_i\}$ est que p_i se réinitialise. C'est ce qu'il fait à la date t_i.
Cela implique que $(p_0, t_0) \rightsquigarrow (p_i, t_i)$. Ainsi le DAG de réinitialisation est trou-transitif (Definition 2.3.2). Par le théorème 53, le DAG de réinitialisation est fini. Ainsi, par le théorème 54, la convergence vers Γ_M est prouvée. Si de plus $K > MC_G$ alors le protocole

est auto-stabilisant pour les spécifications de l'unisson. De plus l'action de l'application ne peut se faire que quand les voisins sont égaux ou en avance sur le processus p, l'algorithme est donc instantanément stabilisant.

□

Proposition 59 Si $\alpha > 0$ alors :

1. si $T_G = 3$ alors $\alpha = 1$ est optimal ;
2. si $T_G > 3$ alors $\alpha = T_G - 2$ est optimal pour $SSGAU(K, \alpha, M)$.

Preuve : Si $T_G = 3$, on peut prendre $\alpha = 1$ qui est optimal sous nos hypothèses.
Supposons maintenant que $T_G > 3$. Soit $p_1 \dots p_{T_G} p_1$ un plus grand trou de G. Supposons que $0 < \alpha < T_G - 2$. Considérons la configuration suivante :

$$(p_1.r, p_2.r, \dots p_\alpha.r, p_{\alpha+1}.r, p_{\alpha+2}.r, \dots, p_{T_G}.r) = (-1, -2, ..., -\alpha, 1, 0, ..., 0)\,,$$

les autres registres contenant 0. Considérons l'exécution suivante sur les processus : $p_1, ...p_{T_G}$, les autres processus demeurant inchangés :
$(-1, -2, \dots, -\alpha, 1, 0, \dots, 0) \to (-1, -2, \dots, -\alpha, 1, 1, 0, \dots) \to (-1, -2, \dots, -\alpha, -\alpha, 1, 0, \dots)$
$\to (-1, -2, \dots, -\alpha + 1, -\alpha, 1, 0, \dots)$ ….
Etape par étape, le système peut aboutir à la configuration suivante :

$$(0, -1, \dots, -\alpha + 1, -\alpha, 1, 0, \dots)$$

Alors, le système peut ne jamais atteindre un état de Γ_M, et donc ne jamais se stabiliser.

□

2.4 Temps de convergence

Rappelons que pour l'algorithme 7, les constantes M, K et α vérifient : $M > 0$, $K > 2M$ et $\alpha \geq 0$.
Notre étude porte sur les conditions permettant d'avoir : $\Gamma \triangleright \Gamma_M \triangleright \Gamma^0_M \triangleright \Gamma^0_1$.
On sait par le corollaire 20 que si $K > M.C_G$, alors $\Gamma_M = \Gamma^0_M$.
On sait par le théorème 28 du chapitre 2 que $\Gamma^0_M \triangleright \Gamma^0_1$ en au plus Md rondes.
Il nous reste à examiner le temps de convergence de $\Gamma \triangleright \Gamma_M$.

2.4.1 Temps de convergence vers Γ_M

Le DAG de réinitialisation ne contient pas de bégaiement, il est trou-transitif, donc en particulier il ne contient aucune hélice. L'algorithme $SSGAU$ converge vers Γ_M si une des deux propositions suivantes est vraie :

1. $\alpha = 0$ et $M > T_G - 1$
2. $\alpha \geq T_G - 2$ et $M \geq 1$

Sous une des conditions ci-dessus, la propriété de trou-transitivité et l'absence de bégaiement impliquent que la projection $p_0, p_1, \ldots, p_k$ d'un chemin de réinitialisations $(p_0, t_0), (p_1, t_0 + 1) \ldots (p_k, t_0 + k)$ sur G est un chemin élémentaire sans corde. Puisque les réinitialisations initiales ont lieu durant la première ronde, on a :

Lemme 60 La profondeur du DAG de réinitialisation est au plus CP_G, où CP_G est la longueur du plus long chemin élémentaire sans corde de G.

Clairement, le temps de la remontée dans la queue de la cerise est borné par α rondes. On obtient le théorème suivant :

Théorème 61 Si une des propositions suivantes est vraie :

1. $\alpha = 0$ et $M > T_G - 1$
2. $\alpha \geq T_G - 2$ et $M \geq 1$

alors on a la convergence $\Gamma \triangleright \Gamma_M$ et le temps de convergence est majoré par $CP_G + \alpha$ rondes.

Il est facile de montrer que la borne CP_G peut être atteinte. On pourrait penser que le temps de propagation des réinitialisations locales est dans $O(d)$ rondes. Malheureusement, les réinitialisations peuvent ne pas se propager "linéairement" dans le réseau, cela est dû au fait qu'un registre peut être égal à zéro, alors il ne se réinitialise pas et reste bloqué. La valeur 0 se comporte comme une barrière qui doit être contournée par les réinitialisations. La figure 3.3 montre un exemple de système avec un tel comportement. L'horloge de p étant égale à 0, elle interdit aux réinitialisations locales de se propager directement de la gauche vers la droite du réseau.

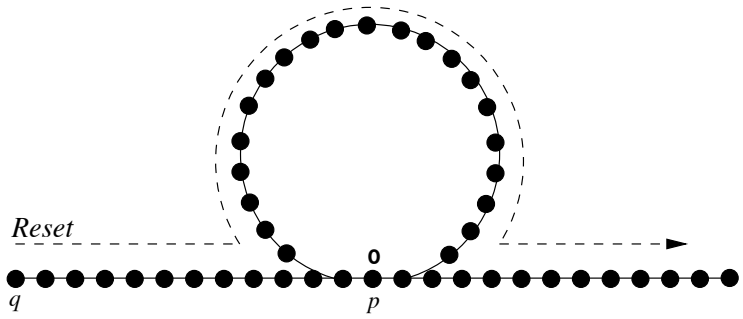


FIG. 3.3 – Un exemple montrant pourquoi une réinitialisation peut ne pas se propager en $O(d)$ rondes.

2.4.2 Temps de convergence pour $\alpha = 0$

Notre étude montre que le temps de stabilisation est majoré par $CP_G + Md$ où $M \geq T_G - 2$ et $K > MC_G$.
On a les relations : $C_G \leq T_G \leq CP_G \leq n$. Ainsi $CP_G + Md \leq n + Md$.

1. Dans le cas général, si on prend $M = T_G - 2$, on obtient $CP_G + Md \leq n(d+1)$ et la convergence est en $O(nd)$ rondes.
2. Si G est un arbre alors $C_G = T_G = 2$ et $CP_G = d$. Le temps de convergence est majoré par $d(M+1)$. Si $M = 1$ alors le temps est majoré par $2d$ rondes. Si $M = 1$ et $K = 3$ alors $\Gamma_M = \Gamma_M^0 = \Gamma_1^0$ et le temps de convergence est égal à 0.
3. Si G est un anneau, alors $C_G = T_G = CP_G = n$. Le temps de stabilisation est majoré par $n(\frac{M}{2} + 1)$. Si on prend $M = T_G - 2 = n - 2$ alors le temps de convergence est majoré par $\frac{n^2}{2} = O(n^2)$.

2.4.3 Temps de convergence pour $\alpha > 0$

Dans le cas $\alpha > 0$, notre étude montre que le temps de convergence est majoré par $CP_G + \alpha + Md$, où $M \geq 1$, $\alpha \geq T_G - 2$ et $K > MC_G$.
On a la relation $C_G \leq T_G \leq CP_G \leq n$. Ainsi $CP_G + Md \leq n + Md$.

1. Si on prend $M = 1$ et $\alpha = n$ on obtient $CP_G + \alpha + d \leq 2n + d = O(n)$ et la stabilisation est dans $O(n)$ rondes. Ce qui est meilleur que dans le cas $\alpha = 0$.
2. Si G est un arbre alors $C_G = T_G = 2$ et $CP_G = d$. On peut prendre $\alpha = 1$, et le temps de convergence est majoré par $d + 1 + Md = O(Md)$. Si $M = 1$ alors le temps de convergence est majoré par $2d + 1 = O(d)$. Sur un arbre, il est plus efficace de choisir $\alpha = 0$.
3. Si G est un anneau alors $C_G = T_G = CP_G = n$. Si on prend $\alpha = T_G - 2 = n - 2$ et $M = 1$ alors le temps de convergence est majoré par $2n = O(n)$. Dans ce cas, le choix $\alpha > 0$ est meilleur que le choix $\alpha = 0$.

2.4.4 Le cas de l'unisson de Couvreur, Francez et Gouda

Dans ce cas : α, $M = n$ et $K \geq n^2$. Le temps de convergence est dans $O(nd)$. En particulier sur un arbre, il est dans $O(nd)$, ce qui n'est guère efficace.

3 Unisson et complexité en espace

Dans cette courte partie, nous discutons le problème de l'optimalité en espace de l'unisson. Nous commençons par montrer comment, à partir d'un unisson, il est possible de construire, de manière auto-stabilisante, un arbre couvrant en largeur d'abord. Cela ne fournit pas un nouvel algorithme de construction d'arbre courvrant en largeur auto-stabilisant particulièrement performant, mais à contrario, cela nous donne un outil pour montrer qu'il n'existe pas d'unisson dont l'occupation mémoire est indépendante de la taille du réseau. Dans le cas général d'un réseau quelconque bien entendu, cet énoncé étant faux par exemple sur la classe des arbres, comme nous l'avons vu.

3.1 Construction auto-stabilisante d'un arbre couvrant en largeur d'abord.

Algorithme 8 $(SS-BFS)$: protocole auto-stabilisant de construction d'un arbre couvrant en largeur basé sur SS_AU

Constantes et variables :
$\mathcal{N}_p$: l'ensemble des voisins du processus p ; $p.parent : \mathcal{N}_p \cup Nil$ le père ou Nil $r_p \in \chi$;
Fonctions booléennes :

$ConvergenceStep_p \equiv r_p \in tail^*_\varphi \wedge (\forall q \in Np : (r_q \in tail_\varphi) \wedge (r_p \leq_{tail_\varphi} r_q))$;
$LocallyCorrect_p \equiv r_p \in ring_\varphi \wedge (\forall q \in Np, r_q \in ring_\varphi \wedge ((r_p = r_q) \vee (r_p = \varphi(r_q)) \vee (\varphi(r_p) = r_q)))$;
$NormalStep_p \equiv r_p \in ring_\varphi \wedge (\forall q \in \mathcal{N}_p : (r_p = r_q) \vee (r_q = \varphi(r_p)))$;
$ExistePere_p \equiv \exists q \in \mathcal{N}_p : \varphi(q.r) = p.r$;

Si p est la racine :
$CorrectPere_p \equiv (p.parent = Nil)$;
Si p n'est pas la racine :
$CorrectPere_p \equiv (p.parent \neq Nil) \wedge (\varphi(p.parent.r) = p.r)$;
Actions :

$CA : ConvergenceStep_p \longrightarrow r_p := \varphi(r_p)$;
$RA : \neg LocallyCorrect_p \wedge (r_p \notin tail_\varphi) \longrightarrow r_p := -\alpha$; (* réinitialisation locale *)
Si p est la racine :
$CP : \neg\, CorrectPere_p \longrightarrow p.parent := Nil$;
Si p n'est pas la racine :
$NA : NormalStep_p \longrightarrow r_p := \varphi(r_p)$;
$CP : \neg\, CorrectPere_p \wedge ExistePere_p \longrightarrow p.parent := ChoisirPère()$;

Dans cette partie nous montrons comment rendre auto-stabilisant l'algorithme 4. L'idée de base est d'utiliser un unisson. On suppose que $K > C_G$. En reprenant les idées de la section 6.2 du chapitre 2, il suffit de supprimer l'action (NA) du programme de la racine. L'algorithme convergera alors vers un interblocage dont le graphe d'interblocage (cf. page 4.2.3) est le $BFS - Dag$ enraciné en la racine $root$. Nous sommes donc en mesure de construire un $BFS - Dag$ de manière auto-stabilisante. Il reste à organiser le choix d'un père.
On introduit pour chaque processus p une nouvelle variable $parent$ dont le contenu est soit Nil (pas de père), soit un pointeur vers un voisin de p.
On introduit deux prédicats $ExistePere$ et $CorrectPere$ définis par :

1. $ExistePere_p \equiv \exists q \in \mathcal{N}_p : \varphi(q.r) = p.r$
2. Pour la racine : $CorrectPere_p \equiv (p.parent = Nil)$
3. Pour les autres processus : $CorrectPere_p \equiv (p.parent \neq Nil) \wedge (\varphi(p.parent.r) = p.r)$

et l'instruction :

1. Pour la racine : $(CP) : \neg\, CorrectPere_p \longrightarrow p.parent := Nil$
2. Pour les autres processus :$(CP) : \neg\, CorrectPere_\wedge ExistePere_p \longrightarrow p.parent := ChoisirPere()$

L'algorithme 8 est une instance basée sur l'algorithme d'unisson SS_AU [BPV04].

3.2 Occupation mémoire

Dans le chapitre 2, nous avons montré que si l'ordre de l'horloge est plus grand que C_G (la constante cyclomatique du graphe), alors tous les résiduels sont nuls, et qu'alors la correction de l'unisson dans Γ_M est contrôlable localement (à distance 1). Nous avons utilisé cette propriété pour détecter l'incorrection globale éventuelle d'un réseau. C'est pourquoi nos algorithmes sont dans $O(\ln(C_G))$ en occupation mémoire. La question est de savoir si on peut descendre en dessous. Descendre en dessous de $\ln(C_G)$ voudrait dire être capable de détecter des résiduels non-nuls et de les briser. Annuler un résiduel est facile. Le protocole SS_RU en lançant une procedure de réinitialisation mettrait tous les résiduels à zéro. On voit donc qu'il est facile de faire disparaître un résiduel non nul. Le problème est de détecter ces résiduels non nuls.
Détecter un résiduel non nul revient à déterminer une propriété stable du réseau. Il est donc possible de détecter ces résiduels par une technique de "prise d'un instantané du réseau" (snap-shoot). Il faudrait pour cela que le réseau soit enraciné avec identités mais l'occupation mémoire nécessaire au calcul d'un instantané global serait prohibitive.
D'un autre point de vue, il est facile de montrer qu'il n'existe pas d'unisson sur un graphe quelconque dont la complexité est dans $O(1)$. Il suffit pour cela de s'appuyer sur le résultat de Dolev, Gouda et Schneider ([DGS96] et pour un énoncé plus correct [Dol00]) sur l'impossibilité d'un protocole de calcul auto-stabilisant d'un arbre couvrant en largeur et silencieux, avec une occupation mémoire dans $O(1)$ pour chaque registre de communication, donc, dans notre modèle à états, avec une taille d'horloge dans $O(1)$.

Théorème 62 Dans le modèle à états, il n'existe pas d'unisson dont l'occupation mémoire par processus est dans $O(1)$ c'est-à-dire indépendante du réseau.

Preuve : Supposons qu'il existe un tel unisson. Alors d'aprés le paragraphe 3.1, on peut construire un arbre couvrant en largeur d'abord avec un protocole dont l'occupation mémoire par processus est dans $O(1)$, indépendant du réseau. Ce qui est en contradiction avec le résultat de [DGS96].

□

Deuxième partie

Optimiser

Chapitre 4

Synchronisme et asynchronisme

FIG. 4.1 – Ville de Wufeng, Taiwan

Sommaire

1 Introduction

Dans ce chapitre nous considérons le problème de *l'unisson* dans le cas particulier d'un réseau anonyme synchrone. Le problème de l'unisson synchrone peut s'énoncer ainsi : définir un protocole qui amène toutes les horloges à l'unisson, celles-ci s'incrémentent ensuite à l'unisson indéfiniment. Dans ce chapitre, notre discussion porte sur un graphe général (connexe) puis sur les arbres. Dans la suite K est l'ordre de l'horloge, S est le nombre d'états par processus. n est le nombre de processus et d le diamètre du réseau. Δ est le degré maximal d'un processus.
Le premier unisson synchrone est proposé dans [GH90]. Il est défini sur un graphe général connexe, il requièrt une horloge non bornée. Le premier protocole d'unisson synchrone avec une horloge bornée est proposé dans [ADG91]. Il nécessite $K \geq 2\Delta d$, et converge en $3\Delta d$ pulsations. Comme il est remarqué dans [HG95], le facteur Δ provient du modèle éudié dans lequel un processus ne peut lire que l'état d'un seul de ses voisins à la fois.
Dans notre modèle chaque processus peut lire les états de tous ses voisins en une étape atomique. Ainsi, dans notre modèle, la solution [ADG91] nécessite $K \geq 2d$ ($S = K$) et stabilise en $3d$ pulsations seulement. A notre connaissance, c'est le seul unisson synchrone déterministe sur le graphe général sans identités connu avant notre travail.
Une solution sur les arbres est proposée dans [HG95]. Elle demande $K = 3^m$ ($m > 0$), $S = K$, et stabilise en $(d \times (K-1))/2$ pulsations. Le temps de convergence est égal à d seulement lorsque $m = 1$ et $K = 3$, mais est plus grand que $2d$ quand $m \geq 2$. Ainsi, quand $3^m \geq 2d$, la solution de [ADG91] est meilleure.
Du point de vue de la convergence, la meilleure solution sur l'arbre est proposée par [NV01]. Elle converge en au plus d pulsations ; K prend n'importe quelle valeur supérieure ou égale à 2 et le protocole n'exige aucune connaissance globale comme la taille n ou le diamètre d, ni même d'un majorant d'une de ces quantités. Il dépend seulement du degré maximal Δ, ainsi $S = (\Delta + 1)K$.

1.1 Contribution.

La contribution de ce chapitre est de trois ordres :

1. On montre d'abord qu'il y a une connexion forte entre l'unisson synchrone et l'unisson asynchrone : quand un unisson asynchrone est implémenté sur un système synchrone alors il converge vers un unisson synchrone à la condition que $K > M.C_G$.

La première conséquence est que les solutions d'unissons asynchrones proposées par [CFG92], par [BPV04] et dans le chapitre 3 du présent mémoire, donnent des solutions d'unissons synchrones.

2. Le deuxième résultat est que l'unisson asynchrone de [BPV04], c'est à dire le protocole $GAU(K, \alpha, 1)$ (voir programme 7 page 96), fournit un unisson synchrone sur l'arbre qui est optimal en espace, c'est-à-dire qui fonctionne avec n'importe quel ordre d'horloge $K \geq 3$, et dans ce cas $S = K$; le protocole converge en au plus $2d$ pulsations. Cette solution donne une réponse positive à la question posée par [Nol02] : "*existe-t-il un unisson synchrone sur l'arbre dont le nombre d'états est indépendant de toute information locale ou globale sur l'arbre (i.e. n, d, ou* Δ*) ?*". On peut remarquer aussi que pour le cas $K = 3$, le temps de convergence est majoré par d seulement. Ce protocole atteint les performances de [HG95]. Il est surprenant qu'un protocole général sur un réseau asynchrone quelconque résolve ainsi le problème de l'unisson synchrone sur l'arbre avec de telles performances.

   Cependant, ces bons résultats ne se prolongent pas au graphe général. Dans le cas général et pour $M = 1$, le protocole nécessite $K > C_G$ et $S = K + T_G - 2$, ce qui est en général moins bon que $2d$ [ADG91]. Il stabilise en $CP_G + T_G + d$ rondes au plus, où CP_G est la longueur du chemin élémentaire sans corde le plus long du réseau G. Le temps de convergence du protocole vers Γ_0 est donc dans $O(n)$ dans le cas général. Le temps de convergence est donc aussi moins bon que le $3d$ du protocole proposé par [ADG91].

3. Le troisième résultat de ce chapitre est une nouvelle solution générale qui utilise les avantages des deux approches de [ADG91, BPV04]. Ce protocole appelé $SS\text{-}MinSU$, demande $K \geq 2$ et $S \geq K + d$. Son temps de convergence est majoré par $2d$ seulement. C'est la meilleure solution actuelle pour un réseau quelconque synchrone, aussi bien en temps de convergence qu'en occupation mémoire.

1.2 Structure du chapitre.

Dans la première section nous précisons les problèmes considérés dans ce chapitre et nous montrons que tout unisson exécuté sur un système synchrone résout le problème de l'unisson synchrone pourvu que $K > M.C_G$. Dans cette même section, nous discutons les performances du protocole [BPV04] et plus généralement du protocole $GAU(K, \alpha, M)$, exécutés sur un système synchrone.

Le protocole $SS\text{-}MinSU$, ainsi que sa correction, sont présentés dans la section 4. Nous terminons par quelques remarques de conclusion dans la section 5.

2 Définition du problème

2.1 L'unisson synchrone

On suppose que chaque processus p maintient un registre d'horloge $p.r$ avec le système d'incrémentation $(\mathcal{X}, \varphi)$ d'ordre K. quand $p.r \in \{0, 1, ..., K-1\}$ on dit que le processus p a fini la phase $p.r$.
La notion de barrière de synchronisation est introduite dans [Mis91], par Misra. Il propose un algorithme non auto-stabilisant qui effectue une synchronisation dite globale.
La *barrière de synchronisation forte asynchrone* est définie par :

Spécification 7 [Barrière de synchronisation forte asynchrone] Pour tout $i \in \{0, 1, ..., K-1\}$

1. **sûreté** :
   (a) (intitialisation) initialement, tous les processus ont effectué correctement la phase 0 ;
   (b) (synchronisation) aucun processus ne peut effectuer *l'action de l'application* de la phase $i+1$ *mod* K et incrémenter son horloge tant que tous les processus n'ont pas fini l'action d'application de la phase i ;
2. **vivacité** : tout processus p exécute *l'action de l'application* puis incrémente son horloge une infinité de fois.

La barrière de synchronisation forte synchrone est définie par :

Spécification 8 [Barrière de synchronisation forte synchrone]
Pour tout $i \in \{0, 1, ..., K-1\}$

1. **sûreté** :
   (a) (intitialisation) initialement, tous les processus ont effectué correctement la phase 0.
   (b) Γ_0 est clos ;
   (c) un processus peut effectuer *l'action de l'application* de la phase $i+1$ *mod* K et incrémenter son horloge seulement si tous les processus ont terminé d'effectuer l'action d'application de la phase i ;
2. **vivacité** : tout processus p exécute *l'action de l'application* puis incrémente son horloge une infinité de fois.

La seule différence entre ces deux spécifications est que dans le cas synchrone, la clôture de Γ_0 est exigée.
Dans l'esprit de la définition d'un *unisson asynchrone* donné page 94, un *unisson synchrone* est une barrière de synchronisation forte auto-stabilisante assurant une synchronisation faible durant la convergence :

Spécification 9 [Unisson synchrone]

Pour tout $i \in \{0, 1, ..., K-1\}$:

1. **sûreté** : (synchronisation) un processus peut effectuer l'*action de l'application* de la phase $i+1$ *mod* K puis incrémenter son horloge seulement si tous ses voisins ont leur horloge à la valeur i ou $i+1$ mod K ;
2. **vivacité** :
   (a) (convergence) $\Gamma \triangleright \Gamma_0$;
   (b) dans Γ_0 chaque processus ne peut qu'effectuer l'*action de l'application* puis incrémenter son horloge et le processus enchaîne ces deux actions une infinité de fois.

La condition de sûreté de l'unisson est exprimée un peu différemment de la condition de sûreté des barrières de synchronisation fortes. La raison vient du fait qu'il n'y a pas d'initialisation. Au démarrage il est possible qu'un processus fasse l'action de l'application sans qu'aucun voisin n'ai fait une seule fois l'action de l'application. Par ailleurs, la condition de sûreté impose une synchronisation faible durant la convergence : $\Gamma \triangleright \Gamma_0$. Ici aussi (voir l'unisson asynchrone page 94), la condition de sûreté : Γ_0 *est clos*, n'est pas nécessaire puisque la condition de vivacité : $\Gamma \triangleright \Gamma_0$, implique la clôture de Γ_0. Clairement, la notion d'unisson synchrone n'est pas équivalente avec la notion de barrière de synchronisation forte auto-stabilisante, elle est plus forte.

2.2 Action d'incrémentation dans Γ_0 et Γ_1.

Pour un processus p, appelons *règle d'action normale*, l'action qui consiste à effectuer l'*action de l'application* puis à incrémenter son horloge La règle d'action normale de chaque processus du système peut être définie par :

$$(NA) : Cond \longrightarrow [<< action \text{ de } l'application >>]; p.r := \varphi(p.r); \tag{4.1}$$

Quand un processus p a effectué l'action (NA), il a fini l'*action de l'application* de la phase $p.r$. Ainsi le registre $p.r$ donne le numéro de la dernière phase effectuée par le processus p. Notre première tâche est de définir le prédicat $Cond$ dans les cas synchrones et asynchrones :

1. Dans le cas d'un réseau synchrone, il faut définir le prédicat $Cond$ de sorte que les spécifications de la synchronisation forte soient vérifiées en commençant dans Γ_0 . Ce problème est trivialement résolu par le prédicat :

$$Cond \equiv vrai$$

2. Dans le cas d'un réseau asynchrone, on relaxe la barrière de synchronisation forte en la *barrière de synchronisation faible* (voir les spécifications page 93). En démarrant dans Γ_1, y-a-t-il plusieurs manières de résoudre ce problème ? La réponse est non, il n'y a pas plusieurs comportements possibles dans Γ_1 pour un unisson asynchrone, comme le montre le petit lemme suivant :

Lemme 63 Si un protocole uniforme $\mathcal{P}$ résout le problème de l'unisson asynchrone alors le prédicat $Cond$ définissant l'action normale (NA) satisfait :

$$Cond \equiv \forall q \in \mathcal{N}_p : \ (q.r = p.r) \vee (q.r = \varphi(p.r))$$

Preuve : En vertu de la propriété de synchronisation, $Cond \Rightarrow \forall q \in \mathcal{N}_p : \ (q.r = p.r) \vee (q.r = \varphi(p.r))$. Supposons qu'il existe une exécution de $\mathcal{P}$ pour laquelle il existe un état γ tel qu'il existe un processus p vérifiant $\forall q \in \mathcal{N}_p : \ (q.r = p.r) \vee (q.r = \varphi(p.r))$ et que $Cond$ soit $faux$ pour ce processus dans γ. Soit Δ_p le nombre de voisins de p et Δ_p^γ le nombre de voisins q de p tels que $q.r = \varphi(p.r)$ dans γ. Considérons maintenant une clique de $n = \Delta_p + 1$ processus. La valeur d'horloge de $n - \Delta_p^\gamma$ processus est égale à $p.r$; ainsi la valeur d'horloge de Δ_p^γ processus est égale à $\varphi\,(p.r)$. Dans une telle configuration, le protocole étant uniforme, $Cond$ est faux pour tous les processus ayant même valeur d'horloge que p, le système est interbloqué. Cela contredit la condition de vivacité de l'unisson.

□

Réciproquement, le chapitre 2 montre que si $K > C_G$ l'action gardée ci-dessus résout le problème de l'unisson asynchrone démarrant dans Γ_1.

Attention, le lemme 63 est évidemment faux si le protocole n'est pas uniforme, par exemple si le protocole tient compte d'identités sur les processus pour autoriser l'exécution de l'action d'application. Voir par exemple le chapitre 6, où nous exploiterons cette remarque. Clairement, nous venons d'obtenir une sorte de théorème d'unicité : un unisson asynchrone uniforme démarrant dans Γ_1 dépend uniquement du choix du système d'incrémentation $(\mathcal{X}, \varphi)$.

3 De l'unisson asynchrone vers l'unisson synchrone

Dans cette section nous montrons que sous un démon synchrone, on a toujours la convergence : $\Gamma_1^0 \rhd \Gamma_0$. C'est-à-dire que toute barrière de synchronisation faible dans Γ_1^0 converge vers une barrière de synchronisation forte. En d'autres termes, nous montrons que, sous l'hypothèse que $K > C_G$, le problème de la barrière de synchronisation forte auto-stabilisante sous un démon synchrone peut être résolu par un protocole résolvant le problème de la barrière de synchronisation faible auto-stabilisante sous un démon asynchrone . Il en résulte que, sous un démon synchrone, tout unisson avec une horloge d'ordre $K > C_G$ définit un protocole auto-stabilisant de barrière de synchonisation forte. La synchronisation faible imposée par les spécifications des unissons synchrones et asynchrones lors de la convergence assure alors qu'un unisson asynchrone est un unisson synchrone sous le démon synchrone.

3.1 Le théorème de convergence

Dans cette section, toute exécution est supposée commencer dans une configuration $\gamma \in \Gamma_1^0$. La seule action possible est donc l'action (NA) (voir action gardée page 115). Suivant le lemme 63, la garde $Cond$ est définie par : $\forall q \in \mathcal{N}_p :\ (q.r = p.r) \vee (q.r = \varphi(p.r))$. Rappelons que dans Γ_1^0 le retard est intrinsèque et que l'on note Δ_{pq} la valeur commune des retards suivant les chemins du processus p au processus q. Nous disons que p "*précède*" q dans une configuration γ si et seulement si $\Delta_{pq} \leq 0$. De même p et q sont "γ-*synchrones*" si $\Delta_{pq} = 0$ dans γ. Puisque le réseau est connexe, la relation de précédence est un préordre total. Deux processus minimaux, au sens du préordre, sont synchrones. L'ensemble des processus minimaux n'est pas vide puisque l'ensemble des processus est fini (non vide). De plus, les processus minimaux sont activables pour l'action (NA). Dans la suite, pour chaque état de Γ_1^0, nous noterons V_0 l'ensemble des processus minimaux.

Lemme 64 Soit $\gamma \in \Gamma_1^0$. Pour toute transition $\gamma \mapsto \gamma'$ ordonnancée par un démon synchrone, si $p \in V_0$ dans γ alors $p \in V_0$ dans γ'.

Preuve : Soit p et q deux processus distincts avec $p \in V_0$ dans γ. Puisque $p \in V_0$, p peut activer l'action (NA). Notons ρ la valeur de Δ_{pq} dans γ. Deux cas sont possibles :

1. $q \in V_0$ dans γ. Alors q est aussi activable dans γ et $\Delta_{pq} = 0$ dans γ'.
2. $q \notin V_0$ dans γ. Alors $\rho > 0$ et par le lemme page 66, $\Delta_{pq} \in \{\rho - 1, \rho\}$ dans γ'.

Dans les deux cas, q précède p dans γ'. On en déduit que p est minimal dans γ'.

□

La distance entre deux processus p et q est notée $d\,(p, q)$. C'est la longueur du plus court chemin entre p et q. Soit $p \in V$. On pose $\delta_p = \max_{q \in V} d(p, q)$, c'est l'écartement du processus p dans le réseau [Ber83]. Soit k un entier strictement positif. Définissons $B(p, k)$ comme l'ensemble des processus tels que $d(p, q) \leq k$, c'est la boule fermée de centre p et de rayon k. Remarquons que le diamètre d du réseau est égal à $\max_{p \in V} \max_{q \in V} d(p, q)$.

Théorème 65 $[\Gamma_1^0 \rhd \Gamma_0]$
Pour toute exécution ordonnancée par un démon synchrone, Γ_0 est un attracteur pour Γ_1^0. C'est-à-dire que toute exécution commençant en $\gamma \in \Gamma_1^0$ converge vers Γ_0
Le temps de convergence de Γ_1^0 vers Γ_0 est majoré par d et cette borne est optimale.

Preuve : Considérons une exécution maximale $e = \gamma_0, \gamma_1, \ldots$ telle que $\gamma_0 \in \Gamma_1^0$. Notons V_0^t l'ensemble des processus minimaux dans l'état γ_t. Soit p un élément de V_0^0 avec la quantité $\delta_p = w$ minimale.
Montrons par récurrence que :

$$\forall i \in \mathbb{N}, B(p, i) \subseteq V_0^i$$

Cela prouvera que si $i = w$, alors $V_0^i = V$, ce qui prouvera la convergence et le fait que le temps de convergence est inférieur ou égal à w, qui est majoré par d.

Initialisation : si $i = 0$, alors $B(p, 0) = \{p\} \subseteq V_0^0$.

Hérédité : supposons que $i \geq 0$ et $B(p,i) \subseteq V_0^i$. Soit $q \in B(p,i+1) \setminus B(p,i)$. Alors q est voisin d'un élément q' de $B(p,i)$. Deux cas sont possibles :
1. $q \in V_0^i$. Par lemme 64, $q \in V_0^{i+1}$.
2. $q \notin V_0^i$. Alors à la date i : $q.r = \varphi(q'.r)$. Donc q ne peut pas faire d'action dans la transition $\gamma_i \rightarrow \gamma_{i+1}$. Il résulte que $q \in V_0^{i+1}$.
Dans les deux cas $q \in V_0^{i+1}$. Nous en déduisons que $B(p,i+1) \subseteq V_0^{i+1}$.
Conclusion : par récurrence $\forall i \in \mathbb{N}, B(p,i) \subseteq V_0^i$.
La première partie du théorème s'en déduit. Montrons que la borne est la meilleure possible

Soit p un processus qui est a une extrémité d'un diamètre de G, i.e. $\delta_p = \max_{q \in V} \delta_q$. Supposons que $p.r = 0$ dans γ_0. Pour chaque q tel que $d(q,p) = k$ ($k \in 1, \ldots, d$), on définit $q.r = \overline{k}$. Cette configuration est clairement dans Γ_1^0 car tous les résiduels sont égaux à 0. En démarrant dans une telle configuration, Γ_0 est atteint en d pulsations. Nous venons de montrer que d peut être atteint. Cela prouve la deuxième partie du théorème.

□

Remarques

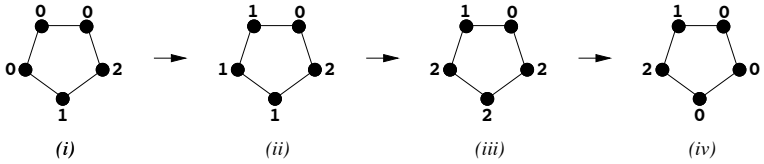
1. Pour prouver le théorème 65, on aurait pu aussi commencer par prouver le théorème dans le cas où les registres sont à valeurs dans $\mathbb{Z}$, puis se ramnener au cas des des registres bornés par le relèvement (voir page 72 et voir la preuve du théorème 28).
2. Le théorème 65 n'est vrai que si le système démarre dans Γ_1^0, c'est-à-dire quand dans l'état de départ le retard est intrinsèque. Si l'état de départ est dans $\Gamma_1 \setminus \Gamma_1^0$, le retard n'y est pas intrinsèque et d'après le théorème 20 page 20, dans cet état il y a au moins un résiduel qui n'est pas égal à 0. Nous savons que les résiduels sont :
   (a) invariants durant l'exécution du protocole ;
   (b) égaux à 0 dans Γ_0.

   On en déduit que si le retard n'est pas intrinsèque, le système ne peut pas converger vers Γ_0. Sur la figure 4.2, toutes les configurations sont dans Γ_1, mais elles ne sont pas dans Γ_1^0. Dans chacun des cas, le système ne converge pas dans Γ_0 ; pour plus de détails, voir le chapitre 2, page 59 et suivantes).

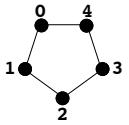
Grâce au théorème 65 et au théorème 20 page 20, on déduit que :

> **Théorème 66** Si $K > C_G$, alors tout unisson asynchrone uniforme $\mathcal{P}$ défini pour un démon asynchrone et ordonnancé par le démon synchrone résout le problème de l'unisson synchrone.

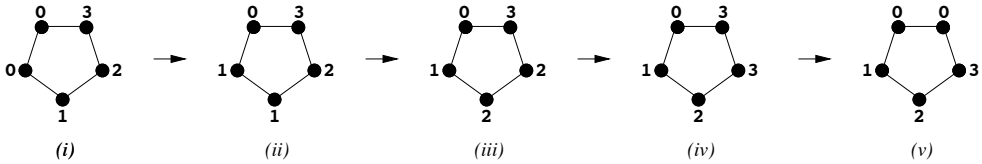
Il suit du théorème 20 que si nous faisons l'hypothèse $K > C_G$ ce qui revient à dire $\Gamma_1 = \Gamma_1^0$, alors pour définir une barrière de synchronisation forte auto-stabilisante (un unisson synchrone), il reste seulement à assurer la convergence vers Γ_1, c'est-à-dire $Vrai \rhd \Gamma_1$. Tout unisson asynchrone avec une horloge d'ordre $K > C_G$ satisfait cette spécification.



$K = 3$.
(Les configurations (i) et (iv) sont identiques.)



$K = 5$.
(Un interblocage.)



$K = 4$.
(Les configurations (i) et (v) sont identiques.)

FIG. 4.2 – Contre-exemples pour $K \leq C_G$ et $C_G = 5$.

Par exemple, les unissons à registres bornés présentés dans [CFG92, Dol00] vérifient $K > C_G$ puisque dans ces cas $K \geq n^2$ et sont uniformes. Dans [BPV04], nous proposons un algorithme appelé $SSAU$ et montrons que l'algorithm $SSAU$ résout le problème de l'unisson si $K > C_G$. L'algorithme plus général $GAU(K, \alpha, M)$ résout le problème, dans ces conditions de convergence (voir page 96).

Corollaire 1 Si $K > C_G$, alors, les unissons uniformes définis dans [BPV04, CFG92, Dol00], conduits par un démon synchrone, définissent des unissons synchrones. De même $GAU(K, \alpha, M)$ résout sous un démon synchrone, le problème de l'unisson synchrone sous les conditions énoncées page 96

3.2 Performances des unissons asynchrones.

Dans cette partie, nous discutons des performances des unissons asynchrones fonctionnant sur un système synchrone en tant qu'unisson synchrone. La question est donc d'évaluer en nombre de pulsations le temps de convergence vers Γ_0 . On suppose donc ici encore que le démon est synchrone.

3.2.1 Algorithme *SS_RU*

L'algorithme SS_RU (voir annexe A) est un unisson synchrone, le temps de convergence est dans $O(d)$, l'occupation mémoire est dans $O(\ln(n))$. Ce protocole exige un réseau avec identités.

3.2.2 Algorithme $GAU\,(K, \alpha, M)$

Nous reprenons les résultats du chapitre 3, page 105. La mauvaise diffusion des réinitialisations locales réduit les performances de cet unisson ; elles ne sont pas améliorées par le synchronisme dans le cas général, par contre sur l'arbre les performances sont trés bonnes :

1. **Temps de convergence pour** $\alpha = 0$.
   Notre étude montre que le temps de convergence est majoré par $CP_G + (M+1)\,d$ pulsations.
   (a) Dans le cas général, si on prend $M = T_G - 2$ on obtient $CP_G + (M+1)\,d \leq n(d+1)$ et la convergence est en $O(nd)$ rondes.
   (b) Si G est un arbre alors $C_G = T_G = 2$ et $CP_G = d$; le temps de convergence est majoré par $d(M+2)$. Si $M = 1$ alors le temps est majoré par $3d$ pulsations ; si $M = 1$ et $K = 3$ alors $\Gamma_M = \Gamma_M^0 = \Gamma_1^0$ et le temps de convergence est majoré par d pulsations.
2. **Temps de convergence pour** $\alpha > 0$**.**
   Dans le cas $\alpha > 0$, notre étude montre que le temps de convergence est majoré par $CP_G + \alpha + (M+1)\,d$, où $M \geq 1$, $\alpha \geq T_G - 2$ et $K > MC_G$.
   (a) Si on prend $M = 1$ et $\alpha = n$ on obtient $CP_G + \alpha + 2d \leq 2n + 2d = O(n)$ et la convergence est en $O(n)$ pulsations.
   (b) Si G est un arbre, alors $C_G = T_G = 2$ et $CP_G = d$. On peut prendre $\alpha = 1$ et alors le temps de convergence est majoré par $d + 1 + (M+1)\,d = O(Md)$. Si $M = 1$ alors le temps de convergence est majoré par $3d+1 = O(d)$. Sur un arbre il est plus efficace de choisir l'algorithme plus simple avec $\alpha = 0$ (voir ci-dessus).
3. **Le cas de l'unisson de Couvreur, Francez et Gouda**
   Dans ce cas : $M = n$ et $K \geq n^2$. Le temps de convegence est dans $O(nd)$ Sur un arbre, en particulier, il est dans $O(nd)$, ce qui n'est pas bon.
4. **Le cas de** $M = 1$
   Conformément à [BPV04], on note $SSAU$ l'algorithme $GAU\,(K, \alpha, 1)$. Le temps de convergence est en $O\,(n)$ pulsations sur le graphe général et en $O(d)$ sur l'arbre dans le cas $\alpha = 0$.

3.2.3 $GAU\,(K, \alpha, 1)$ sur l'arbre.

Sur l'arbre, si $\alpha = 0$, alors, puisque $CP_G = d$, le temps de convergence de l'algorithme $SSAU$ (ou $GAU\,(K, 0, 1)$) est majoré par $2d$.

> **Proposition 67** Sur l'arbre, pour toute horloge de taille $K > 2$, si $\alpha = 0$ alors l'algorithme $SSAU$ est un protocole d'unisson synchrone avec K états par processus, qui stabilise en au plus $2d$ pulsations.

La proposition 67 et l'algorithme 9 donnent une réponse positive à la question posée dans [Nol02] :

"Existe-t-il un unisson synchrone universel, c'est-à-dire avec une horloge d'ordre quelconque supérieur ou égal à trois, sur l'arbre dont une occupation mémoire est indépendante de toute information locale ou globale, c'est-à-dire sur les constantes n, d, ou Δ ?".

Algorithme 9 Unisson synchrone pour p sur l'arbre

Constantes et variables :

$\mathcal{N}_p$: l'ensemble des voisins de p ; $p.r \in \chi$;

fonctions booléennes :

$LocallyCorrect_p \equiv \forall q \in Np, ((p.r = q.r) \vee (p.r = \varphi(q.r)) \vee (\varphi(p.r) = q.r)))$;

$NormalStep_p \equiv (\forall q \in \mathcal{N}_p : (p.r = q.r) \vee (r_q = \varphi(p.r)))$;

Actions :

NA : $NormalStep_p \longrightarrow$ << code de l'application >> ;$p.r := \varphi(p.r)$;

RA : $\neg LocallyCorrect_p \longrightarrow p.r := 0$; (* Correction *)

Si, dans l'algorithme 9, on prend $K = 3$, alors les états sont tous dans Γ_1^0, c'est-à-dire que $\Gamma = \Gamma_1 = \Gamma_1^0$. Dans ce cas, il n'y a jamais de réinitialisation. Le protocole $SSAU$ est alors équivalent au protocole 10 défini par une unique garde, et qui implémente simplement l'incrémentation "naturelle" (voir l'action gardée page 115).

Algorithme 10 Unisson synchrone optimal pour le processus p dans un réseau en arbre.

Constants and variables :

$\mathcal{N}_p$: ensemble des voisins de p ; $p.r \in \{0, 1, 2\}$;

Action :

NA : $\forall q \in \mathcal{N}_p : (p.r = q.r) \vee (q.r = \varphi(p.r)) \longrightarrow$ << code de l'application >> ;$p.r := \varphi(p.r)$;

Théorème 68 Si $\alpha = 0$ et $K = 3$, alors l'algorithme $SSAU$ est un protocole d'unisson synchrone sur l'arbre. Il est optimal à la fois en temps de convergence : convergence en au plus d pulsations, et en nombre d'états : 3 états par processus.

La découverte de ce protocole est à l'origine d'une recherche complémentaire qui sera exposée au chapitre 5.

Remarque

A propos de l'algorithme $SSAU$, on peut remarquer que dans un système synchrone, la longueur de la queue peut être réduite de moitié. La condition devient $K \geq \lceil \frac{T_G}{2} \rceil - 1$. Mais cette amélioration n'est pas vraiment significative, c'est pourquoi nous avons développé un autre protocole sur le graphe général. Sa définition et ses performances font l'objet du chapitre suivant.

4 Unisson synchrone efficace sur le graphe général.

Dans la section précédente, nous avons montré que $CP_G + \alpha + d$ est une borne supérieure pour le temps de convergence de l'algorithme $SSAU$. Nous savons, depuis l'étude du

chapitre 3, que les réinitialisations locales de l'algorithme $SSAU$ ne se propagent pas de manière uniforme dans le réseau, c'est la raison de la lenteur de la propagation des réinitialisations. Nous allons maintenant éviter ce problème en proposant l'algorithme 11. C'est un protocole qui, dans un graphe arbitraire connexe, réduit le temps de propagation des réinitialisations à aux plus $2d$ pulsations. La preuve de correction et l'analyse des performances sont présentées dans cette section.

4.1 L'algorithme

le protocole, appelé protocole $SS\text{-}MinSU$, est présenté dans l'algorithme 11. Il est basé sur un système d'incrémentation, mais la réinitialisation est gérée de la manière suivante :

1. un processus p réinitialise son horloge à $-\alpha$ seulement si la valeur de l'horloge de p et celle des horloges de ses voisins sont toutes dans $ring_\varphi$, mais pas en phase. C'est l'action RA ;
2. si la valeur de l'horloge de p ou l'une ses voisins est dans $tail_\varphi$, alors p met son horloge à la valeur 1 plus la valeur minimum des horloges des processus dans $\{p\} \cup \mathcal{N}_p$. C'est l'action TA.

Si une configuration du système est dans $\Gamma_1 \setminus \Gamma_0$, au moins un processus doit se réinitialiser. Ainsi, cet algorithme n'a rien à voir avec la constante C_G. Il est clair que la non appartenance à Γ_0 est toujours localement contrôlable et que dans Γ_0 les résiduels sont tous trivialement à 0. Plus précisément, l'algorithme 11 requiert seulement la condition $K \geq 2$. Le nombre d'états par processus est $\alpha + K$ avec $\alpha \geq d$. Ainsi, ce protocole fonctionne avec $D + K$ états par processus, où D est un majorant de d. Le temps de convergence est au plus $2d$ pulsations.

Algorithme 11 (SS_MinSU) Unisson synchrone sur le graphe général

Constantes :

$\mathcal{N}_p$: ensemble des voisins du processus p ; $\overline{\mathcal{N}_p} = \mathcal{N}_p \cup \{p\}$;

Variables :

$p.r \in \mathcal{X}$;

Fonctions booléennes :

(* Incrémentation de l'horloge de phase*)

$LocalUnison_p \equiv \forall q \in \mathcal{N}_p : (p.r = q.r)$;

$NormalStep_p \equiv p.r \in ring_\varphi \wedge LocalUnison_p$;

(* Convergence *)

$TailStep_p \equiv \exists q \in \overline{\mathcal{N}_p},\ q.r \in tail^*_\varphi$;

(* Réinitialisation *)

$ResetInit_p \equiv \left(\forall q \in \overline{\mathcal{N}_p},\ q.r \in ring_\varphi\right) \wedge \neg LocalUnison_p$

Actions :

NA : $NormalStep_p \longrightarrow << \text{ code de l'application } >> ; p.r := \varphi(p.r)$;

TA : $TailStep_p \longrightarrow p.r := \varphi(min\{q.r, q \in \overline{\mathcal{N}_p}\})$;

RA : $ResetInit_p \longrightarrow p.r := -\alpha$; (* Reset *)

4.2 Preuve de correction

Pour prouver la validité de cet algorithme, nous introduisons la notion d'*état convergent* et la notion de *DAG de convergence* . Nous commençons par prouver l'absence d'interblocage pour chacun des processus. Ensuite, nous donnons une condition suffisante pour que le DAG de convergence soit fini. Sous cette condition, le système se stabilise dans Γ_0.
L'idée de la preuve d'extinction des réinitialisations est de montrer que chaque chemin dans le *DAG de convergence* est un chemin où les registres sont incrémentés au plus $\alpha+1$ fois.
Ensuite, nous prouvons que si $\alpha \geq d$, la valeur $-\alpha$ n'apparaît pas deux fois dans le même chemin. Comme chaque chemin commence à la date 0 ou 1, on en déduit ensuite qu'à la date $2d$, tous les états convergents ont disparu et le système est stabilisé dans un sens que nous préciserons.

Définition 34 [Etat convergent] Soit $e = \gamma_0\gamma_1\ldots$ une exécution maximale de l'algorithme *SS-MinSU*. Etant donné un processus p et $i \geq 0$, la paire (p,i) est dite être un *état convergent* dans l'exécution e si, dans l'état γ_i, l'une des propositions suivantes est vérifiée :

1. $p.r \in tail^*_\varphi$
2. $i > 0$, $p.r = 0$, et il existe $q \in \overline{\mathcal{N}_p}$ tel que $q.r = -1$ dans γ_{i-1}.

Dans la suite, quand il n'y a pas d'ambiguité, nous nous réfèrerons simplement à la notion d'état convergent. Quand cela est nécessaire, nous notons la valeur de $p.r$ dans γ_i par $\gamma_i.p.r$.

Définition 35 Soit (p_1,i_1) et (p_2,i_2) deux états convergents.
On dit que (p_1,i_1) *génère* (p_2,i_2), noté par : $(p_1,i_1) \overset{g}{\rightsquigarrow} (p_2,i_2)$, si et seulement si les trois conditions suivantes sont vérifiées :

1. $i_2 = i_1 + 1$
2. $p_2 \in \overline{\mathcal{N}_{p_1}}$
3. l'une des trois conditions suivantes est vérifiée :
   (a) $p_2.r = -\alpha$ dans γ_{i_2} et $p_1.r = 0$ dans γ_{i_1}
   (b) $p_2.r = 0$ dans γ_{i_2} et $p_1.r = -1$ dans γ_{i_1}
   (c) $p_2.r \in tail_\varphi - \{0,-\alpha\}$ dans γ_{i_2} et $\gamma_{i_2}.p_2.r = \varphi(\gamma_{i_1}.p_1.r)$.

La relation $\overset{g}{\rightsquigarrow}$ définit un graphe acyclique orienté (DAG), appelé *DAG de convergence*. Si (p_0,i_1) n'est généré par aucun autre état convergent, alors (p_1,i_1) est un *état convergent initial*.

Le lemme suivant explique la condition (b) et la définition d'*état convergent* 123.

Lemme 69 S'il existe deux processus voisins p_1 et p_2 ainsi qu'une transition $\gamma_i \mapsto \gamma_{i+1}$ $(i \geq 0)$ tels que $p_1.r$ et $p_2.r$ soient dans $ring_\varphi$ et $p_1.r \neq p_2.r$ dans γ_{i+1}, alors il existe $q_1 \in \{p_1,p_2\}$ tel que $q_1.r = 0$ dans γ_{i+1}, et il existe $q_0 \in \overline{\mathcal{N}_{q_1}}$ tel que $q_0.r = -1$ dans γ_i, c'est-à-dire que $(q_0,i) \overset{g}{\rightsquigarrow} (q_1,i+1)$.

Attention, le processus q_0 n'est ni le processus p_1 ni le processus p_2.

Preuve : Deux cas sont possibles dans γ_i :

1. Pour tout $q \in \overline{\mathcal{N}_{p_1}} \cup \overline{\mathcal{N}_{p_2}}$, $r_q \in ring_\varphi$.
   Si, à la date i, le prédicat *LocalUnison* est vrai pour p_1 et pour p_2, alors $p_1.r = p_2.r$ dans γ_{i+1}, ce qui est absurde. Donc *LocalUnison* est faux pour p_1 ou pour p_2. Donc, dans γ_{i+1}, $p_1.r$ ou $p_2.r$ est égal à $-\alpha$, ce qui contredit l'hypothèse selon laquelle $p_1.r$ et $p_2.r$ sont dans $ring_\varphi$ à la date $i+1$.
2. Il existe $q \in \overline{\mathcal{N}_{p_1}} \cup \overline{\mathcal{N}_{p_2}}$ tel que $q.r \in tail_\varphi^*$. Donc, dans γ_{i+1}, il existe un voisin q_1 de q dans $\{p_1, p_2\}$ avec $q_1.r \in tail_\varphi$. Donc, puisque $q_1.r$ est dans $ring_\varphi$ à la date $i+1$, on en déduit que $q_1.r = 0$ dans γ_{i+1} ; cela implique qu'il existe un voisin q_0 de q_1 tel que $q_0.r = -1$ dans γ_i. Ainsi, $(q_0, i) \overset{g}{\rightsquigarrow} (q_1, i+1)$.

□

Le lemme 70 étudie à quelle condition il peut y avoir une réinitialisation dans le DAG de convergence, sans que cette réinitialisation ne soit initiale.

Lemme 70 S'il existe un processus p et deux transitions successives $\gamma_i \mapsto \gamma_{i+1} \mapsto \gamma_{i+2}$ $(i \geq 0)$ tels que $p.r = -\alpha$ dans γ_{i+2}, alors :

1. il existe $q_1 \in \overline{\mathcal{N}_p}$ tel que $q_1.r = 0$ dans γ_{i+1},
2. à la date i, il existe $q_0 \in \overline{\mathcal{N}_{q_1}}$ tel que $q_0.r = -1$ dans γ_i, et

$(q_0, i) \overset{g}{\rightsquigarrow} (q_1, i+1) \overset{g}{\rightsquigarrow} (p, i+2)$.

Preuve : Dans γ_{i+1}, nous avons : $\forall q \in \overline{\mathcal{N}_p},\ q.r \in ring_\varphi$, et il existe $q \in \mathcal{N}_p$ tel que $p.r \neq q.r$. Donc, par le lemme 69, il existe $q_1 \in \{p, q\}$ tel que $q_1.r = 0$ dans γ_{i+1} et $q_0 \in \overline{\mathcal{N}_{q_1}}$ tel que $q_0.r = -1$ dans γ_i.

□

Lemme 71 Pour toute exécution maximale $e = \gamma_0\gamma_1 \ldots \gamma_i \ldots$ $(i \geq 0)$, des états convergents initiaux sont possibles que si $i \in \{0, 1\}$. De plus, si à la date $i > 0$, il n'existe pas d'état convergent dans e, alors le système est dans Γ_0.

Preuve : D'après les lemmes 69 et 70, les états convergents initiaux ne sont possibles que si $i \in \{0, 1\}$. Supposons qu'il n'y ait pas d'état convergent à la date $i > 0$ et qu'il y en a un à la date $j > i$, alors il existe un état convergent initial dans $\gamma_{j'}$, $j' \notin \{0, 1\}$. Cela n'est pas possible à cause du lemme 70. Ainsi, il n'y a pas d'état convergent à partir de la date $i > 0$, ce qui signifie que le système est dans Γ_0.

□

Théorème 72 [Absence d'interblocage] Le système est sans interblocage.

Preuve : La preuve se fait par l'absurde. Soit $e = \gamma_0\gamma_1 \ldots \gamma_i$ $(i \geq 0)$ une exécution maximale finie. Deux cas sont possibles :

1. dans l'état γ_i il n'y a pas d'état convergent, alors par le lemme 71, le système est dans Γ_0 et l'absence d'interblocage est garantie.
2. dans l'état γ_i il y a un état convergent. Soit p un processus parmi les processus en état de convergence dans γ_i, tel que $p.r$ contienne la valeur minimale parmi les valeurs des registres des processus en convergence. Supposons que $p.r \neq 0$ dans γ_i ; alors p peut effectuer l'action (TA) et elle changera la valeur de son registre. Supposons que $p.r = 0$ dans γ_i, alors nécessairement $i > 0$ et tous les processus ont leur registre dans $ring_\varphi$; alors soit le système est à l'unisson et donc non interbloqué, soit au moins un processus peut effectuer l'action de réinitialisation RA ; le système n'est donc pas interbloqué.

□

Lemme 73 Soit $(p_0, i) \overset{g}{\rightsquigarrow} (p_1, i+1) \overset{g}{\rightsquigarrow} ... \overset{g}{\rightsquigarrow} (p_l, i+l)$ un chemin d'états convergents dans le DAG de convergence. Alors, $\gamma_i.p_0.r \equiv \gamma_{i+l}.p_l.r - l \ [\alpha + 1]$.

Preuve : Par récurrence, c'est une conséquence immédiate des lemmes 69 et 70.

□

Lemme 74 Soit p et q deux processus tels que $d(p, q) \leq k$. Si pour p, $p.r = -\beta$ dans γ_i tel que $\beta \leq D$, alors $q.r \leq -\beta + k$ dans γ_{i+k}. En particulier si $k = d$: pour tout $q \in V$ nous avons $q.r \leq -\beta + d \leq 0$.

Preuve : Par récurrence. Si $k = 0$, le lemme est vrai. Supposons que le lemme est vrai pour k. Soit $q \in V$ tel que $d(p, q) = k + 1$. Il existe $q_1 \in V$ tel que $d(p, q_1) = k$ et $d(q_1, q) = 1$. Donc, par l'hypothèse de récurrence, $q_1.r \leq -\beta + k$ dans γ_{i+k}, et donc $q.r \leq -\beta + k + 1$ dans γ_{i+k+1}.

□

Lemme 75 Si $\alpha \geq \beta \geq d$, alors tout chemin dans le DAG de convergence a au plus un état convergent (p, i) tel que $p.r = -\beta$, et, dans ce cas, il n'y a pas de réinitialisation dans le chemin après la date i.

Preuve : Supposons qu'il y ait un chemin dans le DAG de convergence avec deux états convergents avec une valeur de registre égale à $-\beta$. Par le lemme 73, il existe un sous-chemin $(p_0, i) \overset{g}{\rightsquigarrow} (p_1, i+1) \overset{g}{\rightsquigarrow} \ldots \overset{g}{\rightsquigarrow} (p_{\alpha+1}, i+\alpha+1)$ tel que $r_{p_0} = -\beta$ dans γ_i, et $r_{p_{\alpha+1}} = -\beta$ dans $\gamma_{i+\alpha+1}$. Soit $k = i + \beta + 1$. A la date k, $r_{p_{\beta+1}} = -\alpha$.
Par le lemme 74, puisque $\beta \geq d$, tout processus p a la valeur de son registre dans $tail_\varphi$ dans $\gamma_{i+\beta}$. Ainsi, il est impossible qu'il y ait une réinitialisation dans $\gamma_{i+\beta+1}$.

□

Théorème 76 Si $\alpha \geq d$, alors le protocole *SS-MinSU* est convergent vers Γ_0, il stabilise en au plus $d + \alpha$ pulsations, et il n'y a plus de réinitialisation pour $i > d$.

Preuve : Pour tout $e = \gamma_0\gamma_1 \ldots \gamma_i \ldots$ $(i \geq 0)$, tout chemin μ du DAG de convergence ayant une longueur maximale commence dans γ_0 ou dans γ_1. Si μ commence avec une

réinitialisation dans γ_0 ou γ_1, alors sa longueur est au plus α. Supposons maintenant que $(p, 0)$ ne soit pas une réinitialisation. Si $p.r \leq -d$, alors, par le lemme 75, sa longueur est au plus $|p.r|$. Sinon $p.r > -d$, et par le lemme 75, sa longueur est au plus $d + \alpha$, et il y a une réinitialistion, à la date $|p.r| + 1 \leq d$.

□

Si $\alpha = d$, alors le système stabilise en au plus $2d$ pulsations.
Si $\alpha \geq d$, alors le système converge vers l'unisson, mais cet unisson peut être atteint alors que les registres ne sont pas encore à valeurs dans $ring_\varphi$. Il est toujours possible d'insérer, entre $SS\text{-}MinSU$ et l'application de l'utilisateur, l'application Ψ de $\mathcal{X}$ sur $ring_\varphi$ telle que :

$$\Psi : p.r \rightarrow \overline{p.r}[K]$$

alors pour l'utilisateur, le système est à l'unisson. On peut aussi simplement étendre l'ensemble Γ_0 de la manière suivante :

$$\Gamma_0 = \{\gamma \in \Gamma, \forall (p, q) \in V^2, p.r = q.r\}$$

En ce sens, si $\alpha \geq d$ alors le système converge vers Γ_0 en au plus $2d$ pulsations, comme le montre le théorème suivant.

Théorème 77 Si $\alpha \geq d$, alors le protocole $SS\text{-}MinSU$ est convergent vers Γ_0, et il converge en au plus $2d$ pulsations.

Preuve : Si dans le DAG de convergence il n'y a pas de réinitialisation et si tout état convergent initial $(p, 0)$ vérifie $p.r > -d$, alors le temps de convergence est inférieur ou égal à d.
Dans le cas contraire, on définit p et i dans les deux cas suivants :

1. Il n'y a pas de réinitialisation. Soit $i = 0$. Soit p un processus tel que $(p, 0)$ est un état convergent tel que $p.r$ est minimal à la date 0.
2. Il existe au moins une réinitialisation. Soit i la date de la dernière initialisation. Soit p un processus qui est réinitialisé à la date i. On déduit du theoreme 76 que $i \leq d$.

$\forall k \in \{0..d\}$, on définit $V_k = \{q \in V,\ q.r = p.r$ à la date $i + k\}$. Par récurrence, et en suivant le même raisonnement que pour le théorème 66, $\forall k \in \{0..d\},\ V_k \cup B(p, k) \subseteq V_{k+1}$. Donc, puisque $B(p, d) = V,\ V_d = V$, le système est dans Γ_0 à la date $i + d \leq 2d$.

□

5 Conclusion.

Nous avons discuté dans ce chapitre la question de la conception d'un unisson synchrone sur l'arbre et le graphe général. Nous avons établi que tout unisson asynchrone uniforme satisfaisant $K > C_G$ et ordonnancé par un démon synchrone, est une barrière de synchronisation forte auto-stabilisante, en fait un unisson synchrone. La restriction aux unissons uniformes

est importante ; par exemple, les algorithmes d'exclusion mutuelle locale proposés au chapitre 6 sont des algorithmes d'unisson asynchrone (ils satisfont les spécifications de l'unisson asynchrone), mais ils ne convergent pas sous un démon synchrone vers Γ_0, ils demeurent des algorithmes d'exclusion mutuelle locale. Ces algorithmes ne sont pas uniformes. Leur implantation sur chaque processus dépend de l'identité du processus.
Nous avons aussi montré que du protocole proposé en [BPV04] dérive un unison synchrone général sur l'arbre qui est optimal en espace. Le cas $K = 3$ donne un protocole qui est optimal en temps et en espace. Enfin, nous proposons un unisson synchrone sur le graphe général exigeant $K \geq 2$ seulement, et $K + D$ états par processus (où D est un majorant de d, le diamètre du réseau).Il stabilise en au plus $2d$ pulsations seulement. La comparaison des résultats de ce chapitre est présentée dans les tables 4.1 et 4.2 pour l'arbre et pour un réseau quelconque synchrone respectivement.

	Taille de l'horloge (K)	Nombre d'états par processus (S)	Temps de convergence en pire cas (pulsations)
Algorithmes spécifiques			
[HG95]	$K = 3^m (m > 0)$	$S = K$	$\frac{d \times (K-1)}{2}$
[HG95]	$K = 3$	$S = 3$	d
[NV01]	$K \geq 2$	$S = (\Delta + 1)K$	d
Algorithmes généraux			
[ADG91]	$K \geq 2d$	$S = K$	$3d$
Algorithme $SSAU$	$K > 3$	$S = K$	$2d$
Algorithme $SSAU$	$K = 3$	$S = 3$	d
Algorithme $SS\text{-}MinSU$	$K \geq 2$	$S = K + D$	$2d$

TAB. 4.1 – Performances des unissons synchrones sur l'arbre

	Taille de l'horloge (K)	Nombre d'états par processus (S)	Temps de convergence en pire cas (pulsations)
[ADG91]	$K \geq 2d$	$S = K$	$3d$
[CFG92]	$K \geq n^2$	$S = K$	nd
Algorithme $SSAU$	$K \geq C_G + 1$	$S \geq K + T_G - 2$	$CP_G + T_G + d$
Algorithme $SS\text{-}MinSU$	$K \geq 2$	$S = K + D$	$2d$

TAB. 4.2 – Performances des unissons synchrones sur le graphe général

Chapitre 5

Vers un unisson optimal sur l'arbre.

> ...great hopes of program transformations can only be based on what seems to me an underestimation of the logical brinkmanship that is required for the justification of really efficient algorithms.
>
> E.W.Dijkstra "My hopes in computing science", EWD709

Sommaire

1 Introduction

Nous abordons à nouveau la question des unissons synchrones et asynchrones sur les arbres ; d désigne le diamètre de l'arbre considéré. Dans les cas synchrone et asynchrone, nous proposons un unisson indépendant de toute connaissance sur l'arbre, qui converge en d pulsations (respectivement d rondes) et dont l'occupation mémoire en nombre d'états est K, la taille de l'horloge. Ces deux algorithmes, qui exigent simplement que la taille de l'horloge soit impaire, sont particulièrement simples, et se définissent par une seule action. Ils utilisent un ordre local un petit peu plus sophistiqué que pour les systèmes d'incrémentation

définis page 57. Il est curieux de constater que les preuves de correction de ces deux algorithmes sont assez techniques. Elles utilisent la panoplie des concepts développés au chapitre 2, qui prouvent ici leur souplesse. Ainsi l'extrême simplicité de ces deux algorithmes n'est qu'apparente. La difficulté est rendue implicite ; elle réside dans la création des concepts nécessaires à la rédaction de preuves de correction compréhensibles.
Comme dans les chapitres précédents, S est le nombre d'états par processus, n est le nombre de processus et Δ est le degré maximal d'un processus. On trouvera un état de l'art sur la conception d'unissons synchrones sur les arbres dans le chapitre 4. Rappelons simplement que sur l'arbre nous avons exhibé page 121 un algorithme optimal à trois états d'horloge (l'algorithme 10). Il implémente à la fois un unisson synchrone et un unisson asynchrone selon la nature du démon. L'idée du chapitre présent est directement issue de la découverte de l'algorithme 10. Dans cet algorithme, l'ordre local utilisé définit en fait localement un ordre total. L'idée est de généraliser cet ordre à d'autres valeurs d'horloge. Malheureusement, une telle idée ne se généralise pas au graphe général. Introduire un ordre "total local", peut introduire des cycles dans le graphe général et donc ne passe pas du local au global quand la topologie contient au moins un cycle.

1.1 Contribution.

Dans ce chapitre notre contribution est de deux ordres : nous proposons un nouvel unisson synchrone et un nouvel unisson asynchrone sur l'arbre, qui sont optimaux en espace et dont le temps de stabilisation, respectivement le temps de convergence, est majoré par d.
Plus précisément : soit K un entier impair plus grand que 1, et soit M tel que $(K = 2M + 1)$. Soit $\chi = \{0, ..., K-1\}$. Soit $b \in \chi$. Nous définissons un ordre total $\leq_b$ sur χ par la séquence ordonnée $\overline{b-M} \leq_b ... \leq_b b \leq_b ... \leq_b \overline{b+M}$, où $\overline{U}$ est l'unique élément $a \in \chi$ tel que $a \equiv U$ modulo K. En conséquence, grâce à cet ordre $\leq_b$ nous sommes capable de définir une notion de minimum, noté min_b.
Soit $\mathcal{N}_p$ l'ensemble des voisins du processus p et $\mathcal{CN}_p = N_p \cup \{p\}$ le *voisinage fermé* de p. Nous supposons le réseau en arbre, chaque processus p est muni d'un registre d'horloge $p.r \in \chi$, l'algorithme distribué est défini sur chaque processus p par l'action gardée :

$$cond \longrightarrow p.r := \overline{min_{p.r}\{q.r, q \in \mathcal{CN}_p\} + 1}$$

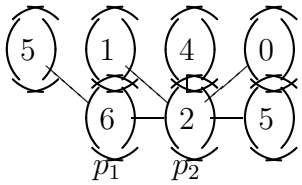


FIG. 5.1 – Un exemple d'arbre de processus, avec la valeur des registres, pour K=7

Nous démontrerons que :

1. Si $cond \equiv vrai$ alors le protocole, cadencé par un démon synchrone, est auto-stabilisant vers une barrière de synchronisation forte (unisson synchrone) en au plus d pulsations.
2. Si $cond \equiv p.r \neq \overline{\min_{p.r}\{q.r, q \in \mathcal{CN}_p\} + 1} \wedge (\forall q \in N_p,\ q.r \leq_{p.r} p.r + 1)$ alors le protocole, cadencé par un démon asynchrone, est un unisson asynchrone qui converge en au plus d rondes vers Γ_1.

Exemple

Considérons l'arbre G défini par la figure 5.1, avec $K = 7$.

Pour le processus p_1 l'ordre total défini par son registre $r = 6$ est $3 <_6 4 <_6 5 <_6 6 <_6 0 <_6 1 <_6 2$.

Pour le processus p_2 l'ordre total est : $6 <_2 0 <_2 1 <_2 2 <_2 3 <_2 4 <_2 5$.

Il est facile de calculer la valeur minimale au voisinage fermé de chacun de ces processus selon l'ordre qu'ils définissent respectivement :

$\min_{p_1.r}\{q.r, q \in \mathcal{CN}_{p_1}\} = \min_6\{5, 2, 6\} = 5$
$\min_{p_2.r}\{q.r, \mathcal{CN}_{p_2}\} = \min_2\{1, 4, 0, 5, 6, 2, \} = 6.$

Il suit que :

$\overline{\min_{p_1.r}\{q.r, q \in \mathcal{CN}_{p_1}\} + 1} = 6$
$\overline{\min_{p_2.r}\{q.r, q \in \mathcal{CN}_{p_2}\} + 1} = 0.$

On trouvera une exécution du protocole dans le cas synchrone sur la figure 5.2.

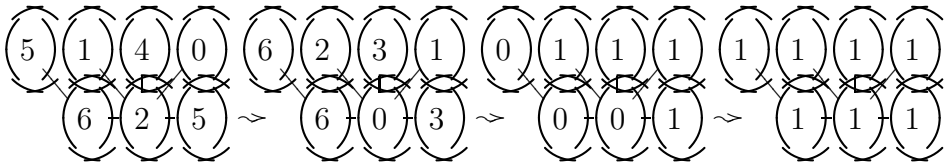


FIG. 5.2 – Une exécution du protocole synchrone sur l'exemple.

Remarque

Quand nous avons commencé cette étude, nous étions persuadés d'avoir défini deux algorithmes répartis auto-stabilisants sans réinitialisation. Ces protocoles ne possèdent pas une règle particulière de réinitialisation locale. En fait ce n'est pas vrai : quand un processus est localement à l'unisson, la règle revient à une incrémentation ; sinon, la règle revient à une correction locale. Les algorithmes que nous proposons sont des algorithmes de réinitialisation au sens proposé page 90. Simplement, le procédé local de correction ne consiste pas à donner une valeur particulière fixée au registre, par exemple $-\alpha$. La question principale est de prouver que ces protocles convergent vers un état de Γ_0, respectivement de Γ_1, en un nombre fini d'étapes.

La définition de chacun des deux protocoles est quasi triviale. Les preuves ne le sont pas. Ils sont optimaux en espace et fonctionnent dans les réseaux dynamiques en arbre. Dans cette famille, seul le cas $K = 3$ a été étudié dans [BPV05] et au chapitre 4 : algorithme 10 page 121. Ce cas est trivial, toute configuration est correcte pour l'unisson asynchrone. Dans le cas synchrone, la stabilisation est une conséquence immédiate du théorème de convergence "$\Gamma_1 \triangleright \Gamma_0$" sous un démon synchrone [BPV05], voir page 117 de ce mémoire. Ce chapitre est une généralisation de ce cas pour toute horloge de phase d'ordre impair. La difficulté est que la méthode de réinitialisation est originale et donc que les méthodes de preuves de [BPV05] ne peuvent être appliquées directement. Un approfondissement des méthodes est nécessaire.

1.2 Organisation du chapitre.

Dans la section 2, nous présentons d'abord les outils algébriques nécessaires dans ce chapitre et nous généralisons les outils développés dans le chapitre 2. Le protocole d'unisson synchrone est prouvé dans la section 3, le protocole d'unisson asynchrone dans la section 4. Enfin, nous ferons quelques remarques dans la section 5.

2 Préliminaires

Dans cette partie, nous généralisons la notion d'ordre local et nous introduisons une notion d'ordre "local total". Nous définissons la notion de variation locale $b \ominus a$ dans ce contexte.

2.1 Relation de comparaison locale

Par un souci de clareté, nous rappelons brièvement quelques notions définies au chapitre 2, ensuite nous généraliserons ces notions dans le contexte de ce chapitre.

1. M est un entier strictement positif et que $K = 2M + 1$.
2. $\bar{a}$ désigne l'unique élément de $[0, K[$ tel que $a \equiv \bar{a}\,[K]$.
3. On définit $d_K(a, b) = \inf\left(\overline{a-b}, \overline{b-a}\right)$, c'est une distance sur $[0, K[$. Dans la suite $\chi = \{0, ..., K-1\}$.

Définition 36 Un *ordre local* sur χ est une relation binaire réflexive et antisymétrique sur χ.

Pour tout entier a et tout entier b, nous avons la relation $d_K(a, b) \leq M$. De cette constatation, nous définissons une relation d'*ordre locale* $\leq_l$ par :

$$a \ \leq_l \ b \Leftrightarrow_{def} 0 \leq \overline{b-a} \leq M.$$

Il est très important de comprendre que cette relation n'est pas *transitive*, elle ne définit pas un *ordre total* sur χ.

Maintenant, comme annoncé, nous définissons un *ordre total* sur χ centré en b et noté $\leq_b$; il est défini en extension par la séquence ordonnée :

$$\overline{b-M} \leq_b ... \leq_b b \leq_b ... \leq_b \overline{b+M}.$$

Cet ordre dépend de b. Le problème est que cet ordre n'est pas toujours accessible "directement" grâce à la relation d'ordre local $\leq_l$. En effet : si a_i et a_j sont deux entiers, ils sont localement comparables à b. Mais comment comparer simplement a_i et a_j selon la relation $\leq_b$? Pour répondre à cette question, on peut remarquer que si :

$$(a_i \leq_l b \text{ et } a_j \leq_l b) \text{ ou } (b \leq_l a_i \text{ et } b \leq_l a_j),$$

alors les ordres entre les deux entiers a_i et a_i selon les relations $\leq_l$ et $\leq_b$ coïncident ; ce qui donne un moyen de calcul local de la relation $\leq_b$. Ces remarques suggèrent la définition effective suivante :

Proposition 78 Si a_i et a_j sont deux entiers dans χ, alors $a_i \leq_b a_j$ si et seulement si une des propositions suivantes est satisfaite :

$$\begin{array}{l} a_i \leq_l b \text{ et } b \leq_l a_j \\ a_i \leq_l b \text{ et } a_j \leq_l b \text{ et } a_i \leq_l a_j \\ b \leq_l a_i \text{ et } b \leq_l a_j \text{ et } a_i \leq_l a_j \end{array}$$

La notion de minimum locale définie par $\leq_b$ est maintenant aisément calculable par chaque processus. Nous noterons $\min_b$ l'opérateur d'infimum défini par l'ordre total $\leq_b$ sur χ.

Proposition 79 Soit a et b deux éléments de χ, alors la relation binaire $\leq_b$ est un ordre total dans χ et nous avons

$$a \leq_a b \Longleftrightarrow a \leq_b b \Longleftrightarrow a \leq_l b$$

Généralisons maintenant la notion de différence locale à l'ordre local $\leq_l$:

Définition 37 Nous définissons $b \ominus a$ par :

$$\text{si } a \leq_l b \text{ alors } b \ominus a =_{def} \overline{b-a} \text{ sinon } b \ominus a =_{def} -\overline{a-b}$$

Clairement, $b \ominus a \equiv b - a \ [K]$. Soit $a_0, a_1, a_2, ...a_\nu$ une séquence d'entiers. La *variation locale* de cette série est la somme : $S = \sum_{i=0}^{\nu-1} (a_{i+1} \ominus a_i)$. Bien sûr $S \equiv a_\alpha - a_0 \ [K]$.

Remarque

La propostion 79 est essentielle. Elle peut paraître triviale au sens qu'elle assure justement que l'intuition que nous pouvons avoir est correcte, encore faut-il prouver cette correction. L'une des origines de la non correction de l'algorithme d'exclusion mutuelle locale défini dans [BDGM00] (voir [BCPV03]) est que la *comparaison cyclique* (que nous appelons ici ordre local) n'est pas correcte, elle

ne satisfait pas l'intuition décrite par la proposition 79. Par exemple, si $B = 8$ (pour nous $K = 8$) : si p et q sont deux processus voisins, que le registre de p est à 7 et le registre de q est à 3, alors on a pour $p : 3 <_7 7$, et pour $q : 7 <_3 3$, ce qui donne une double orientation de l'arête (p, q), ce qui contredit le fait que l'orientation définit un DAG. L'erreur est que l'ordre n'est pas absolu, il est lié à la valeur du registre du processus, donc relatif au processus "regardant", et donc il peut être différent pour le processus p et le processus q. Pour que la notion soit correctement définie, indépendante du processus "regardant", il faut prendre nécessairement B impair, contrairement à ce qu'affirme [BDGM00].

La proposition 79 est une proposition de *cohérence* locale. Elle montre en particulier qu'avec notre construction le phénomène décrit ci-dessus est impossible.

2.2 Variation locale et chemin-compatibilité

Les notions introduites dans le chapitre 2 sont rappelées ou redéfinies dans le contexte de ce chapitre. Le *retard* sur le chemin $c = p_0 p_1 .... p_k$, noté $\triangle_c$, est la *variation locale* de la série $p_0.r, p_1.r, ..., p_k.r$:

$$\triangle_c = \sum_{i=0}^{k-1} (p_{i+1}.r \ominus_l p_i.r)$$

et $\triangle_c = 0$ si la longueur du chemin est 0.
Clairement, la notion de retard est intrinsèque, puisque la topologie du réseau est en arbre. Soit γ_t une configuration, alors $p^t.r$ représente la valeur du registre $p.r$ à la date t, et $\triangle_\mu^t$ représente la variation locale le long du chemin μ à la date t. Soit $\gamma_t \rightarrow \gamma_{t+1}$ une transition. Soit μ un chemin $p_0...p_k$, alors on a la congruence :

$$\triangle_\mu^{t+1} \equiv \triangle_\mu^t + (p_k^{t+1}.r \ominus p_k^t.r) - (p_0^{t+1}.r \ominus p_0^t.r) \ [K]$$

Nous introduisons maintenant la notion de *chemin-compatibilité*.

Définition 38 Un algorithme réparti, cadencé par un certain démon D, est *chemin-compatible* si et seulement si pour tout état $\gamma_t \in \Gamma$, toute transition $\gamma_t \rightarrow \gamma_{t+1}$ cadencée par D et tout chemin $\mu = p_0 p_1 ... p_k$, il y a l'égalité :

$$\triangle_\mu^{t+1} = \triangle_\mu^t + (p_k^{t+1}.r \ominus p_k^t.r) - (p_0^{t+1}.r \ominus p_0^t.r)$$

Cette notion a déjà été rencontrée dans le chapitre 2. Le théorème 14 page 65 montre la chemin-compatibilité de l'incrémentation sur Γ_M, alors la notion était trivilement vérifiée et ne nécessitait pas l'introduction d'une définition particulère. Ici, l'action des processus dans Γ n'est pas l'incrémentation, mais une action plus complexe, d'où la nécessité de "nommer" le concept pour pouvoir travailler clairement.
Redéfinissons dans le contexte de ce chapitre, quelques notions introduites dans le chapitre 2 :

1. Soit γ un élément de Γ. Notons Δ_{pq} la valeur du retard sur le chemin de p à q.
2. On dit que p "*précède*" q dans une configuration γ si et seulement si $\Delta_{pq} \leq 0$. De même, p et q sont "γ-*synchrones*"
3. si $\Delta_{pq} = 0$ dans γ. Comme le réseau est connexe, la relation de précédence définit un préordre sur les processus.
4. Les processus qui sont minimaux pour la relation de précédence sont γ-synchrones. L'ensemble des processus minimaux n'est jamais vide car le réseau est fini.
5. On note V_0 l'ensemble des processus minimaux pour chaque état $\gamma \in \Gamma$.

3 Notre protocole sous le démon synchrone

Soit G un réseau en arbre. $M \in \mathbb{N} - \{0\}$ et $K = 2M + 1$. Si $b \in \chi$, alors la notion de min_b est définie par la proposition 79. L'opération $min_{p.r}\{q.r, q \in \mathcal{CN}_p\}$ est donc bien définie pour chaque processus $p \in V$.

Algorithme 12 (SU_Min) : unisson synchrone sur l'arbre

Action :

SA : $Vrai \longrightarrow p.r := \overline{min_{p.r}\{q.r, q \in \mathcal{CN}_p\} + 1}$;

Remarque

Le lecteur sera peut-être surpris par l'abscence de l'action de l'application dans le protocole défini par l'algorithme 12. Pour alléger, nous avons abstrait cette action. On peut la rétablir simplement de la manière suivante ; on commence par définir le prédicat maintenant classique :

$NormalAction_p \equiv \forall q \in \mathcal{N}_p, (q.r = p.r) \vee (q.r = \varphi(p.r))$

On introduit alors l'action :

if $NormalAction_p$ then << code de l'application >>;

L'action du protocole devient :

SA : $Vrai \longrightarrow$ if $NormalAction_p$ then << code de l'application >> ;
$p.r := \overline{min_{p.r}\{q.r, q \in \mathcal{CN}_p\} + 1}$;

Cette remarque vaut aussi pour l'algorithme 13. Le lecteur vérifiera que ces protocoles sont des unissons, à la condition que l'on puisse prouver la convergence vers Γ_0 (respectivement Γ_1), ce que nous allons faire dans ce chapitre.

Sous un démon synchrone, l'algorithme SU_Min a une bonne propriété : il est *chemin-compatible*. Cette propriété assure la stabilité de l'ensemble des processus minimaux V_0 . Ainsi, pour prouver la convergence vers Γ_0, il est suffisant de prouver que V_0 est strictement croissant jusqu'à ce que $V_0 = V$. En reprenant l'exemple précédent, on peut voir ce phénomène sur la figure 5.3. Nous commencerons par prouver la chemin-compatibilité, ensuite nous prouverons la convergence.

Une conséquence immédiate de la définition du protocole est le lemme suivant :

Lemme 80 Soit $\gamma_t \rightarrow \gamma_{t+1}$ une transition et p un processus. Il existe $\alpha \in \{0, ..., M\}$ tel quet $p^{t+1}.r = \overline{p^t.r - \alpha + 1}$.
Si $q_1 \in \mathcal{CN}_p$ tel que $q_1^t.r = min_{p.r}\{q^t.r, q \in \mathcal{CN}_p\}$, alors $\alpha = p^t.r \ominus q_1^t.r$ et $p^{t+1}.r \ominus p^t.r = -\alpha + 1$.

Dans la suite, on considère la transition $\gamma_t \rightarrow \gamma_{t+1}$. Pour alléger les notations, nous notons $p.r = p^t.r,\ q.r = q^t.r$, $p.r' = p^{t+1}.r,$ and $q.r' = q^{t+1}.r$. D'après le lemme 80, il existe α et β dans $\{0, .., M\}$ tels que $p.r' = \overline{p.r - \alpha + 1}$ et $q.r' = \overline{q.r - \beta + 1}$. Evidemment :

$$q.r' \ominus p.r' \equiv q.r \ominus p.r + \alpha - \beta\,[K]\,.$$

Le coeur de la preuve de convergence consiste à montrer que sous un démon synchrone cette congruence est une égalité. Il y a quatre cas à examiner. Dans les lemmes 81 et 82 nous examinerons les cas : ($q.r \geq_l p.r$, $q.r' \geq_l p.r'$) et ($q.r \geq_l p.r$, $q.r' \leq_l p.r'$), les deux autres cas s'en déduiront par un argument de symétrie (voir preuve de la proposition 83).

Lemme 81 Sous un démon synchrone, si $q.r \geq_l p.r$ et $q.r' \geq_l p.r'$ alors l'égalité $q.r' \ominus p.r' = q.r \ominus p.r + \alpha - \beta$ est vraie.

Preuve : D'après les hypothèses, $q.r \ominus p.r = \overline{q.r - p.r} \in \{0, ..., M\}$ et $q.r' \ominus p.r' = \overline{q.r' - p.r'} \in \{0, ..., M\}$. Mais $q.r' \ominus p.r' = \overline{q.r - p.r - \beta + \alpha}$, alors $\overline{q.r - p.r - \beta + \alpha} \in \{0, ..., M\}$. Comme $q.r \geq_l p.r$, on en déduit que $q.r' \leq_l p.r+1$. Ainsi $\beta \in \{\overline{q.r - p.r}, ..., M\}$, il suit que $\alpha-\beta \in \{-M, ..., M-\overline{q.r - p.r}\}$. Ainsi, $\overline{q.r - p.r}-\beta+\alpha \in \{-M+\overline{q.r - p.r}, ..., M\}$. Supposons que $\overline{q.r - p.r} - \beta + \alpha \in \{-M + \overline{q.r - p.r}, ..., -1\}$ alors $\overline{q.r - p.r - \beta + \alpha} \in \{M + 1 + \overline{q.r - p.r}, 2M\}$, ce qui est impossible car $\overline{q.r - p.r - \beta + \alpha} \in \{0, ..., M\}$. Donc $\overline{q.r - p.r} - \beta + \alpha \in \{0, ..., M\}$. Nous concluons que $\overline{q.r - p.r - \beta + \alpha} = \overline{q.r - p.r} - \beta + \alpha$ et le lemme est prouvé.

□

Lemme 82 Sous un démon synchrone, si $q.r \geq_l p.r$ et $q.r' \leq_l p.r'$ alors l'égalité $q.r' \ominus p.r' = q.r \ominus p.r + \alpha - \beta$ est vraie.

Preuve : Des hypothèses nous déduisons que $q.r \ominus p.r = \overline{q.r - p.r} \in \{0, ..., M\}$ et $q.r' \ominus p.r' = -\overline{p.r' - q.r'} = -\overline{p.r - q.r + \beta - \alpha} \in \{-M, ..., 0\}$. De $q.r \geq_l p.r$, on déduit $\beta \in \{\overline{q.r - p.r}, ..., M\}$ et $\beta - \alpha \in \{-M + \overline{q.r - p.r}, ..., M\}$. De $q.r' \leq_l p.r'$, on déduit $\beta - \alpha \geq \overline{q.r - p.r}$ et $\beta - \alpha \in \{\overline{q.r - p.r}, ..., M\}$. Ainsi, $\overline{p.r - q.r} + \beta - \alpha \in \{\overline{q.r - p.r} + \overline{p.r - q.r}, ..., M+\overline{p.r - q.r}\}$. Si $p.r = q.r$ alors le lemme est vrai, sinon $\overline{q.r - p.r}+\overline{p.r - q.r} = 2M + 1$ et $\overline{p.r - q.r} + \beta - \alpha \in \{2M + 1, ..., M + \overline{p.r - q.r}\}$. Ainsi $\overline{p.r - q.r + \beta - \alpha} = \overline{p.r - q.r}+\beta-\alpha-(2M+1)$. De $\overline{q.r - p.r}+\overline{p.r - q.r} = 2M+1$, On déduit $-\overline{p.r - q.r + \beta - \alpha} = \overline{q.r - p.r} - \beta + \alpha$. Nous concluons que $q.r' \ominus p.r' = q.r \ominus p.r - \beta + \alpha$. Le lemme est prouvé.

□

Proposition 83 Dans les quatre cas, sous un démon synchrone, $q.r' \ominus p.r' = q.r \ominus p.r + \alpha - \beta$.

Preuve : Si $q.r \geq_l p.r$ la preuve a été faite. Si $p.r \geq_l q.r$, il suffit d'échanger les rôles de p et q et de remarquer que $q.r \ominus p.r = -p.r \ominus q.r$. On en déduit la proposition.

□

La proposition suivante est un important corollaire

Proposition 84 [**chemin-compatibilité**] Sous un démon synchrone, l'algorithme *SU_Min* est *chemin-compatible*.

Preuve : Par récurrence sur la longueur du chemin μ.
Si la longueur de μ est égale à 0 la proposition est clairement vraie.
Supposons que la proposition est vraie pour tout chemin $\mu = p_0 p_1 ... p_k$. Soit $\mu_1 = \mu p_{k+1}$ un nouveau chemin :

$$\Delta^t_{\mu p_{k+1}} = \Delta^t_\mu + p^t_{k+1}.r \ominus p^t_k.r \text{ et } \Delta^{t+1}_{\mu p_{k+1}} = \Delta^{t+1}_\mu + p^{t+1}_{k+1}.r \ominus p^{t+1}_k.r \tag{5.1}$$

Soit α tel que

$$p^{t+1}_k.r \ominus p^t_k.r = 1 - \alpha$$

Soit β tel que

$$p^{t+1}_{k+1}.r \ominus p^t_{k+1}.r = 1 - \beta$$

Hypothèse de récurrence : $\Delta^{t+1}_\mu = \Delta^t_\mu + p^{t+1}_k.r \ominus p^t_k.r - p^{t+1}_0.r \ominus p^t_0.r$
Nous avons :

$$\Delta^{t+1}_{\mu p_{k+1}} = \Delta^{t+1}_\mu + p^{t+1}_{k+1}.r \ominus p^{t+1}_k.r. \tag{5.2}$$

Par l'hypothèse de récurrence :

$$\begin{aligned}
\Delta^{t+1}_{\mu p_{k+1}} &= \Delta^t_\mu + p^{t+1}_k.r \ominus p^t_k.r - p^{t+1}_0.r \ominus p^t_0.r + p^{t+1}_{k+1}.r \ominus p^{t+1}_k.r \\
&= \left(\Delta^t_\mu + p^t_{k+1}.r \ominus p^t_k.r\right) + p^{t+1}_{k+1}.r \ominus p^t_{k+1}.r - p^{t+1}_0.r \ominus p^t_0.r + R \\
&= \Delta^t_{\mu\hat{}p_{k+1}} + p^{t+1}_{k+1}.r \ominus p^t_{k+1}.r - p^{t+1}_0.r \ominus p^t_0.r + R
\end{aligned}$$

Avec :
$R = -p^t_{k+1}.r \ominus p^t_k.r + \left(p^{t+1}_k.r \ominus p^t_k.r + p^{t+1}_{k+1}.r \ominus p^{t+1}_k.r - p^{t+1}_{k+1}.r \ominus p^t_{k+1}.r\right)$.
Il reste à montrer que $R = 0$. Par le lemme 83, nous avons :

$$p^{t+1}_{k+1}.r \ominus p^{t+1}_k.r = p^t_{k+1}.r \ominus p^t_k.r + \alpha - \beta$$

D'après la définition de α et de β, nous pouvons dire que :

$$p^{t+1}_k.r \ominus p^t_k.r = 1 - \alpha \text{ et } p^{t+1}_{k+1}.r \ominus p^t_{k+1}.r = 1 - \beta$$

On en déduit que :

$$R = -p^t_{k+1}.r \ominus p^t_k.r + 1 - \alpha + p^t_{k+1}.r \ominus p^t_k.r + \alpha - \beta - 1 + \beta = 0$$

La proposition suit par récurrence.

□

Proposition 85 [**Stabilité de V_0**]
Pour toute transition $\gamma_t \rightarrow \gamma_{t+1}$ cadencée par le démon synchrone : si $p \in V_0$ dans γ_t, alors $p \in V_0$ dans γ_{t+1} et $p^{t+1}.r = \overline{p^t.r + 1}$

Preuve : Avec les mêmes notations qu'au dessus, soit p un processus minimal de γ_t. De la minimalité de p, on déduit que $\Delta^t_{pq} \geq 0$ dans γ_t pour tout processus q de V. p est minimal dans γ_t, donc pour la transition vers γ_{t+1}, on a $\alpha = 0$ et $p^{t+1}.r = \overline{p^t.r + 1}$. L'égalité $\Delta^{t+1}_{pq} = \Delta^t_{pq} - \beta$ est vraie à cause de la chemin-compatibilité.
Si $\beta = 0$ alors $\Delta^{t+1}_{pq} \geq 0$. Si $\beta > 0$ alors il existe $q_1 \in \mathcal{N}_q$ tel que $q^t.r \ominus q^t_1.r = \beta$. Ainsi $\Delta^t_{pq_1} = \Delta^t_{pq} - \beta$ et, par le fait que p est minimal dans γ_t, nous déduisons que $\Delta^t_{pq} \geq \beta$ et que $\Delta^{t+1}_{pq} \geq 0$. La proposition suit.

□

La distance entre deux processus p et q, notée $d\,(p, q)$ est la longueur du plus petit chemin de p à q. Soit $p \in V$, on définit δ_p comme $\max_{q \in V} d(p, q)$. Soit k un entier positif. Définissons $B(p, k)$ comme l'ensemble des processus tels que $d(p, q) \leq k$.

Théorème 86 [**Convergence**] Pour toute exécution commençant en $\gamma \in \Gamma$ et cadencée par un démon synchrone, Γ_0 est un attracteur pour Γ. Le temps de convergence de Γ vers Γ_0 est majoré par d pulsations et cette borne est la meilleure possible.

Preuve : Considérons une exécution infinie $e = \gamma_{t_0}, \gamma_{t_1}, \ldots$. Soit V^t_0 l'ensemble des processus minimaux à la date t. Soit p un élément de $V^{t_0}_0$ avec la quantité $\delta_p = w$ minimale.
On montre par récurrence que :

$$\forall i \in \mathbb{N}, B(p, i) \subseteq V^{t_0+i}_0.$$

Cela prouvera que pour $i = w$, alors $V^{t_0+i}_0 = V$. Ainsi, le temps de convergence est inférieur ou égal à w, qui est majoré par d.
Initialisation : pour $i = 0$, on a $p \subseteq V^{t_0}_0$.
Hérédité : faisons l'hypothèse que $i \geq 0$ et que $B(p, i) \subseteq V^{t_0+i}_0$. Soit $q \in B(p, i + 1)$. Alors q est un voisin d'un certain q' appartenant à $B(p, i)$. Il y a deux cas :
1. $q \in V^{t_0+i}_0$. Par le lemme 85, alors $q \in V^{t_0+i+1}_0$.
2. $q \notin V^{t_0+i}_0$. Alors, $q'.r = min_{q.r}\{s.r, s \in \mathcal{CN}_p\}$ et grâce au lemme 85, $q \in V^{t_0+i+1}_0$.
Dans les deux cas, $q \in V^{t_0+i+1}_0$. Nous en déduisons que $B(p, i + 1) \subseteq V^{t_0+i+1}_0$.
Par récurrence, la première partie du théorème est prouvée, c'est-à-dire que :

$$\forall i \in \mathbb{N}, B(p, i) \subseteq V^{t_0+i}_0$$

.
Soit p à l'extrémité d'un diamètre de G ($\delta_p = \max_{q \in V} \delta_q$). Supposons que $p.r = 0$ dans γ_0. On définit le contenu de chaque processus q de la manière suivante : $d(q, p) = \alpha$

($\alpha \in \{1, ..., D\}$), $q.r = \overline{\alpha}$. Partant d'une telle configuration γ_0, Γ_0 est atteint en au moins d pulsations. Nous avons montré que d peut être atteint, et la deuxième partie du théorème est prouvée.

□

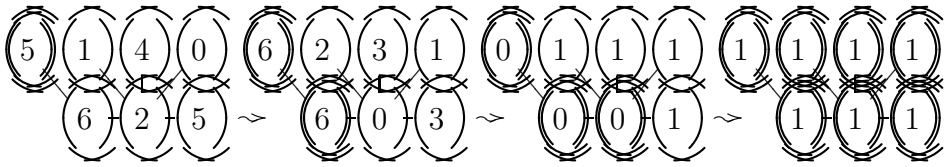


FIG. 5.3 – Croissance de V_0 sur l'exécution de l'exemple page 131

4 Notre protocole sous un démon asynchrone

L'algorithme réparti défini dans la section précédente n'est pas *chemin-compatible* pour un démon asynchrone. Un contre-exemple est donné sur la chaîne de la figure 5.4 par la transition asynchrone $\gamma_t \to \gamma_{t+1}$ présentée sur le schéma 5.5.

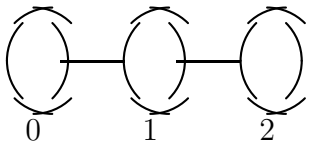


FIG. 5.4 – Une chaine de trois processus

Soit $\mu = p_0 p_1 p_2$; p_0 et p_2 sont invariants. Mais $\triangle_\mu^t = 2M$ et $\triangle_\mu^{t+1} = -1$. Ces quantités ne sont pas égales, mais seulement congruentes modulo K (rappelons que $K = 2M + 1$).

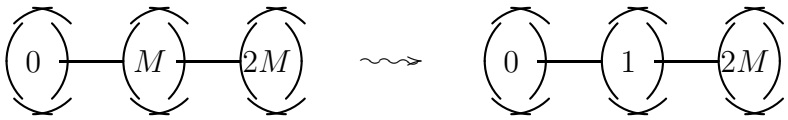


FIG. 5.5 – Violation de la chemin-compatibilité sous un démon asynchrone

4.1 Le protocole

Comme nous venons de le voir, la preuve de correction dans le cas synchrone ne peut nous venir en aide à cause de l'absence de chemin-compatibilité. Il est aussi facile de voir que dans le cas d'un démon asynchrone, l'ensemble V_0 n'est pas stable. Comment transformer le protocole et comment envisager une preuve de convergence vers Γ_1 dans le cas asynchrone ? Dans le cas synchrone, le prédicat de la garde est *vrai*. D'abord, pour éviter de considérer des actions qui consiste au fond à ne rien faire et d'éviter de travailler sous un démon faiblement équitable, on remplace *vrai* par la condition :

$$p.r \neq \overline{min_{p.r}\{q.r, q \in \mathcal{CN}_p\} + 1}$$

Mais cela n'est pas suffisant. Si on conserve seulement cette condition, il y a encore des possibilités de famine. Pour vérouiller, nous ajoutons une deuxième condition qui introduit une priorité entre les ajustements locaux. Cette condition s'énonce par :

$$\forall q \in N_p, q.r \leq_{l_p} p.r + 1.$$

On peut remarquer que cette condition est satisfaite dans Γ_1, qui restera donc stable. Ainsi, pour toutes ces raisons, notre prédicat $Move_p$ sera la conjonction de deux conditions énoncées ci-dessus. L'algorithme 13 définit notre protocole sous un démon asynchrone.

Algorithme 13 Algorithme réparti WU_Min pour le processus p

Prédicat :

$Move_p \equiv (p.r \neq \overline{\min_{p.r}\{q.r, q \in \mathcal{CN}_p\} + 1}) \wedge (\forall q \in N_p,\ q.r \leq_p p.r + 1)$

Action :

$SA : \quad Move_p \quad \longrightarrow \quad p.r := \overline{min_{p.r}\{q.r, q \in \mathcal{CN}_p\} + 1}\,;$

Le prédicat $Move_p$ est équivalent au prédicat :

$$\forall q \in \mathcal{N}_p : \ (q.r = p.r) \vee (q.r = \overline{p.r + 1})$$

.
de plus, quand le prédicat $Move_p$ est satisfait, nous avons l'égalité :

$$\overline{min_{p.r}\{q.r, q \in \mathcal{CN}_p\} + 1} = \overline{p.r + 1}$$

On en déduit que, mise à part la convergence, les propriétés de sûreté et de vivacité de l'unisson asynchrone sont satisfaites (voir remarque page 3). Il reste à prouver la convergence : Vrai $\rhd \Gamma_1$.

4.2 Vivacité et convergence vers Γ_1

Comme le protocole WU_Min n'est pas chemin-compatible, nous ne pouvons pas utiliser la notion de retard. Il faut être plus fin. Nous allons utiliser le rasoir d'Ockham pour abstraire les actions d'incrémentation et ne garder que les actions d'ajustement (réinitialisation) qui sont seules susceptibles de disparaître (voir remarque 131).
Nous lions les actions qui ne sont pas des incrémentations normales par une relation causale, le *DAG d'ajustement* . Quand une action n'est pas une action d'incrémentation, c'est une action d'*ajustement*. Précisons cette notion formellement :

Définition 39 Soit $e = \gamma_0\gamma_1...\gamma_k....$ une exécution maximale de l' algorithme WU_Min. Un *ajustement* est une paire ordonnée (p, t) où p est un processus et t une date telle que l'une des propositions suivantes est vérifiée :

1. $t = 0$
2. $t > 0$ et p exécute une action SA à la date t avec $\alpha \neq 0$; précisément :

$$p^t.r \ominus p^{t-1}.r \neq 1$$

(voir lemme 80).

Si $t > 0$, on dit que p s'*ajuste* à la date t.

Nous prouvons maintenant la propriété de vivacité. puis nous prouverons la convergence vers Γ_1.

Théorème 87 [Vivacité] Sur un arbre, WU_Min est sans interblocage.

Preuve : Soit $\gamma \in \Gamma$ une configuration ; nous définissons à partir de γ une relation binaire sur l'ensemble V des processus par :

$$\forall\,(p,q) \in V^2, p \hookrightarrow_\gamma q \Leftrightarrow_{def} \begin{cases} q \in N_p \\ p.r <_p q.r - 1 \end{cases} \tag{5.3}$$

Remarquons que si $p \hookrightarrow_\gamma q$ alors $p.r <_p q.r - 1$, et de la proposition 79 nous déduisons $p.r <_q q.r - 1$ et, en conséquence, la relation $q \hookrightarrow_\gamma p$ n'est pas vraie. Cette relation binaire ne peut pas avoir de *bégaiement*. Elle définit une orientation qui est acyclique sur V, donc un DAG sur V.

Soit $p_1 \hookrightarrow_\gamma p_2 \hookrightarrow_\gamma .... \hookrightarrow_\gamma p_k$ un chemin maximal sur ce DAG. Si $k = 1$ alors $\gamma \in \Gamma_0$ et γ n'est pas une configuration d'interblocage. Si $k > 1$ alors p_k est activable pour une action d'ajustement. La proposition suit.

□

Nous introduisons maintenant la notion de *DAG d'ajustement*. Cette structure contient toute l'information sur la propagation des ajustements sur l'arbre. L'objectif est de prouver que durant une exécution maximale, cette structure est finie.

Lemme 88 Si (p, t_1) est un ajustement vérifiant $t_1 > 0$, alors il existe $q \in N_p$ et $t_0 < t_1$ tels que :

1. $p^{t_1}.r \ominus q^{t_1-1}.r = 1$;
2. $\forall \tau \in [t_0, t_1[\,, q^{\tau+1}.r = q^\tau.r$ ou $q^{\tau+1}.r = \overline{q^\tau.r + 1}$;
   i.e. il n'y a que des actions d'incrémentation pour le processus q durant la période $]t_0, t_1]$.
3. $t_0 = 0$ ou q ajuste à la date t_0.

Preuve : Soit (p, t_1) un ajustement, vérifiant $t_1 > 0$. Par définition, il existe un voisin q de p tel que $p^{t_1}.r \ominus q^{t_1-1}.r = 1$. Nous avons prouvé la première affirmation.

Soit t_0 la plus petite date telle que :

$$\forall \tau \in [t_0, t_1[\,, q^{\tau+1}.r = q^\tau.r$$

ou $q^{\tau+1}.r = \overline{q^{\tau}.r + 1}$; t_0 est bien défini. Alors soit $t_0 = 0$ soit $t_0 > 0$. Dans ce dernier cas, à cause de la minimalité de t_0, q ajuste à la date t_0, ce qui prouve la troisième propriété.

□

Remarquons que dans le lemme 88, q n'est pas nécessairement unique.

Définition 40 Soit (p_0, t_0) et (p_1, t_1) deux *ajustements*.
Nous disons que (p_0, t_0) ajuste (p_1, t_1) , et nous le notons $(p_0, t_0) \rightsquigarrow (p_1, t_1)$ si et seulement si :

1. $t_0 < t_1$,
2. $\forall \tau \in [t_0, t_1[\,, p_0^{\tau+1}.r = p_0^{\tau}.r$ ou $p_0^{\tau+1}.r = \overline{p_0^{\tau}.r + 1}$;
   i.e. p effectue au plus une action d'incrémentation durant la période $[t_0, t_1[$
3. $p_1^{t_1}.r \ominus p_0^{t_1-1}.r = 1$.

De cette définition et du lemme 88, on obtient la proposition suivante :

Proposition 89 Pour tout ajustement (p_1, t_1) avec $t_1 > 0$, il existe un ajustement (p_0, t_0) tel que $(p_0, t_0) \rightsquigarrow (p_1, t_1)$.

Puisque $(p_0, t_0) \rightsquigarrow (p_1, t_1)$ implique $t_0 < t_1$, la relation $\rightsquigarrow$ définit un graphe acyclique orienté (DAG) appelé *DAG d'ajustement*. Un ajustement (p_1, t_1) n'est pas généré par un autre si et seulement si $t_1 = 0$.
Pour prouver la convergence de l'algorithme réparti WU_Min, il suffit de prouver que le *DAG d'ajustement* est toujours fini. Pour atteindre cet objectif, nous devons prouver que ce DAG n'a pas de bégaiement (cf. page 99), c'est-à-dire qu'il n'y a pas de chemin de la forme : $(p_0, t_0) \rightsquigarrow (p_1, t_1) \rightsquigarrow (p_0, t_2)$.

Proposition 90 Si $(p_0, t_0) \rightsquigarrow (p_1, t_1)$ alors, pour tout $t \in]t_0, t_1[$, ni p_0 ni p_1 ne font une action et à la date t_1, seul p_1 fait une action, qui est un ajustement ; précisément :

1. $\forall \tau \in [t_0, t_1[\,, p_0^{\tau+1}.r = p_0^{\tau}.r$ et $\forall \tau \in]t_0, t_1[\,, p_1^{\tau}.r = p_1^{\tau-1}.r$;
2. $p_1^{t_1} r \ominus p_0^{t_0}.r = 1$.

Preuve : Supposons que $(p_0, t_0) \rightsquigarrow (p_1, t_1)$. La question est : que se passe-t-il durant la période $[t_0, t_1]$?
Soit t_0' la date la plus petite telle que durant la période $]t_0', t_1[$, p_0 et p_1 ne font aucune action. Nous avons :

$$t_0 \leq t_0'.$$

Si $t_0 = t_0'$ la proposition est prouvée. Supposons maintenant que $t_0 < t_0'$. Par hypothèse, p_0 ou p_1 fait une action à la date t_0'. En fait, il n'y a que deux actions possibles pour p_0 et p_1. Remarquons, à cause de la condition supplémentaire, que si un processus p s'incrémente à la date $t_0 + 1$, alors à la date t_0 nous avons : $\forall q \in \mathcal{N}_p, q.r \in \left\{p.r, \overline{p.r + 1}\right\}$. Nous discutons suivant p_0 à la date t_0' :

1. p_0 ne fait pas d'action. Alors p_1 fait une action et il y a deux cas :

(a) Si p_1 s'incrémente à la date t'_0 alors, puisque p_0 ne fait aucune action, p_1 est correct avec p_0 à cette date et $p_1.r \in \{p_0.r, p_0.r+1\}$. Comme ensuite p_1 et p_0 ne font aucune action jusqu'à la date t_1, p_1 ne peut pas s'ajuster à partir de p_0 à la date t_1. Ainsi ce cas est impossible.

(b) Si p_1 s'ajuste à la date t'_0, alors à cette date $p_1.r \leq_{p_1.r} \overline{p_0.r+1}$ et donc p_1 ne peut pas s'ajuster à partir de p_0 à la date t_1. Ce cas est donc impossible.

2. p_0 s'incrémente à la date t'_0. Il y a trois cas à étudier :

(a) Si p_1 ne fait rien, alors à cette date : $p_0.r \in \{p_1.r, p_1.r+1\}$. Comme il n'y a aucune action pour p_1 et p_0 jusqu'à la date t_1, p_1 ne peut pas s'ajuster à partir de p_0 à la date t_1. Ce cas est donc impossible.

(b) Si p_1 s'incrémente, alors à la date t'_0 nous avons $p_1.r = p_0.r$, ce qui est impossible.

(c) Si p_1 s'ajuste, remarquons d'abord que, parce que p_0 s'incrémente à la date t'_0, à la date $t'_0 - 1$, nous avons :
$p_0^{t'_0-1}.r \in \left\{p_1^{t'_0-1}.r - 1, p_1^{t'_0-1}.r\right\}$
et à la date t'_0 :
$p_0^{t'_0}.r \in \left\{p_1^{t'_0-1}.r, p_1^{t'_0-1}.r + 1\right\}$
Remarquons maintenant que, puisque p_1 s'ajuste à partir de p_0 :

$$p_1^{t'_0}.r \in \left\{p_1^{t'_0-1}.r - M, ..., p_1^{t'_0-1}.r\right\}$$

On en déduit qu'à la date $t'_0 - 1$, nous avons $p_1.r \leq_l p_0.r$. Une fois encore, comme p_1 et p_0 ne font aucune action jusqu'à la date t_1, p_1 ne peut pas s'ajuster à partir de p_0 à la date t_1. Ce cas est donc impossible.

3. p_0 s'ajuste : c'est la seule possibilité. Par définition de la relation d'ajustement, on en déduit que $t_0 = t'_0$.

De plus, à la date t_1, p_0 est bloqué par p_1, donc p_0 ne fait aucune action à la date t_1.
La proposition est démontrée.

□

Une conséquence immédiate est le lemme suivant :

Lemme 91 Le *DAG d'ajustement* est sans *bégaiement*.
Si $(p_0, t_0) \rightsquigarrow (p_1, t_1)$ est un ajustement, alors $]t_0, t_1]$ est inférieur ou égal à une ronde.

Théorème 92 [Autostabilisation] Le *DAG d'ajustement* est fini, et le protocole *WU_Min* est convergent vers Γ_1 ; le temps de convergence est inférieur ou égal à d rondes.

Preuve : Soit $(p_0, t_0) \rightsquigarrow (p_1, t_1) ... \rightsquigarrow (p_k, t_k) ...$ un chemin maximal dans le *DAG d'ajustement*. Alors $t_0 = 0$; comme il n'y a pas de bégaiement, et comme G est un arbre, la

longueur du chemin est au plus d. Ainsi, le DAG est fini, et après d rondes, il n'y a plus d'ajustements et le système est dans Γ_1.

□

Remarque

D'après [BPV05] ou le théorème 65 (ch. 4 ,p. 117), le deuxième algorithme converge vers Γ_0 sous un démon synchrone, mais on n'est pas assuré qu'il converge en au plus d pulsations, mais seulement en au plus $2d$ pulsations.

Il est possible de donner une preuve de convergence du premier algorithme (algorithme 12) vers Γ_0 en utilisant une technique similaire à la preuve de convergence du deuxième algorithme (algorithme 13). Malheureusement cette méthode de preuve ne semble pas être capable de donner une estimation du temps de convergence suffisamment fine. Avec cette approche, on obtient seulement comme majoration $2d$ et non d. C'est pourquoi l'approche mathématique utilisée pour l'analyse de chacun des deux algorithmes est différente.

5 Conclusion

Nous avons proposé dans ce chapitre deux algorithmes distribués d'unisson sur les arbres. Le protocole SU_Min, cadencé par le démon synchrone, est un unisson synchrone autostabilisant en au plus d pulsations. Le protocole WU_Min, cadencé par un démon asynchrone, est convergent en au plus d rondes. De plus, les deux protocoles sont optimaux en espace. La comparaison des résultats de ce chapitre avec les autres algorithmes déjà connus est présentée dans les tableaux 5.1 et 5.2 pour un réseau en arbre synchrone, et asynchrone respectivement.

Nous avons de bonnes raisons de penser que ces protocoles ne peuvent pas s'étendre au graphe général, à cause des cycles et de l'impossibilité de prolonger de manière cohérente l'ordre "total local" proposé. Une dernière question intéressante est de savoir si les deux algorithmes présentés sont optimaux en temps, et dans quel sens.

	Ordre de l'horloge (K)	Nombre d'états par processus (S)	Temps de convergence en pire cas (pulsations)
Algorithmes spécifiques			
[HG95]	$K = 3^m (m > 0)$	$S = K$	$\frac{d(K-1)}{2}$
[HG95]	$K = 3$	$S = K$	d
[NV01]	$K \geq 2$	$S = (\Delta + 1)K$	d
Algorithme SU_Min	$K = 2M + 1;\ M > 0$	$S = K$	d
Algorithmes généraux			
[ADG91]	$K \geq 2d$	$S = K$	$3d$
Algorithme $SSAU$ [BPV04]	$K > 3$	$S = K$	$2d$
Algorithme $SSAU$ [BPV04]	$K = 3$	$S = 3$	d
Algorithme $SS\text{-}MinSU$ [BPV05]	$K \geq 2$	$S = K + d$	$2d$

TAB. 5.1 – Performances d'unissons sur un arbre synchrone

	Ordre de l'horloge (K)	Nombre d'états par processus (S)	Temps de convergence en pire cas (rondes)
[CFG92]	$K \geq n^2$	$S = K$	$O(nd)$
Algorithme $SSAU$ [BPV04]	$K \geq 3$	$S \geq K$	$2 + 2d$
Algorithme WU_Min	$K = 2M + 1;\ M > 0$	$S = K$	d

TAB. 5.2 – Performances d'unissons sur un arbre asynchrone

Troisième partie

Synchroniser

Chapitre 6

Synchronisation locale

... the "problem of the Dining quintuple"...
The global subspace remains in existence and we have indeed actions taking place with regard to it : the critical section in the person behaviour, mutually excluded by the P and V operation on the binary semaphore...

E.W. Dijkstra, 1967, EWD 198

Sommaire

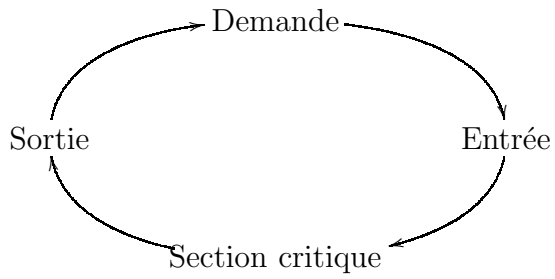


FIG. 6.1 – Cycle d'allocation de ressource d'un processus

1 L'allocation locale de Ressources

1.1 Introduction

Dans ce chapitre nous montrons comment l'unisson peut être utilisé comme protocole sous-jacent pour résoudre efficacement de nombreux problèmes de synchronisation locale.
Un cadre élégant et général de formulation de nombreux problèmes de synchronisation locale est celui de *l'allocation locale de ressources*, encore appelé le problème de l'ALR [CDP03]. L'ALR autorise deux processus voisins à accéder à une ressource (est donc à entrer en section critique) en même temps si les deux ressources demandées ne sont pas en conflit l'une avec l'autre. Nous montrerons que ce cadre permet de formuler des problèmes classiques comme l'exclusion mutuelle locale, l'exclusion mutuelle de groupe locale, les Lecteurs-écrivain locaux ou encore le problème des philosophes buveurs locaux. On peut en particulier utiliser l'ALR pour écrire un transformateur auto-stabilisant transformant un algorithme écrit pour un démon central en un algorithme utilisant une hypothèse moins forte, par exemple un démon inéquitable (Voir la discussion page 44).

1.2 Présentation du problème

On suppose que chaque processus suit un cycle composé de quatre phases (voir figure 6.1) :

une phase d'attente, appelé section *non critique*,
une phase d'*entrée en section critique*,
une phase éxecution d'un code d'application utilisant une ressource partagée : la *section critique*,
et une phase *sortie de section critique*.

On suppose que le partage des ressources par deux processus voisins est soumis à des contraintes de compatibilité :

> **Définition 41** [Partage de ressources] Deux ressources X et Y sont dites **partageables**, noté $X \rightleftharpoons Y$, si deux processus voisins peuvent accéder aux ressources X et Y simultanément. Sinon, X et Y sont dits **être en conflit**, ce qui est noté $X \not\rightleftharpoons Y$.

la définition précédente fait l'hypothèse (implicite) que la relation $X \rightleftharpoons Y$ est symétrique, c'est la seule hypothèse que nous ferons à priori. Durant le passage d'un processus en section

critique, ce processus est supposé exécuter un code spécifique de la couche d'application. Un processus ne peut accéder à une certaine ressource qu'en section critique. On suppose que les processus exécutent leur section critique en un temps fini mais inconnu. On peut voir le problème de l'ALR comme un problème de partage de ressources entre des processus voisins. Plus formellement, le problème de l'ALR est de définir un protocole pour les phases d'entrée et de sortie de la section critique qui satisfasse la spécification suivante lors de toute exécution :

Spécification 10 [Allocation locale de ressources]
Absence de conflit de ressources (Sureté) : Si deux processus voisins p et q exécutent leur section critique simultanément en utilisant les ressources X et Y respectivement, alors $X \rightleftharpoons Y$.
Accès à la ressource garanti (Vivacité) : Si un processus p demande à entrer en section critique, alors p finit par exécuter sa section critique.

Remarque

Le problème de l'ALR se résout facilement avec un algorithme d'exclusion mutuelle locale. Il est intéressant de remarquer que réciproquement l'*exclusion mutuelle locale* est une instance de l'allocation locale de ressources. Dans cette instance, l'ensemble des ressources est l'ensemble des processus et la relation de compatibilité est la relation d'égalité sur les processus.

Pour résoudre le problème de l'ALR, il suffit donc de résoudre l'instance de l'exclusion mutuelle locale. Mais notre objectif est aussi d'assurer le maximum de concurrence. C'est-à-dire, par exemple, de ne pas empêcher que deux processus voisins exécutent simultanément leur section critique s'ils demandent des ressources partageables. Et cela de manière auto-stabilisante.

2 Le protocole

2.1 Le principe

Nous avons défini une famille d'unissons instantanément stabilisants dans le chapitre 3. L'idée est d'utiliser un de ces unissons pour construire une solution auto-stabilisante du problème de l'ALR. Pour cela, nous allons modifier la garde de l'entrée en section critique de notre unisson pour organiser la synchronisation, c'est-à-dire assurer la condition de sûreté de problème de d'ARL traité. La question cruciale sera de prouver qu'on ne produit pas d'interblocage ni de famine par cette modification.
Nous appellerons le nouveau protocole SS_ALR. A priori nous ne faisons pas d'hypothèse sur le réseau. Beaucoup de problèmes classiques d'allocation de ressources exigent que chaque processeur ait une identité différente des autres processeurs ; en allocation locale de ressources nous verrons qu'une coloration locale du réseau est suffisante (deux processus

voisins ont une couleur (identité) différente). Mais il peut arriver qu'un ordre sur les ressources suffise et que les ressources soient indépendantes des processus, alors la possibilité de travailler dans un réseau anonyme devient réaliste.
Dans le cas général, on prendra le protocole SS_GAU (algorithme 7, chapitre 3). Le schema de protocole est décrit par l'algorithme 14. Dans le cas d'un réseau avec identités, on pourra aussi utiliser le protocole SS_RU (algorithme 24, annexeA).

2.2 Le schéma de programme

L'action gardée d'entrée en section critique de l'unisson pour le processus p s'écrit :

$$NormalStep_p \rightarrow << \text{ section critique } >>; p.r := \varphi\,(p.r)$$

il est transformée en :

$$NormalStepTask_p \rightarrow << \text{ section critique } >>; p.r := \varphi\,(p.r)$$

L'algorithme 14 présente le schéma de programme. Seuls, les variables, les prédicats et les actions gardées spécifiques sont présentés. L'utilisateur complétera le schéma de programme avec l'unisson qu'il aura choisi. Le prédicat $Prior_p(q)$ est défini en fonction de la condition de sûreté.

Algorithme 14 Schema de programme SS_ALR

Constante :
$\mathcal{N}_p$: Ensemble des voisins de p ;
Variables :
$p.r \in \mathcal{X}$; $p.v : \mathbb{V}$ (* $\mathbb{V}$ n'importe quel type de donnée *)
Prédicats :
$Prior_p(q) \equiv$ (* Dépend de la condition de sûreté *)
$Task_p \equiv \forall q \in \mathcal{N}_p : p.r = q.r \Rightarrow Prior_p\,(q)$
...
$NormalStepTask_p \equiv NormalStep_p \wedge Task_p$
Actions :
NA : $NormalStepTask_p \longrightarrow << \text{ section critique } >> ; p.r := \varphi(p.r)$;
...

Le problème est de définir le prédicat $NormalStepTask_p$. Si on veut béneficier de la structuration apportée par l'unisson, il est nécessaire d'avoir l'implication :

$$NormalStepTask_p \Rightarrow NormalStep_p$$

Cette implication garantit que le protocole est convergent vers Γ_M. En revanche, rien ne garantit qu'une exécution n'entraîne pas d'interblocage. Mais si nous pouvons prouver que le prédicat $NormalStepTask_p$ n'entraîne pas d'interblocage, du fait du préordre défini par l'unisson, le protocole SS_ALR est alors sans famine et définit un protocole d'unisson.

Il reste à définir le prédicat $NormalStepTask_p$. Pour plus de clareté, nous décomposons le prédicat $NormalStepTask_p$ en la conjontion :

$$NormalStepTask_p \equiv NormalStep_p \wedge Task_p$$

Il reste à définir le prédicat $Task_p$ et de montrer qu'il assure l'absence d'interblocage ainsi que la condition de sûreté de l'ALR considérée.

2.3 Variables et prédicats

1. L'horloge : chaque processeur p maintient un registre $r \in \chi$, où χ est le domaine d'un système d'incrémentation (χ, φ) de signature (K, α). L'horloge de l'unisson sous-jacent est ainsi définie. K et α sont choisis de manière à ce que l'unison soit auto-stabilisant.
2. La requête : chaque processeur p maintient une autre variable v à valeurs dans un ensemble $\mathbb{V}$. Cet ensemble est soit l'ensemble des ressources, soit un produit cartésien de l'ensemble des ressources avec l'ensemble des valeurs d'un système d'identification des processus. La ressource demandée par le processus est accessible par le champ res dela variable v. L'identité du processus, si elle est définie, est accessible dans le champ Id de la variable v, c'est un simple pointeur vers le système d'identification du processus. On suppose $\mathbb{V}$ muni d'une relation binaire $\lhd$.
3. Le prédicat : on utilise un nouveau prédicat $Prior_p(q)$ défini quelque soit $q \in \mathcal{N}_p$ en fonction de $p.v$ et de $q.v$ par :

$$Prior_p(q) \equiv p.v \lhd q.v$$

Voici le principe de notre protocole : si le processus q est en avance sur le processus p, alors p est prioritaire. La configuration délicate est le cas $p.r = q.r$, dans ce cas c'est le prédicat $Prior_p(q)$ qui établira si p est prioritaire par rapport à q. Les deux processus peuvent être prioritaires simultanément si les ressources sont partageables. Pour éviter toute circularité, nous imposons que $\lhd$ définisse un préordre sur $\mathbb{V}$. Nous définissons donc le prédicat $Task_p$ par :

$$Task_p \equiv (\forall q \in \mathcal{N}_p : p.r = q.r \Rightarrow Prior_p(q))$$

2.4 Absence d'interblocage et vivacité

Dans la définition de l'ALR, il n'est fait aucune hypothèse sur la relation de partage de ressources sinon la symétrie. L'ALR est donc une notion très générale. La relation de partage de ressources n'ayant aucune structure à priori, il est possible rencontrer des situations de symétrie où aucun choix de priorité ne peut être fait si l'on tient uniquement compte de la relation de partage sur les ressources.
Pour deux processus voisins p et q, la relation $\lhd$ doit être compatible avec la relation de partage des ressources, c'est-à-dire qu'elle doit vérifier l'implication :

$$(p.v \lhd q.v) \wedge (q.v \lhd p.v) \Rightarrow p.v.res \rightleftharpoons q.v.res \tag{6.1}$$

Théorème 93 Etant donnés un problème de ALR et l'ensemble $\mathbb{V}$. Si la relation $\lhd$ est un préordre total sur $\mathbb{V}$, compatible avec la relation $\rightleftharpoons$, alors le protocole 14 est auto-stabilisant pour les spécifications du problème de ALR donné.

Preuve : $\lhd$ est un préordre total sur l'ensemble des valeurs $\mathbb{V}$ de la variable v. Ce préordre total est compatible, par construction, avec la relation de partage des ressources. La sûreté est donc garantie. Sur l'ensemble des processus minimaux pour la relation de précédence définie par l'unisson, celui qui sera minimal pour la relation $\lhd$ pourra rentrer en section critique. Notre algorithme est dans ces conditions sans interblocage et, grâce au cadencement de l'unisson, il sera sans famine. C'est-à-dire que notre protocole satisfera les spécifications attendus.

□

2.5 Discussion sur la définition de la relation $\lhd$

Une solution paresseuse consiste à travailler sur un réseau avec une identité différente sur chaque processus. Dans ce cas, la relation $\lhd$ définie par :

$$v_p \lhd v_q \equiv v_p.Id \leq v_q.Id$$

convient et le problème est résolu. Cela revient à programmer la plus contraignante des relations de partage, celle définie par l'exclusion mutuelle locale, c'est-à-dire aucun partage local de ressources.
Dans cette partie, on se pose deux questions : à quelle condition ne peut-on pas se passer d'un système d'identification des processus ? Et dans le cas, l'identification doit-elle être différente pour chaque processus ?

2.5.1 Identification chromatique

Définition 42 On dit qu'un système d'identités sur un réseau est chromatique si et seulement si chaque processus a une identité, et que les identités de deux processus voisins sont différentes.

Un réseau avec identités est un réseau avec identités chromatiques, la réciproque est fausse. Il est clair que, dans un réseau avec identités chromatiques, la relation $\lhd$ définie par :

$$v_p \lhd v_q \equiv v_p.Id \leq v_q.Id$$

reste un préordre et répond à la question posée. Il programme une exclusion mutuelle locale, et donc satisfera aux spécifications du problème de ALR considéré. Cette solution est triviale et , comme dans la solution précedente, n'assure pas en général, le maximum de concurrence. Notre problème est d'augmenter la concurrence, ce qui revient à définir plus finement la relation $\lhd$.

2.5.2 Condition nécessaire et suffisante d'utilisation d'identités

Proposition 94 La relation $\rightleftharpoons$ se prolonge en un préordre compatible sur les ressources si et seulement si la relation $\rightleftharpoons$ est une relation d'équivalence sur l'ensemble des ressources.

Preuve : Si la relation de partage de ressources est une relation d'équivalence, il suffit de définir un ordre sur l'ensemble des classes d'équivalence et de relever cet ordre en un préordre sur l'ensemble des ressources.
Si la relation de partage de ressources n'est pas une relation d'équivalence alors, comme la relation est symétrique, deux cas sont possibles :

1. la relation $\rightleftharpoons$ n'est pas réflexive, alors elle ne se prolonge pas en un préordre, puisque un préordre est reflexif, ce qui contredirait la compatibilité (6.1) ;
2. la relation est réflexive, mais pas transitive, dans ce cas le prolongement est nécessairement transitif, ce qui contredirait à nouveau la compatibilité (6.1).

□

Deux possibilités s'offrent :

1. la relation $\rightleftharpoons$ de partage sur les ressources se prolonge en une relation de préordre calculable localement, dans ce cas on peut prendre ce préordre pour définir $\lhd$;
2. dans le cas contraire, il suffit de disposer d'une identification chromatique sur chaque processus à valeur dans un ensemble totalement ordonné. On peut alors prendre pour $\lhd$ la relation :

$$(p.v.res \rightleftharpoons q.v.res) \vee (p.v.Id \leq q.v.Id)\,.$$

L'idée que nous mettons en œuvre est très simple : en cas de conflit, le processus qui a la plus petite identité sera prioritaire.

Dans les réseaux anonymes, on peut résoudre un problème d'ALR à la condition de pouvoir définir un préordre compatible sur les ressources ; ce qui n'est pas toujours possible. Par exemple, dans le cas de l'exclusion mutuelle locale, le graphe de conflit est le graphe G (à un renommage près). Il faut donc pouvoir distinguer deux processus voisins par la relation $\lhd$, cela veut dire que deux processus voisins doivent être dans deux classes d'équivalence distinctes pour la relation d'ordre définit sur le quotient de V par la relation d'équivalence $\equiv$ définie par :

$$p \equiv q \Leftrightarrow p \lhd q \wedge q \lhd p$$

c'est, par définition, l'ordre quotient associé au préordre $\lhd$. Il est donc nécessaire qu'il y ait implicitement une coloration chromatique sur les processus. La condition est clairement suffisante. Nous avons retrouvé le résultat de[KY02].

2.6 Exemples de problèmes

Dans cette partie nous présentons la définition du prédicat $Task_p()$ dans un certain nombre de cas, afin d'implémenter des problèmes classiques de synchronisation locale. On suppose que le réseau est muni d'une identification chromatique. Chaque problème d'ALR se définit par une spécification de sûreté et une spécification de vivacité. La spécification de vivacité est toujours la même :

Spécification 11 [Vivacité]

Quand un processus demande une ressource, il finit par l'obtenir.

2.6.1 Exclusion mutuelle locale.

La propriété de sûreté de l'exclusion mutuelle locale est définie par :

Spécification 12 [Sûreté de l'exclusion mutuelle locale]

Dans toute exécution, deux processus voisins n'exécutent jamais leur section critique simultanément.

La propriété de sûreté de l'exclusion mutuelle locale est assurée par la relation binaire $\lhd$ définie par :

$$v.Id_p \leq v.Id_q$$

2.6.2 Les lecteurs-écrivain locaux.

Ce problème est discuté dans [Can03]. C'est la version locale du problème bien connu des lecteurs-écrivain introduit par [CHP71]. On suppose que chaque processus peut exécuter soit une opération de lecture soit une opération d'écriture sur une ressource partagée avec les voisins (partage de mémoire). La condition de sûreté est définie par :

Spécification 13 [Sûreté des lecteurs -écrivain locaux]

Dans toute exécution, si deux voisins exécutent leur section critique simultanément alors ils exécutent chacun une opération de lecture.

Soit $p.v.res \in \{R, W\}$ où "R" signifie *lecture* et "W" signifie *écriture*.
la propriété de sûreté est implémentée avec la relation binaire $\lhd$ définie par :

$$p.v \lhd q.v \equiv (p.v.res = R \wedge q.v.res = R) \vee (p.v.Id \leq q.v.Id)$$

2.6.3 Exclusion mutuelle locale de groupe.

Comme le problème précédent, l'exclusion mutuelle de groupe est discutée dans [Can03]. C'est la version locale de l'exclusion mutuelle de groupe introduite par [Jou98]. Chaque

processus partage un nombre fini de ressources mutuellement exclusives avec ses voisins. A chaque instant, une seule ressource peut être utilisée par un processus et ses voisins. La propriété de sûreté se définit comme suit :

Spécification 14 [Sûreté de l'exclusion mutuelle locale de groupe]

Dans toute exécution, si deux processus voisins exécutent leur section critique simultanément alors ils utilisent tous les deux la même ressource.

Toute solution du problème de l'exclusion mutuelle de groupe doit maximiser la concurrence. On suppose que chaque processus demande continuellement à accéder à une ressource. La relation $\lhd$ est définie par :

$$p.v \lhd q.v \equiv (q.v.res = p.v.res) \vee (p.v.Id \leq q.v.Id)$$

3 Synchronisation locale silencieuse, ou à la demande

On peut reprocher à l'algorithme 14 de ne pas être silencieux. En permanence les processus demandent une ressource, quitte à introduire une ressource $\perp$ qui correspond à ne vouloir aucune ressource (la "non demande" de ressource devient une demande particulière de ressource). L'unisson ne s'arrête jamais. Est-il possible de donner une version de cet algorithme qui soit silencieuse ?

3.1 Un protocle d'ALR à la demande

La réponse est oui. L'idée est de stabiliser l'unisson en un unisson global quand il n'y a pas de demande d'entrée en section critique. Ce choix permet aussi de faire remonter les processus et d'assurer le maximum de concurrence au redémarrage des demandes.
Pour ce faire, nous utilisons la constante $\perp$ ajoutée à l'ensemble $\mathbb{V}$. La constante $\perp$ signifie que le processus ne demande pas à entrer en section critique. On étend le préordre total $\lhd$ sur $\mathbb{V} \cup \{\perp\}$ par : $\forall x \in \mathbb{V},\ \perp \lhd x$. L'idée est que les processus qui ne veulent pas entrer en section critique sont prioritaires. La priorité aux paresseux ! Maintenant, nous introduisons le nouveau booléen $DoNotStop$:

$$DoNotStop_p \equiv p.v \neq \perp \vee (\exists q \in \mathcal{N}_p : (q.r >_l p.r) \vee q.v \neq \perp)$$

Ce qui signifie que le processus p ne doit pas s'arrêter de progresser si :

1. p ou un de ses voisins est demandeur d'une ressource,
2. un voisin est en avance sur p au sens de l'unisson.

On donne une nouvelle définition de $NormalStep$:

$$NormalStep_p \equiv (\forall q \in \mathcal{N}_p : q.r \geq_l p.r) \wedge DoNotStop_p$$

Ce qui signifie que le processus p ne s'incrémente que s'il le peut au sens de l'unisson et si le prédicat $DoNotStop_p$ est satisfait.
Alors le prédicat $NormalStepTask_p$ s'écrit à nouveau :

$$NormalStep_p \wedge (\forall q \in \mathcal{N}_p : q.r = p.r \Rightarrow Prior_p(q))$$

L'action NA est transformée de la manière suivante :

$$\begin{array}{rl} NA : NormalStepTask_p \rightarrow & \text{if } p.v \neq \perp \text{ then } << \text{ Section critique } >>; \\ & p.v := \perp;\ p.r := \varphi(p.r) \end{array}$$

On appelle ce nouvel algorithme $SS\_SilentALR$. Voir algorithme 15.

Algorithme 15 Schema de programme $SS\_SilentALR$

Constante :
$\mathcal{N}_p$: Ensemble des voisins de p ;
Variables :
$p.r \in \mathcal{X}$; $p.v : \mathbb{V}$ (* $\mathbb{V}$ n'importe quel type de donnée contenant "$\perp$" *)
Prédicats :

$Prior_p(q)$	$\equiv$	(* Dépend de la condition de sûreté *)
$Task_p$	$\equiv$	$\forall q \in \mathcal{N}_p : p.r = q.r \Rightarrow Prior_p(q)$
$DoNotStop_p$	$\equiv$	$p.v \neq \perp \vee (\exists q \in \mathcal{N}_p : (q.r >_l p.r) \vee q.v \neq \perp)$
$NormalStep_p$	$\equiv$	$(\forall q \in \mathcal{N}_p : q.r \geq_l p.r) \wedge DoNotStop_p$
...		
$NormalStepTask_p$	$\equiv$	$NormalStep \wedge Task_p$

Actions :

NA : $NormalStepTask_p$	$\longrightarrow$	if $p.v \neq \perp$ then << code de l'application >> ;
		$p.v := \perp$; $p.r := \varphi(p.r)$;
...		

Comme pour l'algorithme SS_ALR, on a l'implication :

$$NormalStepTask_p \Rightarrow NormalStep_p$$

ce qui garantit que le protocole est convergent vers Γ_M. Mais, comme précédemment, rien ne garantit qu'une exécution n'entraîne pas d'interblocage. Remarquons que pour l'algorithme $SS\_SilentALR$, il peut y avoir des interblocages ; on l'a même défini pour cela.

3.2 Preuve de la vivacité de $SS\_SilentALR$.

Théorème 95
Dans Γ_M, si un processus n'exécute jamais l'action NA durant une exécution maximale $e = \gamma_0\gamma_1\ldots$, alors e est fini et le dernier état de e est un unisson statique (c'est-à-dire que tous les processus sont synchrones) et $\forall p \in V :\ v_p = \perp$.

Preuve : S'il existe un processus qui n'exécute jamais l'action NA, alors son horloge n'est jamais incrémentée. Or le réseau est fini, donc le retard est borné par le diamètre de G ; on en déduit que e est nécessairement fini.

Considérons l'état final γ_r de l'exécution $e = \gamma_0\gamma_1 \dots \gamma_r$. Soit Λ l'ensemble des processus minimaux pour la relation de précédence. Si cet ensemble n'est pas le réseau tout entier alors par la connexité de celui-ci, il existe un processus q parmi les processus minimaux, qui est voisin d'un processus qui n'est pas minimal ; q peut donc entrer en section critique. La configuration n'est pas un interblocage et e n'est pas maximal, ce qui est une contradiction. Donc tous les processus sont minimaux, cela signifie que le système est à l'unisson, et les les variales $p.v$ sont toutes à $\perp$.

□

On en déduit immédiatement :

Corollaire 1 [Vivavité] Si, dans l'état $\gamma \in \Gamma_M$, le processus p demande à entrer en section critique, alors il finit pas le faire. La spécification de vivacité est satisfaite.

Corollaire 2 [Silence] Si tous les processus, vérifient $v_p = \perp$ et ne feront plus aucune demande d'entrée en section critique, alors tout processus p maximal pour la relation de précédence n'effectuera plus jamais l'action NA, et le système converge vers un unisson statique.

4 Remarques finales

Nous avons rappelé la notion d'allocation locale de ressources introduite par [CDP03] et énoncé diverses instances classiques de ce problème. Nous avons montré comment utiliser un unisson auto-stabilisant pour résoudre ces divers problèmes de manière auto-stabilisante.
La méthode consiste à définir un préordre sur les ressources compatible avec la relation de partage sur les ressouces, où, si ce n'est pas possible, sur les couples $(ressource, identité)$ pourvu que chaque processus soit muni d'une identité chromatique.
On peut remarquer que l'algorithme 14 satisfait les spécifications de l'unisson ; il est donc un unisson particulier non uniforme (voir conclusion page 126). On peut voir cela géométriquement : considérons le graphe des transitions de l'unisson stabilisé défini uniquement par l'action gardée NA ; en définissant l'algorithme 14, nous n'avons fait que "couper" les transitions qui sont incompatibles avec la condition de sûreté, et quelques autres qui dépendent du choix de l'ordre.
Revenons maintenant sur l'algorithme silencieux 15. On peut voir dans cet algorithme une nouvelle sorte d'unisson, un unisson silencieux, dont les spécifications sont :

Spécification 15 [Unisson silencieux]
Le problème de l'unisson asynchrone silencieux est de définir un protocol qui satisfasse pour toute exécution aux propriétés suivantes :

1. **Sûreté** :
   (a) Γ_M est clos ;
   (b) synchronisation : un processus peut incrémenter son horloge et effectuer l'action de l'application seulement si la valeur de son horloge est localement

inférieure ou égale à la valeur de celle de chacun de ses voisins.

2. **Vivacité** :
   (a) convergence : $\Gamma \triangleright \Gamma_M$;
   (b) dans Γ_M, tout processus p qui demande à effectuer l'action de l'application, finit par effectuer l'action de l'application ;
   (c) silence : si aucun processus ne demande à effectuer l'action de l'application, alors le système aboutit en un temps fini à un interblocage.

Pour construire un algorithme qui vérifie les spécifications de l'unisson silencieux, il suffit de prendre un des unissons non silencieux définis dans ce mémoire et d'utiliser le schema de protocole décrit par l'algorithme 16.

Algorithme 16 ($SS\_UnissonSilencieux$) : unisson silencieux

Constante :

$\mathcal{N}_p$: Ensemble des voisins de p ;

Variables :

$p.r \in \mathcal{X}$; $p.v : \mathbb{V}$ (* $\mathbb{V}$ n'importe quel type de donnée contenant "$\perp$" *)

Prédicats :

$DoNotStop_p \equiv p.v \neq \perp \vee (\exists q \in \mathcal{N}_p : (q.r >_l p.r) \vee q.v \neq \perp)$

$NormalStep_p \equiv (\forall q \in \mathcal{N}_p : q.r \geq_l p.r) \wedge DoNotStop_p$

...

Actions :

NA : $NormalStep_p \longrightarrow$ if $p.v \neq \perp$ then << code de l'application >> ;
$p.v := \perp$; $p.r := \varphi(p.r)$;

...

Chapitre 7

Vagues et Vaguelettes

> La connaissance mathématique considère le général dans le particulier, même dans le singulier, mais *a priori* et au moyen de la raison.
>
> E. Kant *Critique de la raison pure (III, 469).* [Kan87]

> I see no meaningful difference between programming methodology and mathematical methodology in general .
>
> E.W. Dijkstra, EWD 1209

Sommaire

1 Introduction

La plupart des tâches dans les systèmes répartis se réduisent à un problème de transmission de messages. On peut citer la diffusion d'une information avec ou sans retour, la synchronisation globale, l'établissement d'une réinitialisation globale, la détection de terminaison ou encore le calcul d'une fonction définie globalement sur le réseau. Autant de tâches dont les valeurs d'entrées dépendent de plusieurs processus, voire de tous les processus [RH90, Tel94, Lyn96b]. Il est clair qu'un bon point de vue est celui de la structure des communications. Après, savoir ce que l'on met dans ces communications est une autre question, liée au problème particulier que l'on veut résoudre. Nous adoptons dans ce chapitre le point de vue de Tel dans [Tel00]. Nous généraliserons légèrement son travail en introduisant deux nouveaux types de vagues, les vaguelettes et les vagues fortes. Nous aborderons la résolution auto-stabilisante de ces questions dans le cadre des réseaux anonymes.

1.1 Quelques remarques sur l'état de l'art

Dans les systèmes répartis asynchrones, on peut juste assurer qu'aucun processus ne commence à effectuer la phase $i + 1$ avant que tous les processus n'aient terminé la phase i et que chaque processus n'effectue chacune de ces phases successivement et indéfiniment. Cette tâche de synchronisation est appelée *barrière de synchronisation*. Elle fut introduite par Misra dans [Mis91] dans un graphe complet. Les recherches sur la synchronisation ont commencé avec Awerbuch [Awe85]. Des vagues de communications sont en général utilisées à cette fin. [KA98a] propose une solution auto-stabilisante sur le graphe complet. [HL01] propose une solution sur un anneau uniforme de taille impaire.
Bien sûr, une forme relaxée de la barrière de synchronisation est la barrière de synchronisation faible : les horloges sont en phase si les valeurs d'horloges de deux processus voisins ne diffèrent pas de plus d'une unité (sûreté), et chaque horloge s'incrémente indéfiniment (vivacité).

De nombreux algorithmes de vagues utilisent un arbre couvrant avec un processus initiateur appelé racine [Kru79, ABDT98, BDPV99]. [KMM02] présente une version de vagues en pipeline. Dans ces cas, les protocoles ne sont pas uniformes, mais seulement au mieux semi-uniformes, puisqu'ils utilisent un leader : la racine, qui est distincte des autres processus. Rappelons qu'un protocole est uniforme si chaque processus avec le même degré exécute le même programme.
Ainsi, si l'on veut définir des vagues dans un réseau anonyme, chaque processus peut-être un initiateur de vague et plus globalement, l'initiateur d'une tâche globale. Il y a là une concurrence entre les processus qui est irréductible dans un réseau anonyme. Pour faire face à cette concurrence, une solution pourrait être que chaque processus maintienne une mémoire de l'initiateur de chacune des vagues successives, voir par exemple [CDPV02], mais alors il y aurait un moyen de distinguer les processus et le réseau ne serait pas anonyme. Nous ne pouvons donc utiliser une telle idée.
[KA98a] propose une barrière de synchronisation auto-stabilisante dans un réseau anonyme complet ; pour les autres topologies, les auteurs utilisent une racine, le réseau est donc semi-uniforme.
Il est intéressant de chercher une solution auto-stabilisante à ce problème dans un réseau anonyme quelconque. Pour autant que nous le sachions, l'algorithme de "phase" proposé par Tel [Tel91] est le seul algorithme de vague proposé dans un réseau anonyme général. Cet algorithme n'est pas auto-stabilisant. Il requiert que les processus connaissent le diamètre du réseau, ou plus simplement une majoration du diamètre du réseau, cette majoration devant être la même pour chacun des processus. Réciproquement, Tel montre dans [Tel91] que pour définir une vague dans un réseau anonyme il est nécessaire de connaître une majoration du diamètre du réseau. Cette contrainte s'impose donc.

1.2 Contributions

L'objet principal de ce chapitre est de montrer que l'unison peut être vu comme un courant de vagues auto-stabilisant sur un réseau qui peut être anonyme, cadencé par un démon inéquitable.
Notre contribution porte sur trois points.

1. Nous introduisons la notion de barrière de synchronisation à distance ρ. C'est une généralisation de la notion de barrière de synchronisation [Mis91] selon laquelle aucun processus ne commence à executer sa phase $i+1$ avant que tous les processus à distance inférieure ou égale à ρ n'aient terminé leur phase i. Nous montrons comment définir une barrière de synchronisation à distance ρ auto-stabilisante dans un réseau anonyme quelconque. Pour cela nous utilisons un unisson. Le temps de stabilisation de notre barrière de synchronisation à distance ρ est celui de l'unison sous-jacent. La complexité en espace est aussi celle de l'unisson sous-jacent.
2. Nous introduirons deux variantes de vagues : les vaguelettes et les vagues fortes. Nous montrons que les vaguelettes sont utiles pour des calculs à distance uniformément bornée, et les vagues fortes sont utiles pour effectuer des calculs qui nécessitent une

inondation du réseau comme les problèmes de routage pour différentes métriques. Ces derniers problèmes trouvent une élégante formulation grâce aux r-opérateurs introduits par [Duc98]. Nous introduirons et utiliserons ce formalisme.

3. Enfin, nous montrons que les structures de communication de notre barrière de synchronisation organisent un courant de vaguelettes, ou pipeline de vaguelettes, ou encore flux de vaguelettes dans tout réseau anonyme. Nous montrons que si $\rho \geq D$ les communications définissent un courant de vagues, et que si ρ est plus grand ou égal à la longueur du plus long chemin simple dans le réseau, alors le protocole définit un courant de vagues fortes sur le réseau.

Le reste de ce chapitre est organisé comme suit. Dans la section 2, nous introduisons la notion de Dag causal. Dans la section 3, nous définissons la notion de barrière de synchronisation à distance ρ et nous introduisons un protocole réparti qui implémente une barrière de synchronisation à distance ρ auto-stabilisante dans un réseau anonyme quelconque. Dans la section 5, nous introduisons les deux types de vagues évoquées plus haut : les *vaguelettes* et les *vagues fortes*, et nous montrons la relation qu'il y a entre une vague forte et les problèmes $r-$*paramétrés*, c'est-à-dire les problèmes de calculs globaux paramétrés par des $r-$opérateurs idempotents sur chaque arête orientée du réseau. De nombreux problèmes silencieux peuvent s'exprimer dans ce formalisme, par exemple les problèmes de calcul de tables de routage pour différentes métriques, calcul d'arbres couvrants en profondeur d'abord, distribution multissources adaptative etc...[Duc98, Tix02, Duc06]
Dans la section 6, nous montrons comment un unisson peut être vu comme un courant de vagues, et plus généralement comme un courant de vaguelettes ou de vagues fortes. Dans la section 7, nous donnons quelques remarques de conclusion.

2 DAG de causalité

Revenons sur notre modèle, afin de préciser la notion d'événement. L'observation de l'exécution d'un algorithme réparti symbolisé par la suite des états du système : $\gamma_0\gamma_1....$ permet de saisir les événements. A chaque transition, un ou plusieurs processus changent leur état interne. Dans notre modèle, à chaque changement interne d'un processus correspond une lecture de l'état de chacun des voisins de ce processus, ces opérations étant exécutées simultanément (une étape atomique). Chaque changement interne est nommé *événement*, il est modélisé par un couple (p, t) où p est le processus concerné et t la date du changement interne de p. A chaque événement d'un processus p correspond la lecture des états des voisins de ce processus ainsi que la lecture de l'état du processus lui-même. Chaque événement est donc causalement lié aux événements les plus proches dans le passé de chacun des voisins ainsi qu'à l'état de p lui-même. Bien sûr, afin que les premiers événements puissent être causalement définis, on définit comme événements les couples $(p, 0)$ correspondant à l'état global de départ γ_0.

Définition 43 Evénements et relation de précédence.
Soit $e = \gamma_0\gamma_1....$ une exécution finie ou infinie.

Pour tout $p \in V$, $(p, 0)$ est un événement.

Soit $\gamma_t \rightarrow \gamma_{t+1}$ une transition de e . Si le processus p exécute une action gardée durant cette transition, on dit que p exécute une action à la date $t+1$, et $(p, t+1)$ est un événement ou encore un p-événement.

Plus loin, certains événements seront appelés *événements décidants*, voir la définition formelle des vagues page 5.1.

La *relation de précédence* associée est la plus petite relation binaire $\rightsquigarrow$ sur l'ensemble des énénements qui satisfasse les deux conditions suivantes :

1. Soit (p, t) un événement satisfaisant $t > 0$. Si t' est le plus grand entier tel que $0 \leq t' < t$ et (p, t') est un événement, alors $(p, t') \rightsquigarrow (p, t)$.
2. Soit (p, t) un événement satisfaisant $t > 0$. Soit $q \in \mathcal{N}_p$ et t' le plus grand entier tels que $0 \leq t' < t$ et (q, t') est un événement, alors $(q, t') \rightsquigarrow (p, t)$.

Dans les deux cas, un tel entier t' existe puisque $(p, 0)$ ou $(q, 0)$ sont des événements.

La notion d'événement et la notion de précédence étant définies, il est maintenant facile de définir une notion de passé d'un événement en définissant la clôture transitive de la relation de précédence. On obtient ainsi un ordre partiel sur les événements appelé DAG de causalité. Les notions de cône du passé d'un événement et de cône du futur d'un événement auront des propriétés suffisantes pour la suite de cette étude.

Définition 44 Dag de causalité.

1. L' *ordre causal* $\preceq$ sur l'ensemble des événements est la clôture reflexive et transitive de la relation de précédence $\rightsquigarrow$.
2. Le *cône du passé* de l'événement (p, t) est le DAG causal induit par les événements (q, t') tels que $(q, t') \preceq (p, t)$. Un *cône du passé* comprend un processus q s'il y a un q-événement dans ce cône.
3. La *couverture* d'un événement (p, t) est l'ensemble des processus q compris dans le cône du passé de (p, t), cet ensemble est noté $Cover(p, t)$.

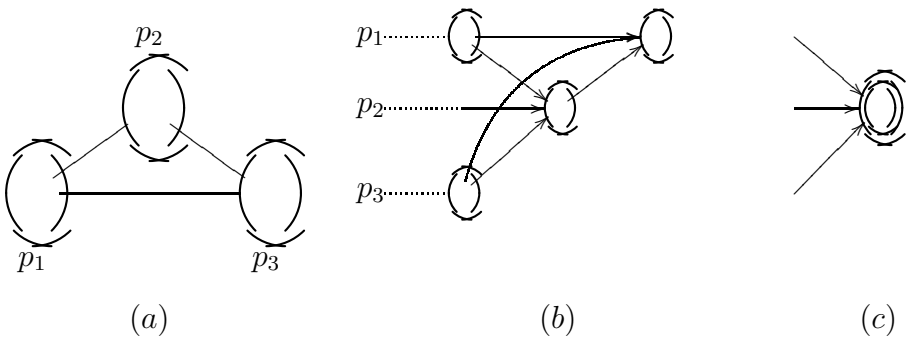


FIG. 7.1 – (a) Processus en triangle, (b) DAG de causalité (c) Evénement décidant

Sur la figure 7.1 nous représentons un réseau en triangle, puis une partie d'un DAG causal

associé à une exécution où les événements sont représentés par un disque vide entouré. Les événements décidants sont représentés par un disque vide doublement entouré.

Définition 45 Coupures.

1. Une *coupure* C sur un DAG causal est une application de V sur $\mathbb{N}$, qui associe à chaque processus p une date t_p^C telle que (p, t_p^C) est un événement ; on confondra cette application avec son graphe : $C = \left\{ \left(p, t_p^C\right), p \in V \right\}$.
2. Le *passé* de la coupure C est l'ensemble des événements (p, t) tels que $t \leq t_p^C$. Cet ensemble est noté $]\leftarrow, C]$.
3. Le *futur* de C est l'ensemble des événements (p, t) tels que $t_p^C \leq t$. Il est noté $[C, \rightarrow[$.
4. Une coupure est *cohérente* si $(q, t') \preceq (p, t)$ et $(p, t) \preceq \left(p, t_p^C\right)$ implique $(q, t') \preceq \left(q, t_q^C\right)$.
5. Une coupure C_1 est inférieure ou égale à la coupure C_2 si le passé de C_1 est inclus dans le passé de C_2. Cette relation d'ordre sur les coupures est notée par $\preceq$.
6. Si C_1 et C_2 sont cohérentes et que $C_1 \preceq C_2$ alors $[C_1, C_2]$ est le DAG causal *induit* par les événements (p, t) tels que $\left(p, t_p^{C_1}\right) \preceq (p, t) \preceq \left(p, t_p^{C_2}\right)$. Ce DAG induit est appelé le segment $[C_1, C_2]$, il est une *séquence* d'événements définie par deux coupures cohérentes , chaque événement de C_1 est applé un *événement initial* du segment.

3 Barrière de synchronisation

3.1 Barrière de synchronisation à distance ρ

La notion de *barrière de synchronisation* est introduite dans [Mis91], par Misra. Elle est reprise par [KA98a]. Elle correspond à ce que nous avons appelé dans notre travail, *barrière de synchronisation faible*. Dans les chapitres précédents nous avons distingué trois notions :

1. la notion de barrière de synchronisation faible (page 93) ;
2. la notion de barrière de synchronisation forte asynchrone (page 114) ;
3. la notion de barrière de synchronisation forte synchrone (page 114).

Les deux premières spécifications sont définies pour un démon quelconque, la dernière est spécifique au démon synchrone. On peut penser les deux premières notions comme des instances particulières d'une notion plus générale, la *barrière de synchroisation à distance* ϱ.
Soit ρ un entier strictement supérieur à 0, soit K un entier strictement plus grand que 1, on suppose que chaque processus p maintient une horloge d'ordre K, définie par le registre $p.r \in \{0, 1, ..., K-1\}$.
Ce protocole est une barrière de synchronisation à distance ρ si :

Spécification 16 [Barrière de synchronisation à distance ϱ] Pour tout $i \in \{0, 1, ..., K-1\}$

1. **sûreté** :
   (a) (intitialisation) initialement, tous les processus ont effectué correctement la phase 0 ;
   (b) (synchronisation) aucun processus ne peut incrémenter son horloge pour effectuer *l'action de l'application* de la phase $i+1$ *mod* K tant que tous les processus q, tels que $d(p,q) \leq \rho$ n'ont pas fini l'action d'application de la phase i ;
2. **vivacité** :
   tout processus p incrémente son horloge et exécute *l'action de l'application* une infinité de fois.

Pour $\rho = 1$, cette spécification est la spécification de la barrière de synchronisation faible (barrière locale). Pour $\rho \geq d$, cette spécification est la spécification de la barrière de synchronisation forte asynchrone (barrière globale).

3.2 Le schema général auto-stabilisant

L'idée est d'utiliser un unisson sous-jacent afin de synchroniser une horloge d'ordre δK, avec δ assez grand afin de garantir que la valeur absolue du délai entre deux processus éloignés d'une distance d'au plus ρ ne soit jamais plus grande que δ. Il est clairement suffisant de prendre $\delta \geq \rho$.
Nous utiliserons l'unisson défini page 96 et dans [BPV04]. Le temps de convergence vers Γ_1 est dans $O(n)$. On prend le système d'incrémentation défini sur $\chi = \{-\alpha, .., 0, .., \delta K - 1\}$ avec $\alpha \geq T_G - 2$. Le protocole est défini par l'algorithme $GAU(K, \alpha, M)$. On suppose $\delta K > C_G$ afin d'assurer la convergence vers $\Gamma_1^0 = \Gamma_1$.
Si on veut programmer une barrière de synchronisation, on doit prendre $\delta \geq d$. dans ce cas, puisque $C_G \leq 2d$, si $K \geq 3$ alors l'inégalité $K\delta > C_G$ est assurée. Dans le reste de ce chapitre nous supposerons que l'inégalité $K\delta > C_G$ est assurée.

Algorithme 17 $(SS - WS)$Barrière de synchronisation à distance ρ pour le processus p

Constante et variable :

$\mathcal{N}_p$: l'ensemble des voisins du processus p ; $p.r \in \chi$;

Fonctions booléennes :

$ConvergenceStep_p \equiv p.r \in tail_\varphi^* \wedge (\forall q \in \mathcal{N}_p : (q.r \in tail_\varphi) \wedge (p.r \leq_{tail_\varphi} q.r))$;
$LocallyCorrect_p \equiv p.r \in ring_\varphi \wedge (\forall q \in \mathcal{N}_p, q.r \in ring_\varphi \wedge ((p.r = q.r) \vee (p.r = \varphi(q.r)) \vee (\varphi(p.r) = q.r)))$;
$NormalStep_p \equiv p.r \in ring_\varphi \wedge (\forall q \in \mathcal{N}_p : (p.r = q.r) \vee (q.r = \varphi(p.r)))$;
$ResetInit_p \equiv \neg LocallyCorrect_p \wedge (p.r \notin tail_\varphi)$;

Actions :

NA : $NormalStep_p \longrightarrow$ if $p.r \equiv \rho - 1[\rho]$ then << CA 2 >> else << CA 1 >> ; $p.r := \varphi(p.r)$;
CA : $ConvergenceStep_p \longrightarrow p.r := \varphi(p.r)$;
RA : $ResetInit_p \longrightarrow p.r := -\alpha$ (reset) ;

3.3 Analyse de l'algorithme

3.3.1 Relèvement

Afin d'analyser le protocole 17 nous utiliserons la notion de *relèvement* défini page 72. Rappelons les objets de cette construction : on introduit pour chaque processus p le registre virtuel $\widetilde{p.r}$ à valeur dans $\mathbb{Z}$.
Soit $\gamma_0\gamma_1....$ une exécution débutant dans Γ_1. Soit p_0 le processus minimal, au sens de la relation de *précédence*, pour l'état γ_0. Soit $\perp_0 = p_0.r$ à la date 0. Pour chaque processus $p \in V$, nous relevons le registre $p.r$ de la manière suivante : pour l'état γ_0, on initialise le registre virtuel par l'instruction $\widetilde{p.r} := \perp_0 + \Delta_{(p_0,p)}(\gamma_0)$. Ensuite, durant l'exécution à partir de γ_0, pour chaque transition $\gamma_t \to \gamma_{t+1}$, l'instruction $\widetilde{p.r} := \widetilde{p.r} + 1$ est effectuée si et seulement si l'instruction $p.r := \overline{p.r + 1}$ est effectuée durant la transition.
Comme le retard est borné par le diamètre du réseau, alors si $k \geq \perp_0 + d$, la coupure $C_k = \left\{\left(p, t_p^k\right), p \in V\right\}$, où t_p^k est la plus petite date telle que $\widetilde{p.r} := k$ est bien définie. La première question est de savoir si ces coupures sont cohérentes. Nous commençons par donner le lemme facile :

Lemme 96 Si $(p,t) \rightsquigarrow (q,t')$ alors : $\widetilde{q^{t'}.r} \in \left\{\widetilde{p^t.r}, \widetilde{p^t.r} + 1\right\}$. Et par récurrence, si $(q_0,t_0) \rightsquigarrow (q_1,t_1) \rightsquigarrow (q_2,t_2) ... \rightsquigarrow (q_i,t_i)$ alors : $\widetilde{q_i^{t_i}.r} \in \left\{\widetilde{q_0^{t_0}.r}, ..., \widetilde{q_0^{t_0}.r} + i\right\}$ où $p^t.r$ est la valeur de $p.r$ à la date t.

D'après le lemme 96, si $(q,t) \preceq \left(p, t_p^k\right)$ alors $(q,t) \preceq \left(q, t_q^k\right)$. On en déduit la proposition :

Proposition 97 Pour tout $k \geq \perp_0 + D$ la coupure C_k est cohérente.

Prenons l'exemple d'une chaîne de 4 processus selon la figure 7.2. Sur la figure 7.3, nous avons représenté un segment $[C_1, C_2]$ d'un DAG causal associé à ces quatre processus exécutant un unisson déjà stabilisé.

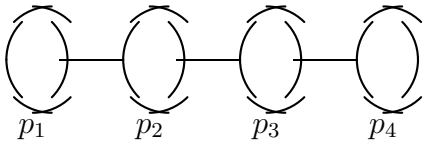


FIG. 7.2 – Une chaîne de 4 processus

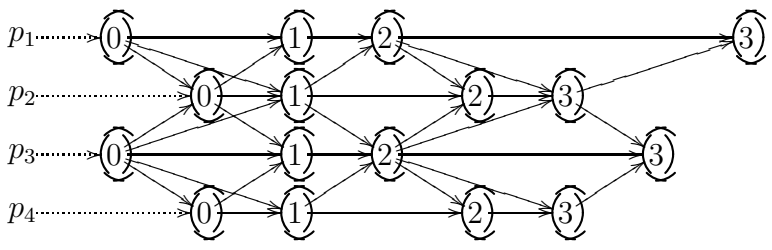


FIG. 7.3 – Unisson et structure du DAG causal sur le segment $[C_0, C_3]$

3.3.2 Registre virtuel $p.R$ et horloge virtuelle

A chaque processus p, nous associons maintenant un autre registre : le registre $p.R$, qui est virtuel. Sa valeur est donnée par la procédure :

$$\text{si } p.r \in stab_\varphi \text{ alors } p.R := p.r/\delta \text{ sinon } p.R := -1$$

où le symbole / est l'opérateur de *division entière*. La valeur de $p.R$, contrairement au relèvement, est accessible au processus p.
Le registre $p.R$ définit une horloge sur $\{-1, 0, ..., K-1\}$. L'algorithme 17 résout le problème de l'unisson asynchrone car chaque processus p incrémente son horloge $p.r$ infiniment souvent ; on en déduit que $p.R$ s'incrémente infiniment souvent et que le code de l'application $<<$ CA 2 $>>$ est exécuté infiniment souvent (**Vivacité**).

Théorème 98 Si $\delta \geq \rho$, et une fois que le protocole a convergé vers Γ_1 , alors celui-ci résout le problème de la barrière de synchronisation à distance ρ pour l'horloge virtuelle définie par le registre $p.R$.

Preuve : Considérons le segment $[C_{U\delta}, C_{U\delta+\delta-1}]$, avec U entier tel que $U\delta \geq \perp_0$. Pour tout événement (p,t) dans cette séquence, le registre $p.R$ est égal à $\overline{U}\,[K]$. Soit p et q deux processus tels que $d(p,q) \leq \rho$. Soit (p,t_p) et (q,t_q) deux événements dans $C_{U\delta+\delta}$.
Supposons que $t_p \leq t_q$; à la date t_p, le registre $\widetilde{q.r} \in \{U\delta+\delta-i, i \in \{0,...,\rho-1\}\}$, donc, puisque $\delta \geq \rho$, à la date t_p le code d'application $<<$ CA 2 $>>$ dans le segment $[C_{U\delta}, C_{U\delta+\delta-1}]$ est terminé pour le processus q.

□

Définition 46 En reprenant les notations de la preuve précédente, on appelle phase U le segment $[C_{U\delta}, C_{U\delta+\delta-1}]$. Ainsi la dernière phrase de la preuve précédente peut se dire : "à la date t_p le code d'application $<<$ CA 2 $>>$ de la phase U est terminé".

Remarques

1. Notre protocole synchronise les processus à distance ρ dans n'importe quel réseau anonyme (connexe). Sur le graphe général, ce synchroniser n'a besoin d'aucune identité, et ne construit, ou n'utilise aucun arbre couvrant.
2. En général, pour programmer une barrière de synchronisation, on utilise un arbre couvrant enraciné et on déclanche une vague le long de cet arbre avec un retour (feedback) de l'information. La phase démarre alors à partir de la racine. Ici, la phase démarre à partir de n'importe quel processus p, le démarrage de la nouvelle phase est décentralisé. Pour chaque noeud $q \in V(p,\rho)$, une fois que q a fini sa phase, ce noeud sait que l'information est parvenue à tous les autres nœuds de la boule $V(q,\rho)$, le feedback est implicite.
3. La complexité en temps de la phase $[C_{U\delta}, C_{U\delta+\delta-1}]$ est de δ rondes en pire cas. la complexité en messages est $2\delta\,|E|$. C'est le prix que nous payons pour l'uniformité du réseau.

Cette complexité en messages est-elle utilisable à d'autres fins ? Nous allons donner une réponse positive et spectaculaire à cette question dans la section 6, mais avant il faut définir avec précision ce que sont les $r-$opérateurs et les vagues. Ce sera l'objet des sections 4 et 5.

4 Infima et $r-$opérateurs

4.1 Les opérateurs d'infimum.

Tel, dans son travail sur les algorithmes de vagues [Tel94], introduit les opérateurs d'infima. Un infimum $\oplus$ sur un ensemble $\mathbb{S}$ est un opérateur binaire associatif, commutatif et idempotent (i.e. $x \oplus x = x$). Si $P = \{a_1, a_2, ..., a_r\}$ est une partie finie de $\mathbb{S}$, on convient, grâce à l'associativité, que $\oplus P$ est défini comme : $a_1 \oplus a_2 \oplus ... \oplus a_r$; et si $a \in \mathbb{S}$, alors $a \oplus P$ est défini comme : $a \oplus a_1 \oplus a_2 \oplus ... \oplus a_r$. L'opérateur $\oplus$ definit une relation d'ordre partiel $\leq_\oplus$ sur $\mathbb{S}$, par : $x \leq_\oplus y$ si et seulement si $x \oplus y = x$.
Nous supposons que $\mathbb{S}$ a un plus grand élément $e_\oplus$, tel que pour tout $x \in \mathbb{S}$, $x \leq_\oplus e_\oplus$. Ainsi $(\mathbb{S}, \oplus)$ est un semi-groupe idempotent avec $e_\oplus$ comme élément neutre pour l'opérateur $\oplus$.
Les opérateurs $\oplus$, ou $s-$opérateurs permettent de modéliser un calcul d'infimum global dans un réseau. l'algorithme local est défini de la manière suivante, chaque processeur possède une donnée privée, le calcul global consiste à calculer l'infimum des données privées de chacun des processus, et que cette valeur soit connue de tous les processus.
Tel montre que le calcul d'un infimum global revient à mettre en place une vague.
On peut proposer l'algorithme local suivant : à chaque pas de calcul d'un processus, celui-ci calcule la quantité *donnée_privée* $\oplus$ *entrée_*1 $\oplus$ .... $\oplus$ *dernière_entrée* et la propose en sortie. Ce calcul converge vers un point fixe, la valeur de l'infimum global sous un démon faiblement équitable. Le résultat est encore vrai sous un démon inéquitable, si l'on considère qu'un processus n'effectue pas de pas de calcul si le calcul d'infimum local ne change pas la valeur précédement calculée.
Malheureusement, cette méthode n'est pas auto-stabilisante : si la valeur de sortie est corrompue, elle peut entrainer la corruption du calcul global. Prenons par exemple comme opérateur le *min* sur les entiers, supposons que tous les processus aient pour valeur privée la valeur 1, et qu'une faute transitoire donne pour valeur de sortie d'un des processus la valeur 0, alors le système va converger vers 0.

4.2 Les $r-$opérateurs.

Ducourthial cherche des opérateurs intéressant du point de vue applicatif qui, comme les $s-$opérateurs, assurent la convergence vers un point fixe, et ne sont pas sensibles aux fautes transitoires. Pour cela Ducourthial a introduit dans [Duc98] la notion de r-opérateur qui généralise les opérateurs d'infimum.

> **Définition 47** L'opérateur binaire $\triangleleft$ sur $\mathbb{S}$ est un $r-$opérateur s'il existe un $(\mathbb{S}, \oplus)$-endomorphisme r, appelé r-fonction, tel que : $\forall x, y \in \mathbb{S}, x \triangleleft y = x \oplus r(y)$.

On dit que $\triangleleft$ est *idempotent* si et seulement si $\forall x \in \mathbb{S},\ x \leq_{\oplus} r(x)$.
$\triangleleft$ est strictement *idempotent* si et seulement si : $\forall x \in \mathbb{S} \backslash \{e_{\oplus}\},\ x <_{\oplus} r(x)$ et $e_{\oplus} = r(e_{\oplus})$.
On généralise de la manière suivante : Une application $\triangleleft$ de $(\mathbb{S})^n$ dans $\mathbb{S}$ est un $n-$aire $r-$opérateur s'il existe $n-1$ $(\mathbb{S}, \oplus)$-endomorphismes $r_1, r_2, ..., r_{n-1}$ tels que pour tout $(x_0, x_1, ..., x_{n-1}) \in (\mathbb{S})^n : \triangleleft(x_0, x_1, ..., x_{n-1}) = x_0 \oplus r_1(x_1) \oplus ... \oplus r_{n-1}(x_{n-1})$

Remarque

"r est un endomorphisme" signifie que pour tous x et y dans $\mathbb{S}$, $r(x \oplus y) = r(x) \oplus r(y)$. De la définition de $\leq_{\oplus}$, on déduit que r est compatible avec la relation $\leq_{\oplus}$, c'est à dire que : $\forall x, y \in \mathbb{S}, x \leq_{\oplus} y \Rightarrow r(x) \leq_{\oplus} r(y)$

Remarque

Pour tout $x \in \mathbb{S}$, on a : $r(x) = e_{\oplus} \oplus r(x) = e_{\oplus} \triangleleft x$, cela signifie que r est défini de manière unique par $\triangleleft$.

La proposition 99 est un exercice facile (voir[Tix02]).

Proposition 99
Si $\triangleleft$ un $r-$opérateur binaire $\mathbb{S}$, et si r est le $r-$opérateur associé, alors :
r est $r-$associatif : $\forall x, y, z \in \mathbb{S},\ (x \triangleleft y) \triangleleft r(z) = x \triangleleft (y \triangleleft z)$
r est $r-$commutatif : $\forall x, y \in \mathbb{S},\ r(x) \triangleleft y = r(y) \triangleleft x$
r est $r-$idempotent : $\forall x \in \mathbb{S},\ r(x) \triangleleft x = r(x)$

Dans la suite, nous supposerons toujours que l'opérateur $\triangleleft$ est idempotent.

4.3 Algorithme global paramétré par des $r-$opérateurs.

L'objet d'une tâche statique est de calculer un résultat global, où à chaque exécution, chaque processus calcul la même valeur de sortie. L'état global correspondant à ces valeurs de sortie est le résultat du calcul. C'est la spécification de la tâche statique.
Pour résoudre cette tâche, chaque processus p exécute un algorithme local à partir de valeurs d'entreée, qui sont les valeurs de sortie $q.res$ des processus voisins q, et d'une valeur privée $p.v$ du processus p. Ces valeurs appartiennent à un ensemble $\mathbb{S}$ fini. Le calcul local est une fonction calculable de ces valeurs, la valeur calculée est la nouvelle valeur de sortie du processus p. Les tâches statiques paramétrées par des $r-$opérateurs idempotents se définissent ainsi :
On associe à chaque arête orientée (p_i, p_j) du graphe $G = (V, E)$ une $r-$*fonction* idempotente : r_{p_i,p_j}. Par extension, pour la séquence (p_i, p_i) on associe l'identité : $r_{ii} = id$. Chaque registre $p.res$ est initialisé par la valeur $p.v$.
Lors d'un calcul, $p.res$ est affecté de la valeur $p.v \oplus \left\{r_{q,p}(q.res), q \in \mathcal{N}_p^t\right\}$. Chaque p peut être vu comme un $(d+1)$-aire r-opérateur si d est le degré du noeud.
[Duc98] montre que si les $r-$opérateurs sont idempotents, alors ce calcul distribué converge vers un point fixe sous un démon faiblement équitable. Le résultat est encore vrai sous un démon inéquitable, si l'on considère qu'un processus p n'est pas activable si le calcul d'infimum local ne change pas la valeur de $p.res$ précédement calculée.

De plus, si les $r-$opérateurs sont strictement idempotents, alors cet algorithme est auto-stabilisant : si la valeur de sortie est corrompue, alors cela n'affectera ni la convergence, ni le résultat final [DT01]. Dans ce cas, la convergence vers le point fixe se fait en $O(|\mathbb{S}| + d)$ en pire cas sous un démon synchrone [DDT03].

Définition 48 On définit un *problème r−paramétré* comme la donnée D pour chaque processus p de la valeur privée $p.v$ et des $r-$opérateurs idempotents associés à chaque arête orientée entrant vers le noeud p. On appelle *résultat du problème donné* D, le point fixe associé au problème dans un envirronement sans panne. Rappelons que si le problème est un calcul d'infimum, alors r est l'identité.

4.4 Etude d'un exemple

Dans un réseau dynamique, si l'on veut maintenir un arbre couvrant, il faut commencer par maintenir une racine de cet arbre. Un moyen trés simple est de désigner la racine comme le processus avec l'identité la plus petite (on suppose pour simplifier, que les identités sont prises sur l'ensemble des entiers naturels). Un bon moyen est de programmer l'opérateur d'infimum du min sur les entiers. On obtient un algorithme qui converge vers un "consensus" : la racine est le noeud de plus petite identité.
Comme nous l'avons vu, l'algorithme n'est pas auto-stabilisant. Une faute transitoire peut introduire une fausse racine (par exemple, à cause du crash de la racine effective), cette racine devient une fausse racine et l'algorithme ne la repère pas. Le "consensus" porte alors sur une valeur qui n'est plus une valeur proposée par les processus du réseau.
On peut corriger cette question en utilisant un $r-$opérateur et utiliser les résultats précédents. Pour cela nous allons travailler sur des couples d'entiers (id, k) où i représente une identité et k représente une distance, en gros la distance d'un processus au processus qui a l'identité id (si il existe). L'idée est de borner k par D, un majorant du diamètre du réseau.
$\mathbb{S}$ est l'ensemble $\mathbb{N}\times\{0, 1, ..., D\} \cup \{+\infty\}$
L'infimum $\oplus$ est défini par (définition par cas, voir page 22) :

$$(id_1, k_1) \oplus +\infty = +\infty \oplus (id_1, k_1) = (id_1, k_1)$$

$$(id_1, k_1) \oplus (id_2, k_2) = \begin{array}{|ll} id_1 < id_2 \rightarrow & (id_1, k_1) \\ id_2 < id_1 \rightarrow & (id_2, k_2) \\ id_2 = id_1 \rightarrow & (id_1, \inf(k_1, k_2)) \end{array}$$

Le $r-$opérateur r est défini par :

$$\begin{aligned} r(+\infty) &= +\infty \\ r(id, D) &= +\infty \\ r(id, k) &= (id, k+1) \text{ avec } k \in \{0, 1, ..., D-1\} \end{aligned}$$

Ce $r-$opérateur est clairement strictement idempotent.

On paramètre maintenant chaque arête orientée par l'opérateur r, la valeur privée de chaque processu p est $p.v = (p.id, 0)$ où $p.id$ est l'identité du processuer p.
l'algorithme paramétré ainsi défini est donc silencieux, auto-stabilisant. Il calcule l'identité de la racine *root*, et donne aussi la distance de chaque processus à la racine. Il définit donc le DAG couvrant en largeur d'abord enraciné en *root*.

5 Vaguelettes, vagues et vagues fortes

Comme nous allons le voir dans la section 6, durant chaque phase de l'algorithme 17, la structure des communications est une sorte de vague qui dépend de la valeur de δ. La structure de communication qui apparait est formellement définie de manière générale dans cette section, cette partie est indépendante de l'unisson. Dans la section 6, suivant le théorème 107, nous serons capables d'utiliser ces communications pour calculer certaines fonctions sur le réseau, par exemple un infimum si $\delta \geq d$, ou plus généralement pour calculer le point fixe d'un problème paramétré par des r-opérateurs idempotents quand $\delta \geq n$, et ainsi de résoudre de nombreuses tâches silencieuses [Duc98].

5.1 Marches et Vagues

Définition 49 [**Marche**]
Une *marche* est un mot fini non vide $m = q_0 q_1 .... q_r$ sur l'alphabet V, tel que pour tout $i \in \{0, r-1\}$, $q_i = q_{i+1}$ ou $q_{i+1} \in \mathcal{N}_{q_i}$.
Une marche est circulaire si $r > 1$ et $q_0 = q_r$. Une marche est stationnaire si elle n'est composée par une seule lettre.
La marche m commence en q_0 noté $head(m)$, et finit en q_r. Sa longueur est r.

Définition 50 [**Réduction**]

1. Soit m une *marche*. S'il existe deux mots m_1 et m_2 et un chemin circulaire u tel que $m = m_1 u m_2$, (u est un facteur de m), alors $m_1 head(u) m_2$ est une marche et nous écrivons : $m \rightarrow m_1 head(u) m_2$
2. la clôture transitive de la relation binaire $\rightarrow$ définit un ordre partiel strict $\stackrel{*}{\rightarrow}$ sur l'ensemble des *marches*.
3. Une *marche simple* est une *marche minimale* au sens de la relation d'ordre partiel $\stackrel{*}{\rightarrow}$. Plus simplement, une *marche simple* est une *marche* sans *répétition*.
4. Une *marche élémentaire* est une *marche* telle que si pour $i < j$, $q_i = q_j$; alors pour tout $k \in \{i, ..., j\}$, $q_k = q_i$.
5. Une *réduction* d'une *marche* m est une *marche simple* m' telle que $m \stackrel{*}{\rightarrow} m'$.

Généralement, la réduction élémentaire d'une marche réduite n'est pas unique, comme le montrent les figures 7.4 et 7.5 ; en revanche si la marche m est *élémentaire*, alors sa réduction est unique.

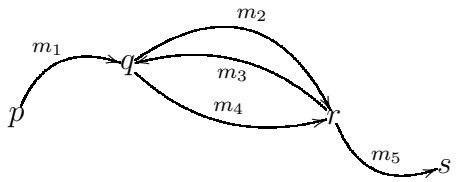


FIG. 7.4 – La marche $pm_1qm_2rm_3qm_4rm_5s$

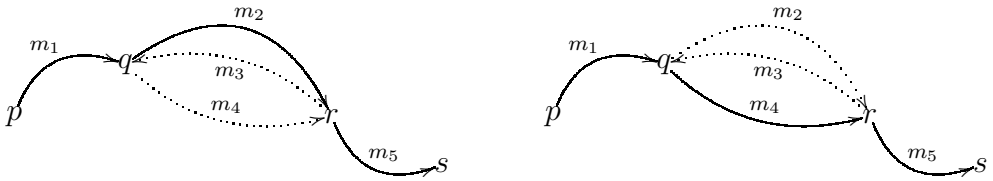


FIG. 7.5 – Deux réductions possibles de la marche $pm_1qm_2rm_3qm_4rm_5s$

Définition 51 [**Couverture des marches d'un événement dans une séquence**]
Soit $S = [C_1, C_2]$ une *séquence d'événements*.
Si dans S, $(q, t') \preceq (p, t)$ alors il existe une *chaîne de causalité* de (q, t') vers (p, t) : $(q, t') = (q_0, t_0) \rightsquigarrow (q_1, t_1) \rightsquigarrow (q_2, t_2) ... \rightsquigarrow (q_r, t_r) = (p, t)$; sa *marche* associée est la *marche* $q_0q_1...q_r$.
La *couverture des marches* d'un événement $(p, t) \in S$ est l'ensemble des *marches* associées aux chaînes de causalité de S finissant en (p, t). Cet ensemble est noté $WalkCover_S(p, t)$. Bien sûr, cet ensemble contient la *marche* de longueur 0 notée p.

Lemme 100 Si $m \in WalkCover_S(p, t)$ alors il existe un chemin élémentaire m' dans $WalkCover_S(p, t)$ tel que $m \xrightarrow{*} m'$

Preuve :
Soit $m = q_0q_1.....q_r$ la marche associée à la chaîne de causalité $(q_0, t_0) \rightsquigarrow (q_1, t_1) \rightsquigarrow (q_2, t_2) ... \rightsquigarrow (q_r, t_r)$. Supposons que $m = m_1um_2$ où u est une marche circulaire non stationnaire $q_iq_{i+1}.....q_j$. D'après la définition de la relation $\rightsquigarrow$, il existe une chaîne : $(q_i, t_i) \rightsquigarrow (q_i, t_{i_1}) \rightsquigarrow (q_i, t_{i_2}) ... \rightsquigarrow (q_i, t_j)$. Soit l la longueur de cette chaîne. Si $v = \prod_{k=1}^{l} q_i = q_i^l$ alors $\bar{m} = m_1vm_2$ est un élément de $WalkCover_S(p, t)$. Comme le mot de départ est fini et que u est de longueur au moins trois, la réécriture fait décroître strictement la longueur du mot. Notre opération de *réécriture* n'est donc possible qu'un nombre fini de fois. A la fin du processus de réduction le mot obtenu est élémentaire.

□

Définition 52 [**Vaguelettes, vagues et vagues fortes**]
En suivant [Tel94], nous supposons qu'il existe un type spécial d'événements appelés *événements décidants*, la nature de ces événements dépend de l'algorithme et du problème traité. Soit k un entier. Une *k-vaguelette* est une *séquence d'événements*

$[C_1, C_2]$ qui satisfait aux deux propositions suivantes :

1. Le *DAG causal* induit par $[C_1, C_2]$ contient au moins un événement décidant.
2. Pour chaque *événement décidant* (p, t) , l'ensemble des processus du *passé* de (p, t) dans $[C_1, C_2]$ contient $V(p, k)$.

Si $k \geq d$, où d est le diamètre du réseau, une telle séquence est appelée une *vague*.
Une *vague forte* est une *vague* $[C_1, C_2]$ qui satisfait à la proposition supplémentaire :
Pour chaque *événement décidant* (p, t) dans $[C_1, C_2]$, et pour chaque marche simple $m_0 = q_0 q_1 ... q_{n-1} p$ (finissant en p), il existe une *chaîne de causalité* $(q_0, t_0) \rightsquigarrow (q'_1, t_1) ... \rightsquigarrow \left(q'_{r-1}, t_{r-1}\right) \rightsquigarrow (p, t)$ dans $[C_1, C_2]$, telle que sa marche associée m soit élémentaire et $m \stackrel{*}{\rightarrow} m_0$.

Reprenons l'exemple d'une chaîne de 4 processus selon la figure 7.2, et oublions pour l'instant qu'il s'agit de l'exécution d'un unisson. Sur la figure 7.6, nous avons représenté un segment $[C_1, C_2]$ d'un DAG causal associé à ces quatre processus. Remarquons que sur cet exemple la couverture des événements décidants est exactement l'ensemble des processus.

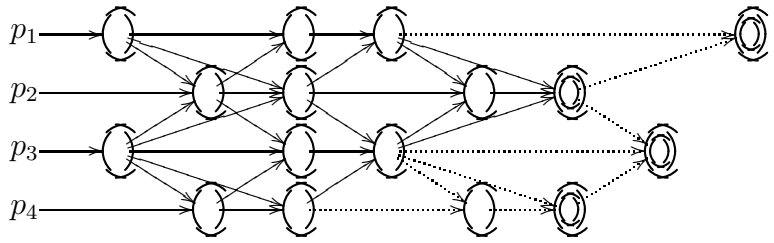


FIG. 7.6 – Structure des communications et événements décidants

5.2 Les problèmes $r-$paramétrés

Dans cette partie nous donnons une extension simple de la théorie de Tel sur les vagues. Après avoir rappelé le résultat de Tel relatif aux vagues pour calculer un infimum global, nous donnons deux résultats analogues à propos des vaguelettes pour le calcul d'un infimum à distance ρ, et des vagues fortes pour le calcul des problèmes $r-$paramétrés.

Définition 53 $[\mathcal{N}_p^t]$
Soit $U = [C_1, C_2]$ une séquence, on note $\mathcal{N}_p^t$ l'ensemble des processus tels qu'il existe une date t_q telle que $(q, t_q) \rightsquigarrow (p, t)$ dans U. Remarquons que p peut être dans $\mathcal{N}_p^t$.

5.2.1 Le cas particulier du calcul d'un infimum

On associe à chaque processus p un registre supplémentaire $p.res$: $\mathbb{S}$. Chaque registre $p.res$ est initialisé par la valeur $p.v_0$ appartenant à $\mathbb{S}$ durant l'événement initial de p. Soit (p, t) n'importe quel événement de $[C_1, C_2]$. Durant l'événement (p, t), $p.res$ reçoit la valeur $p.v_0 \bigoplus \left\{q^{t_q}.res, q \in \mathcal{N}_p^t\right\}$. Tel prouve le théorème suivant :

Théorème 101 [Tel94] Une vague peut être utilisée pour le calcul d'un infimum.

Preuve : On suppose que $[C_1, C_2]$ est une vague (voir 174). Avec le contexte et les notations ci-dessus, nous démontrons qu'à la fin de l'événement $(p,t) \in [C_1, C_2]$, le registre $p.res$ est égal à :

$$\bigoplus \{q.v_0, q \in Cover((p,t))\}.$$

Soit $\mathcal{A}$ l'ensemble des événements (p,t) tels que $p.res \neq \bigoplus \{q.v_0, q \in Cover((p,t))\}$.
Si $\mathcal{A}$ est vide, la preuve est finie. Remarquons que les événements minimaux de $[C_1, C_2]$ ne sont pas dans $\mathcal{A}$.
Supposons que $\mathcal{A}$ n'est pas vide. Soit (p,t) un événement minimal de $\mathcal{A}$ pour la relation $\preceq$. A la date t, $p.res := p.v_0 \bigoplus \left\{q.res, q \in \mathcal{N}_p^t\right\}$. Si $q \in \mathcal{N}_p^t$, on note t_q la date à laquelle $(q,t_q) \rightsquigarrow (p,t)$, alors $Cover(p,t) = \{p\} \bigcup_{q \in \mathcal{N}_p^t,} Cover\,(q,t_q)$. A cause de la minimalité de l'événement (p,t), on déduit que pour chaque $q \in \mathcal{N}_p^t$, l'égalité $q.res = \bigoplus \{s.v_0., s \in Cover((q,t))\}$ a lieu à la date t_q.
Il résulte que $p.res = \bigoplus \{q.v_0., q \in Cover((p,t))\}$ et que $\mathcal{A}$ est vide.
Nous savons, par la définition d'une vague, que pour tout événement décidant (p_0,t_0) de la séquence $[C_1, C_2]$, $Cover(p_0,t_0) = V$. Donc, suivant le résultat précédent, $p_0.v = \bigoplus \{q.v_0, q \in V\}$ à la date t_0.

□

5.2.2 Problèmes $r-$paramétrés généraux

Définissons précisément le contexte. soit $S = [C_1, C_2]$ une vague forte. On associe à chaque arête orientée (p_i, p_j) du graphe $G = (V, E)$ une $r-$*fonction* idempotente : r_{p_i,p_j}. Par extension, pour la séquence (p_i, p_i) on associe l'identité : $r_{ii} = id$. Comme précédemment, on associe à chaque processus p une nouvelle constante $p.v_0 : \mathbb{S}$ et un registre $p.res$. Chaque registre $p.res$ est initialisé durant l'événement initial de p par la valeur $p.v_0$. Soit (p,t) n'importe quel événement du segment $[C_1, C_2]$; durant l'événement (p,t) , $p.res$ est affecté de la valeur $p.v_0 \oplus \left\{r_{q,p}\,(q.res), q \in \mathcal{N}_p^t\right\}$.
Chaque p peut être vu comme un $(d+1)$-aire r-opérateur si d est le degré du nœud.

Définition 54 Pour toute marche $m = p_0 p_1 .... p_n$, on définit $eval\,(m) = r_m\,(p_0.v_0)$, avec
$r_m = r_{p_{n-1},p_n} \circ r_{p_{n-2},p_{n-1}} \circ ... \circ r_{p_0,p_1}$, où $\circ$ est l'opérateur de composition des fonctions.
Pour tout $p \in V$, les ensembles Λ'_p et Λ_p sont définis par : $\Lambda'_p = \left\{eval\,(m), m \in \Sigma'_p\right\}$ and $\Lambda_p = \{eval\,(m), m \in \Sigma_p\}$, où Σ'_p est l'ensemble des *marches* finissant en p, et Σ_p est l'ensemble des *marches simples* finissant en p.

Des définitions précédentes et de l'idempotence des r-opérateurs, on obtient le lemme suivant :

Lemme 102 Supposons que m et m' sont deux marches avec $p = head(m)$. On suppose que $m \xrightarrow{*} m'$. Alors $r_m(p.v_0) \geq_{\oplus} r_{m'}(p.v_0)$; et si m est élémentaire, alors $r_m(p.v_0) =$

$r_{m\prime}(p.v_0)$

Définition 55 [Valeur de sortie] Pour chaque processus p, nous définissons la quantité : $\oplus\Lambda_p$ comme la valeur de sortie légitime du processus p.
Ces valeurs légitimes pour chacun des processus définissent le point fixe à calculer pour le problème *r-paramétré* donné.

Nous montrons dans la sous-section suivante que l'utilisation d'une vague forte permet de résoudre le problème *r-paramétré*. C'est ce qu'exprime le théorème suivant :

Théorème 103 Une vague forte peut être utilisée pour résoudre les problèmes *r-paramétrés*.

5.3 Preuve du théorème 103

Proposition 104 Pour tout $p \in V$, la quantité $\oplus\Lambda'_p$ existe, et l'égalité $\oplus\Lambda'_p = \oplus\Lambda_p$ est vraie.

Preuve :
Λ_p est un ensemble fini, donc $\oplus\Lambda_p$ existe, de plus $\Lambda_p \subset \Lambda'_p$. Si $m = p_0p_1....p_n$ n'est pas une marche élémentaire , alors il existe une marche simple m' telle que $m \xrightarrow{*} m'$. Suivant le lemme 102, les inégalités $r_m(p.v_0) \geq_\oplus r_{m\prime}(p.v_0)$ et $r_{m\prime}(p.v_0) \geq_\oplus \oplus\Lambda_p$ sont vérifiées. La proposition en découle.

□

Lemme 105 Soit (p,t) un événement de $[C_1, C_2]$, alors à la date t :

$$p^t.res = \bigoplus \{eval(\mu), \mu \in WalkCover_S(p,t)\}$$

Preuve :
Soit $\mathcal{A}$ l'ensemble des événements (p,t) dans $[C_1, C_2]$ tels que l'égalité n'est pas vraie. Remarquons que les événements minimaux de $[C_1, C_2]$ ne sont pas dans $\mathcal{A}$.
Si $\mathcal{A}$ est vide, la preuve est finie.
Supposons que $\mathcal{A}$ n'est pas vide. Soit (p,t) un événement minimal de $\mathcal{A}$ au sens de la relation $\preceq$:

$$p^t.res := p.v_0 \bigoplus \left\{r_{q,p}(q^{t_q}.res), q \in \mathcal{N}_p^t\right\}$$

Mais alors, par définition :

$$WalkCover_S(p,t) = \bigcup_{q \in \mathcal{N}_p^t} \{\mu p, \mu \in WalkCover_S(q,t_q)\} \cup \{p\}$$

.
De la minimalité de (p,t) dans $\mathcal{A}$, les événements (q,t_q) ne sont pas dans $\mathcal{A}$, donc :

$$p^t.res = p.v_0 \bigoplus_{q \in \mathcal{N}_p^t} r_{qp}\left(\bigoplus \{eval(\mu), \mu \in WalkCover_S(q,t_q)\}\right)$$

Mais r_{pq} est compatible avec $\leq_{\oplus}$ (remarque 4.2), donc :

$$p^t.res = p.v_0 \bigoplus_{q \in \mathcal{N}_p^t} \{r_{qp}(eval(\mu)), \mu \in WalkCover_S(q, t_q)\}$$

(p,t) n'est pas un événement initial, donc $p \in \mathcal{N}_p^t$ et :

$$p.v_0 \geq_{\oplus} \bigoplus \{r_{pp} or_{\mu}(head(\mu).v_0), \mu \in WalkCover_S(p, t_p)\}$$

On en déduit, grâce à l'associativité de $\oplus$ et l'égalité r_{qp}
$circr_{\mu} = r_{\mu p}$ que :

$$p^t.res = \bigoplus_{q \in \mathcal{N}_p^t} \{eval(\mu p), \mu \in WalkCover_S(q, t_q)\}$$

or $WalkCover_S(p,t) = \bigcup_{q \in \mathcal{N}_p^t} \{\mu p, \mu \in WalkCover_S(q, t_q)\}$, donc :

$$p^t.res = \{eval(\mu), \mu \in WalkCover_S(p, t)\}$$

On déduit que (p,t) n'est pas dans $\mathcal{A}$, ce qui est une contradiction, donc $\mathcal{A} = \emptyset$ et le lemme est prouvé.

□

Corollaire 1 [**Théorème**103]
Une vague forte peut être utilisée pour résoudre le "problème r-paramétré".

Preuve :
Si (p,t) est un événement décidant alors, à partir du lemme 105, on établit l'égalité :

$$p^t.res = \{eval(\mu), \mu \in WalkCover_S(p, t)\}$$

alors $WalkCover_S(p,t)$ satisfait à la définition 5.1.
Rappelons que $\Lambda_p = \{eval(\mu), \mu \in \Sigma_p\}$. Pour tout $m \in WalkCover_S(p,t)$, il existe $m_0 \in \Sigma_p$ tel que $m \xrightarrow{*} m_0$ et, grâce au lemme 102, on obtient l'inégalité $eval(m) \geq eval(m_0)$. On en déduit que $p^t.res \geq \oplus\Lambda_p$. Réciproquement, si $m_0 \in \Lambda_p$, il existe $m \in WalkCover_S(p,t)$ tel que $m \xrightarrow{*} m_0$, or grâce au lemme 100, il existe aussi une marche $m_1 \in WalkCover_S(p,t)$ telle que $m \xrightarrow{*} m_1$ et $m_1 \xrightarrow{*} m_0$; alors du Lemme 100 on déduit que $eval(m_1) = eval(m_0)$. Nous déduisons que $p^t.res \leq \oplus\Lambda_p$.
De ces deux inégalités on déduit l'égalité $p^t.res = \oplus\Lambda_p$ et le théorème 103 est prouvé.

□

6 L'unisson : un courant de vagues auto-stabilisant

Dans la section précédente, nous avons montré comment certaines structures de communication peuvent être utilisées pour résoudre des problèmes classiques de calcul global. L'objet de cette partie est de montrer en quel sens, précisément, la structure des communications de l'unisson possède ces propriétés, et permet donc de résoudre ces problèmes.

6.1 Analyse du comportement d'un unisson commençant dans Γ_1

Définition 56 Pour tout processus p et pour tout $\varrho \in \mathbb{N}$, on note Σ_p^ϱ l'ensemble des marches simples de longueur inférieure ou égale à ρ et finissant en p.

Lemme 106 Soit $k \geq \perp_0 + d$.
Si (p,t) est un événement dans l'intervalle $S = [C_k, \rightarrow[$, alors : $V(p, \widetilde{p^t.r} - k) \subset Cover_S(p,t)$ et $\Sigma_p^{\widetilde{p^t.r}-k} \subset WalkCover_S(p,t)$.

Preuve :
Le lemme est vrai pour les événements initiaux de $[C_k, \rightarrow[$. Soit $\mathcal{A}$ l'ensemble des événements (p,t) dans $[C_k, \rightarrow[$ pour lesquels la proposition suivante est fausse :

$$V(p, \widetilde{p^t.r} - k) \subset Cover_S(p,t) \ \wedge \ \Sigma_p^{\widetilde{p^t.r}-k} \subset WalkCover_S(p,t).$$

On suppose que $\mathcal{A}$ n'est pas vide. Soit (q,τ) un événement minimal dans $\mathcal{A}$ au sens de la relation $\preceq$. Soit $\delta = \widetilde{q^\tau.r} - k$, et soit $p_1 \in V(q,\delta)$. Si $p_1 = q$ alors $p_1 \in Cover_S(q,\tau)$, sinon il existe $q_1 \in \mathcal{N}_q$ tel que $p_1 \in V(q_1, \delta - 1)$.
(q,τ) n'est pas un événement initial, donc $q_1 \in \mathcal{N}_q^\tau$ et il existe τ_{q_1} tel que $(q_1, \tau_{q_1}) \rightsquigarrow (q,\tau)$, et par la minimalité de (q,τ), on obtient l'inclusion $V(q_1, \delta - 1) \subset Cover_S(q_1, \tau_{q_1})$; on en déduit $V(q_1, \delta - 1) \subset Cover_S(q,\tau)$ et $p_1 \in Cover_S(q,\tau)$.
En suivant le même raisonnement, soit m une marche dans Σ_p^δ , si $m = q$ alors $m \in WalkCover_S(q,\tau)$, sinon si $m = p_1 p_2 .... p_r q$ alors $p_r \in \mathcal{N}_q^\tau$, car (q,τ) n'est pas un événement initial. On en déduit qu'il existe τ_{p_r} tel que $(p_r, \tau_{p_r}) \rightsquigarrow (q,\tau)$, et par la minimalité de (q,τ) on obtient l'inclusion $\Sigma_{p_r}^{\delta-1} \subset WalkCover_S(p_r, \tau_{p_r})$, et donc $p_1 p_2 .... p_r q \in WalkCover_S(q,\tau)$. Ainsi (q,τ) n'est pas dans $\mathcal{A}$. Nous avons prouvé que $\mathcal{A} = \emptyset$, le lemme en découle.

□

Comme corollaire de ce lemme, nous déduisons un théorème important, qui exprime le fait que l'unisson peut être vu comme un courant de vagues, de vagues fortes ou de vaguelettes auto-stabilisant :

Théorème 107 Soit $k \geq \perp_0 + d$ et δ deux entiers strictement positifs ; le segment $[C_k, C_{k+\delta}]$, avec $C_{k+\delta}$ comme ensemble des événements décidants, est :

1. une *δ-vaguelette* si $\delta < d$
2. une *vague* si $\delta \geq d$

3. une vague forte si δ est plus grand ou égal à la longueur du plus long chemin simple dans G.

Remarques

1. Reprenons l'exemple de la chaîne de la figure 7.2. La figure 7.7 donne un exemple de DAG causal sur un segment, ici le segment $[C_0, C_3]$. On voit bien sur le dessin que la couverture des événements décidants recouvre l'ensemble des processeurs. Le système de communication organisé par l'unisson structure un système de vagues pipelinées.
2. La longueur de *la plus longue marche simple dans G* est majorée par n. En suivant le théorème 101 et le théorème 103, le théorème 107 montre comment utiliser l'unisson pour calculer un infimum global, où pour résoudre le "problème $r-$paramétré".

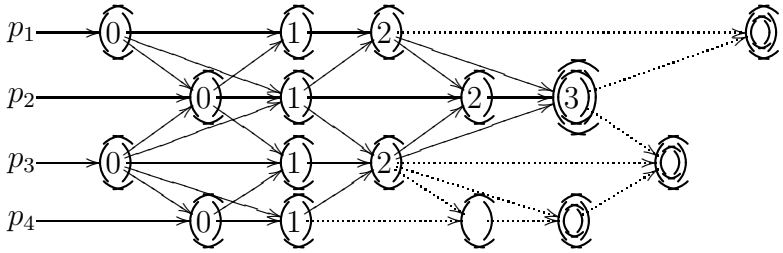


FIG. 7.7 – Unisson et structure du DAG causal sur le segment $[C_0, C_3]$

6.2 Calcul auto-stabilisant d'un infimum à distance ρ

6.2.1 Cas $\rho = d$

Pour chaque processus p, les registres $p.v_0$ et $p.res$ sont initialisés par la même valeur. Une "pulsation" est nécessaire pour cette initialisation, et D "pulsations" pour la vague de calcul. C'est pourquoi nous posons $\delta \geq d + 1$. Pour tout entier U, $[C_{U\delta}, C_{U\delta+\delta-1}]$ est une vague. Nous définissons le *code de l'application* de la manière suivante :

$$<< CA2 >> \equiv \text{ initialiation de } p.v_0 \text{ and } p.res;$$

$$<< CA1 >> \equiv p.res := p.v_0 \bigoplus \{q.res, q \in \mathcal{N}_p\}$$

D'après le théorème 101, sur la coupure $C_{U\delta+\delta-1}$ le registre $p.res$ contient la valeur correcte : $\bigoplus \{q.v_0, q \in V\}$.

6.2.2 Cas $\rho < d$

On prend $\delta = \rho + 1$. Nous supposons que le registre $p.v_0$ de chaque processus p est initialisé au début du code d'application $<< CA2 >>$ à la date $C_{U\delta}$, avant que le registre $p.r$ prenne la valeur $U\delta$. Pour atteindre notre objectif, on définit pour chaque processus p

deux registres supplémentaires $p.v_1$ et $p.v_2$. Ces registres sont initialisés à la date $C_{U\delta}$ durant l'exécution de $<< CA2 >>$, par la valeur $p.v_0$. Pour $\alpha \in \{1, 2, ..., \rho\}$, à la date $C_{U\delta+\alpha}$, l'action $<< CA1 >>$ est la séquence :

$$p.v_1 := p.v_2;$$
$$p.v_2 := p.v_0 \bigoplus \left\{q.v_{\varphi(q)}, q \in \mathcal{N}_p\right\}$$

avec ; si $q.r = p.r$ alors $\varphi(q) = 2$, et si $q.r = p.r + 1$ alors $\varphi(q) = 1$.
Pour prouver la correction de cet algorithme, commençons par prouver la proposition suivante :

Proposition 108 Pour $p \in V$ et $\alpha \in \{1, ..., \rho\}$, à la date $C_{U\delta+\alpha}$, les égalités suivantes sont vérifiées :

$$p.v_1 = \bigoplus \{q.v_0, q \in V(p, \alpha - 1)\}$$
$$p.v_2 = \bigoplus \{q.v_0, q \in V(p, \alpha)\}$$

Preuve :
A la date $C_{U\delta}$, chaque processus p vérifie $p.v_1 = p.v_0$ et $p.v_2 = p.v_0$, c'est l'étape de l'initialisation. Soit $\mathcal{A}$ l'ensemble des événements de $[C_{U\delta+1}, C_{U\delta+\delta-1}]$ pour lesquels la proposition à démontrer n'est pas vraie. On suppose que $\mathcal{A}$ nest pas vide. Soit (p, t) un événement minimal de $\mathcal{A}$. Soit $\alpha \in \{1, 2, ..., \rho\}$ tel que $(p, t) \in C_{U\delta+\alpha}$. Il existe t_0 tel que $(p, t_0) \rightsquigarrow (p, t)$. Nous avons $p^t.v_1 = p^{t_0}.v_2$ et $p^{t_0}.v_2 = \bigoplus \{q.v_0, q \in V(p, \alpha - 1)\}$, cette égalité etant vraie même si $\alpha = 1$. Par ailleurs, $p^t.v_2 = p.v_0 \bigoplus \left\{q^{t_q}.v_{\varphi(q)}, q \in \mathcal{N}_p\right\}$. En vertu de la minimalité de l'événement (p, t), les événements (q, t_q), quand $t_q < t$, ne sont pas dans $\mathcal{A}$ et sont dans $[C_{U\delta}, C_{U\delta+\delta-1}]$. Donc $p.v_0 \bigoplus \left\{q^{t_q}.v_{\varphi(q)}, q \in \mathcal{N}_p\right\} = \bigoplus \{q.v_0, q \in V(p, \alpha)\}$. On obtient une contradiction. On déduit que $\mathcal{A}$ est vide, et la proposition est démontrée.

□

Comme corollaire, on obtient le théorème attendu :

Théorème 109 Sur la coupure $C_{U\delta+\delta-1}$ le registre $p.v_2$ contient la valeur correcte : $\bigoplus \{q.v_0, q \in V(p, \rho)\}$.

7 Conclusions

Dans ce chapitre nous avons montré comment utiliser l'unisson pour définir une barrière de synchronisation auto-stabilisante à distance ρ dans un réseau anonyme. Nous avons introduit deux variantes de vagues : les vaguelettes et les vagues fortes, et nous avons montré que les vagues fortes peuvent être utilisées pour résoudre le "problème r-paramétré". Nous avons montré comment la structure des communications de l'unison peut être utilisée pour définir un courant de vagues auto-stabilisant.

Chapitre 8

Vaguelettes et démon ρ-central

Sommaire

1 Introduction

Les vaguelettes, ou ϱ-vaguelettes, présentées au chapitre 7 et dans [Bou06] traitent du calcul à distance ϱ. Bien sûr si $\varrho = 1$, il s'agit de la localité au sens de [Dij68]. Si ϱ est plus grand ou égal à d, où d est le diamètre du réseau, alors la $\varrho-$vaguelette est une vague

traitant des problèmes globaux. Les vaguelettes permettent de traiter sans calcul préalable des problèmes aussi bien globaux que locaux. Dans ce chapitre, nous montrons comment synchroniser une double horloge dans un réseau anonyme asynchrone. L'horloge principale définit un courant de vaguelettes qui assure un calcul préalable. Utilisant ces vaguelettes, l'horloge esclave définit une barrière de synchronisation à distance ϱ. Nous montrons comment utiliser cette double horloge pour résoudre efficacement les problèmes de synchronisation à distance ϱ dans les réseaux asynchrones cadencés par un démon inéquitable. Comme première instance, et afin de résoudre tous ces problèmes, nous donnons une solution efficace au problème de l'exclusion mutuelle à distance ϱ, qui permet de simuler un démon $\varrho-$central, et de définir un transformateur d'algorithmes cadencés par un démon $\varrho-$central en des algorithmes cadencés par un démon distribué. Puis nous donnons une solution spécifique à deux autres problèmes ϱ-locaux : l'exclusion mutuelle de groupe à distance $\varrho-$et le problème des ϱ-lecteurs-écrivain.

1.1 Augmenter la concurrence

La localité est une propriété essentielle des réseaux répartis. Chaque processus est connecté à d'autres processus, mais pas nécessairement à tous. Les tâches assignées au réseau peuvent être locales ou globales, silencieuses ou dynamiques. Ce qu'un processus doit connaître sur l'état du réseau peut porter sur ces voisins, qui sont donc situés à une distance 1. Mais cette connaissance nécessaire peut aussi se trouver à une distance ϱ plus grande que 1. Aussi, la synchronisation à distance plus grande que 1 est une question importante dans la littérature récente : calcul à distance ϱ [NS93], transformateur de connaissance à distance ϱ [GHJT06], philosophes dinants auto-stabilisants avec conflit générique [DNT06].
Il est souvent plus facile de définir un algorithme auto-stabilisant en utilisant un démon faible comme par exemple un démon central. Il devient alors important de pouvoir dériver de cet algorithme, un algorithme auto-stabilisant réalisant la même tâche sous un démon plus fort. Une technique bien connue pour cela est de composer l'algorithme initial avec un algorithme auto-stabilisant d'exclusion mutuelle locale. L'exclusion mutuelle locale donne le privilège à un processus d'entrer en section critique seulement si aucun de ses voisins n'a le privilège, elle assure aussi que chaque processus a le privilège infiniment souvent. Une telle composition est correcte puisque chaque processus n'écrit que dans ses propres registres internes et ne lit que les registres de ses voisins. Dans ce cas la vue de chaque processus est à distance 1.
Cependant, il peut être beaucoup plus facile de penser un algorithme local qui soit defini à distance k plus grande que 1. Dans ce cas, si on définit l'algorithme sous un démon central, il est clair que la composition avec un algorithme d'exclusion mutuelle locale ne permettra pas de passer à un démon distribué, car quand un processus exécutera sa section critique (son code d'application) les valeurs des registres des voisins à distance inférieure ou égale à k ne doivent pas changer. On ne peut augmenter la concurrence aussi facilement que dans le cas $k = 1$. Un exemple classique est de chercher sur un graphe un ensemble maximal *2-indépendant* de manière auto-stabilisante. Cela consiste à chercher un ensemble maximal de nœuds V', tel qu'il n'y ait pas deux nœuds de V' adjacents ni deux nœuds de V' qui aient

un nœud voisin commun, ni des voisins de ces deux nœuds, qui soient eux-même voisins ; ce qui s'énonce plus simplement par la condition que deux nœuds différents de V' sont à distance strictement plus grande que deux l'un de l'autre. Ce problème est une extension du problème classique de la recherche d'un ensemble maximale indépendant. Afin d'illustrer notre propos, présentons d'abord le problème de la recherche d'un *ensemble indépendant maximal*, puis présentons ses extensions.

1.2 Exemple : la recherche d'un ensemble maximal indépendant

Soit $G = (V, E)$ un graphe. Un ensemble indépendant de G est un sous-ensemble V' de V, tel que deux sommets quelconques de V' ne soient pas adjacents. Un ensemble indépendant maximal est un ensemble indépendant du graphe G qui est maximal au sens de l'inclusion.

1.2.1 Une solution élégante

Il y a une jolie solution avec connaissance à distance 2 [SRR95] sous un démon central. Cette solution autorise un calcul parallèle pourvu que deux processus activés simultanément n'aient pas un voisin commun. Chaque processus p maintient un regitre $p.fc$ à valeurs dans $0, 1$. Ce registre défini la fonction caractéristique de l'ensemble cherché. Le protocole ést défini par l'algorithme 18. L'action IA est une action de correction, l'action CA est une action de constuction de l'ensemble.

Algorithme 18 Calcul d'un ensemble indépendant maximal.

Constante and variable :

$\mathcal{N}_p$: l'ensemble des voisins de p ;

$p.fc$: à valeurs dans $0, 1$;

Fonctions booléennes :

$Choix_p \equiv \forall q \in Np, q.fc = 0 \wedge p.fc = 0$;

$Incorrect_p \equiv (p.fc = 1) \wedge (\exists q \in \mathcal{N}_p : (q.fc = 1)$;

Action :

CA : $Choix_p \longrightarrow p.fc := 1$;

IA : $Incorrect_p \longrightarrow p.fc := 0$;

Avec ce protocole (algorothme 18), sous un démon central, chaque processus fait au plus une fois l'action de correction (IA) et au plus une fois l'action de choix (CA). Le protocole fait converger le système en au plus $2n$ steps. Il est alors facile de voir que l'ensemble des sommets p défini par $p.fc = 1$ est un ensemble indépendant maximale ; indépendant car l'action (IA) n'est activable sur aucun processus et maximal car l'action (CA) n'est activable sur aucun processus.

L'exclusion mutuelle locale permet de gérer cette contrainte "à distance 1". Elle permet de paralléliser l'algorithme précédent sous un démon asynchrone quelconque.

1.2.2 Généralisation : recherche d'un ensemble maximal k-indépendant

Une généralisation immédiate du problème précédent est la recherche d'un ensemble maximal k-indépendant d'un réseau $G = (V, E)$, avec k un entier naturel non nul. Un ensemble $k-$indépendant est un sous-ensemble V' de V, tel que deux sommets quelconques de V' soient toujours à distance strictement supèrieure à k. Un ensemble $k-$indépendant maximal est un ensemble $k-$indépendant du graphe G, maximal au sens de l'inclusion. Remarquons qu'un ensemble indépendant est un ensemble $1-$indépendant.
Dans [GHJT06] les auteurs proposent un transformateur général pour construire des algorithmes auto-stabilisants qui utilisent un savoir à distance k. Ils appliquent en particulier leur transformateur à la détermination d'un ensemble $k-$indépendant maximal d'un réseau G. A nouveau, cet algorithme fonctionne sous un démon central. Il est intéressant d'augmenter la concurrence.
Cette concurrence potentielle n'est pas la même que celle énoncée au paragraphe précédent. Le problème de la détermination d'un ensemble $k-$indépendant maximal autorise un calcul parallèle pour deux processus si leur distance mutuelle est strictement supèrieure à k. Il ne s'agit plus ici d'exclusion mutuelle locale, mais d'exclusion mutuelle à distance k. C'est-à-dire assurer que deux processus ne soient pas activables simultanément si ils sont à une distance inférieure ou égale à k l'un de l'autre.

Algorithme 19 Calcul d'un ensemble maximal k-indépendant.

Constante and variable :

$\mathcal{N}_p^k$: l'ensemble des processus à distance inférieure ou égale à k de p ;

$p.fc$: à valeurs dans $0, 1$;

Fonctions booléennes :

$Choix_p \equiv \forall q \in \mathcal{N}_p^k, q.fc = 0 \wedge p.fc = 0$;

$Incorrect_p \equiv (p.fc = 1) \wedge (\exists q \in \mathcal{N}_p^k : (q.fc = 1)$;

Action :

CA : $Choix_p \longrightarrow p.fc := 1$;

IA : $Incorrect_p \longrightarrow p.fc := 0$;

1.3 Calcul local, concurrence et ϱ-exclusion mutuelle locale

Plus généralement, la notion de calcul local est développée dans [NS93]. [GMM04] propose un modèle de calcul par un système de réécriture local de graphes étiquetés . La localité signifie que chaque étape de réécriture change les étiquettes de sous-graphes connexes de rayon fixé ϱ, le changement des étiquettes dépendant des valeurs de toutes les étiquettes du sous-graphe. Dans ce cas, la réécriture autorise aussi un calcul parallèle pourvu que entre deux boules qui exécutent une étape indépendante de calcul ne se recouvrent pas. Dans ce cas l'exclusion mutuelle locale à distance 2ϱ résout le problème de la concurrence avec cette nouvelle contrainte.
Nous faisons face à deux problèmes distincts, le premier est de construire un tranformateur qui autorise seulement un calcul parallèle à distance supérieur à ϱ. Le deuxième consiste à résoudre le problème de la simulation du calcul à distance ϱ.

Dans ce chapitre nous n'aborderons pas la question du calcul à distance ϱ. Nous n'aborderons que la question de l'exclusion mutuelle à distance ϱ, ou plus simplement, la ϱ-LME. On peut voir la ϱ-LME comme un problème d'allocation de ressources ou comme un problème de synchronisation locale à distance ϱ. Nous utiliserons ces deux points de vue simultanément. L'exemple de la $\varrho-$LME montre combien la synchronisation à distance ϱ est riche de conséquences. Beaucoup de problèmes de partage de ressources synchronisés ont été étudiés d'un point de vue global, mais aussi du point de vue local. Dans ce chapitre nous proposons d'élargir cette discussion en prenant le point de vue plus général de la synchronisation à distance ϱ. Nous considérons que les $\varrho-$vaguelettes sont un bon outil pour aborder ces problèmes.

1.4 Contributions.

Dans ce chapitre, nous utilisons un outil simple appelé *vaguelette* (ou *ϱ-vaguelette*). Les $\varrho-$vaguellettes permettent la coordination de processus à distance au plus ϱ. Elles sont liées à la notion de *vague* ([Tel90, Tel94]).
Au chapitre 7, nous avons présenté une solution auto-stabilisante efficace du problème de l'implémentation d'une ϱ-vaguelette. Il y a plusieurs méthodes pour implémenter une vaguelette, elles dépendent des propriétés du réseau. Par exemple, dans [DNT06] les auteurs, supposant que le réseau est avec identités (distinctes), proposent une solution en combinant une propagation de vagues avec retour auto-stabilisante (PIF) [BDPV99] sur des arbres couvrants en largeur auto-stabilisants [HC92, Joh97] enraciné sur des processus à distance ϱ les uns des autres. Ils proposent ainsi un schéma de ϱ-wavelets. Par contre, notre solution ne requiert aucune structure sous-jacente et fonctionne sur un réseau uniforme quelconque. Notre solution est basée sur l'unisson et fonctionne sous un démon distribué quelconque.
Dans ce chapitre, nous utilisons les vaguelettes pour constituer une double horloge auto-stabilisante. L'horloge du "dessous", appelée horloge maître, définit le courant de vaguelettes. L'horloge du "dessus", appelée horloge *esclave*, détermine un mécanisme de *barrière* de synchronisation à distance ϱ , où aucun processus p ne commence à exécuter sa phase $i+1$ tant que tous les processus de la $\varrho-$boule centrée en p n'ont pas complété leur phase i.
Finalement, nous montrons que cette double horloge fournit un outil de couche basse efficace pour implémenter de nombreux problèmes d'allocation de ressources à distance ϱ. Ces problèmes incluent l'exclusion mutuelle [Dij65], l'exclusion mutuelle de groupe[Jou00], les lecteurs-écrivain [CHP71]. Certaines de ces solutions (ϱ-LME) fournissent un transformateur qui permet d'utiliser des algorithmes écrits pour un démon ϱ-central dans des systèmes fonctionnant sous n'importe quel démon distribué.

1.5 Structure du chapitre

Dans la section 2, nous présentons quelques problèmes d'allocation de ressources, d'abord d'un point de vue global, puis d'un point de vue local. Nous présentons une généralisation

de ces problèmes à distance ϱ. Dans la section 3, nous introduisons la double horloge autostabilisante, et nous montrons comment elle peut être utilisée pour résoudre des problèmes de coordination à distance ϱ. Dans la section 4 nous présentons un algorithme de d'exclusion mutuelle locale à distance ϱ, nous donnons notament une étude détaillée de sa complexité. Nous présentons aussi une solution pour le problème plus général de l'exclusion mutuelle de groupe à distance ϱet ue solution pour les lecteurs écrivain à distance ϱ. Des remarques de conclusion sont données dans la section 5.

2 Exemples de problèmes d'allocation de ressources

Définition 57 [Graphe de compatibilité [CDP03]] Soit A un ensemble, parfois appelé ensemble des ressources. On appelle relation de compatibilité toute relation binaire réflexive R sur l'ensemble A. On dit que la relation binaire R est une relation de compatibilité sur A. Si $(a, b) \in R$, alors nous disons que a et b sont compatibles ou partageables. Si $(a, b) \notin R$ alors nous disons que a et b sont incompatibles, ou en conflit.

La spécification du problème général de l'allocation de ressources est la suivante :

Vivacité : dans toute exécution, si deux processus entrent et exécutent leur section critique simultanément alors ils utilisent tous les deux des ressources compatibles.

Sûreté : Si un processus demande une ressource dans A pour entrer en section critique , alors sa requête sera satisfaite et il entrera en section critique au bout d'un temps fini.

Voici quelques exemples bien connus.

1. Point de vue global.
   (a) **Exclusion mutuelle** : A est l'ensemble des processus et $R = \{(a, a), a \in A\}$. La condition de sûreté est : *dans toute exécution, à aucun moment deux processus n'exécutent leur section critique simultanément* .
   (b) **Lecteurs-écrivain** : A est l'ensemble des 2-uples $\{(p, r), p \in V, r \in \{read, write\}\}$, et R est défini par la condition de sûreté :
   dans toute exécution, si deux processus exécutent leur section critique simultanément alors ils exécutent tous les deux une opération de lecture.
   (c) **Exclusion mutuelle de groupe** : R est une relation d'équivalence sur un ensemble de ressources A. R est défini par la condition de sûreté : *dans toute exécution, si deux processus exécutent leur section critique simultanément, alors ils utilisent tous les deux des ressources qui sont dans la même classe d'équivalence.*
   (d) **Allocation de ressources** : R est une relation binaire générale sur un ensemble de ressources A. La condition de sûreté est : *dans toute exécution, si deux processus exécutent leur section critique simultanément, alors ils utilisent des ressources compatibles.*

2. Voici maintenant une généralisation à distance ϱ. L'idée est de restreindre la vue à une distance ϱ. Nous appelons ce point de vue, le point de vue ϱ-local. Bien sûr, si $\varrho = D$ on trouve le point de vue global, et si $\varrho = 1$ on trouve le point de vue local où les conflits ne se posent qu'entre processus voisins. Le plus populaire des problèmes présentés est celui du *dîner des philosophes*, aussi appelé *exclusion mutuelle locale.*

   (a) **ϱ-exclusion mutuelle locale** :
   A est l'ensemble des processus et $R = \{(a,b) \in A^2, d(a,b) > \varrho \text{ ou } a = b\}$

   (b) **ϱ-local lecteurs-écrivain** :
   A est l'ensemble des 2-uples $\{(p,r), p \in V, r \in \{read, write\}\}$, et R est défini par la condition de sûreté : *dans toute exécution, si deux voisins à distance inférieure ou égale à ϱ exécutent leur section critique simultanément, alors ils exécutent tous les deux une action de lecture.*

   (c) **ϱ-exclusion mutuelle locale de groupe** : R est une relation d'équivalence sur l'ensemble des ressources A . R est défini par la condition de sûreté : *dans toute exécution, si deux voisins à distance inférieure ou égale à ϱ exécutent leur section critique simultanément, alors ils utilisent tous les deux des ressources qui sont dans la même classe d'équivalence.*

   (d) **ϱ-allocation de ressources locale** : R est une relation binaire générale sur l'ensemble des ressources A. La condition de sûreté est : *dans toute exécution, si deux voisins à distance inférieure ou égale à ϱ exécutent leur section critique simultanément, alors ils utilisent tous les deux des ressources qui sont compatibles.*

Il y a de nombreux exemples qui motivent cette généralisation, comme le calcul local à distance ϱ [GMM04], ou le problème des interférences dans les réseaux de capteurs [DNT06]. Il est clair que l'exclusion mutuelle globale est un moyen de résoudre tous ces problèmes ; la question est : comment augmenter la concurrence. Un bon moyen générique est d'implémenter la $\varrho-$exclusion mutuelle locale. C'est pourquoi nous développerons en détail une solution à ce problème, avec une analyse complète de complexité. Nous donnerons ensuite une intéressante solution à deux autres problèmes : le problème plus général de la ϱ-exclusion mutuelle de groupe et le problème des ϱ-lecteurs-écrivain.

3 Schéma de programme à distance ϱ

3.1 Double horloge auto-stabilisante.

L'idée est de coordonner un courant de $\delta-$vaguelettes avec $\delta \geq \varrho$. Si $\delta > \varrho$ les temps supplémentaires peuvent être utilisés à une initialisation par exemple. Une horloge organise ce flux. Les vaguelettes sont utilisées pour calculer en parallèle un infimum local sur chaque boule de rayon ϱ. Pour chaque processus p, une fois que l'infimum est calculé, une deuxième horloge définit une notion de retard sur le réseau. Ce retard est un préordre total sur le

réseau. Ce préordre sera utile pour organiser l'entrée en section critique de chacun des processus.
Plus formellement, nous définissons deux horloges sur chaque processus, la première horloge est notée C_1 (l'horloge maîtresse) et la seconde est notée C_2(l'horloge esclave) d'ordre respectif K_1 et K_2, en général K_1 est un multiple de δ. Les systèmes d'incrémentation sont respectivement (χ_1, φ_1) et (χ_2, φ_2). Le comportement de la deuxième horloge est ordonnancé par la première horloge et par un prédicat noté *cond*. Le prédicat *cond* dépend du problème à résoudre. Pour distinguer les deux horloges, les registres sont indicés par 1 ou 2 respectivement pour l'horloge maîtresses et l'horloge esclave. Les prédicats sont notés avec les exposants 1 ou 2 respectivement. Par exemple, le registre de l'horloge maîtresse est noté r_1 et le registre de l'horloge esclave est noté r_2. Le prédicat $NormalStep_p^1$ est défini sur le registre r_1 du processus p ; le prédicat $NormalStep_p^2$ est défini sur le registre r_2 du processus p. Nous définissons de la même manière Γ_1^1, Γ_1^2, et $\Gamma_1 = \Gamma_1^1 \cap \Gamma_1^2$. Stabiliser la double horloge revient à garantir $\Gamma \rhd \Gamma_1$.
Une fois le système stabilisé, le cadencement de l'horloge esclave du processus p est défini dans $<< CS2 >>$ par une action gardée :

$$NormalStep_p^2 \wedge cond \rightarrow << CS2 >>; p.r_2 := \varphi_2\,(p.r_2) \tag{8.1}$$

où *cond* est un prédicat indépendant du registre $p.r_1$. C'est ce prédicat qui exprime le problème de synchronisation à distance ϱ à résoudre. Parallèlement, les procédures *Initialization* et *Computation* sont ordonnancées par la vaguelette, et fournissent un calcul préalable utilisé par *cond*. Les procedures *Initialization* et *Computation* dépendent aussi du problème à résoudre. Nous donnerons des exemples plus loin.
Définissons le prédicat $NormalStep_p$ par $NormalStep_p \equiv NormalStep_p^1 \wedge LocallyCorrect_p^2$.
La double horloge utilise essentiellement une composition hiérarchisée de deux unissons du type SS_AU.

Lemme 110 Les prédicats Γ_1^1, Γ_1^2 et Γ_1 sont clos.

Proposition 111 L'ensemble des états des horloges C_1 converge vers Γ_1^1.

Preuve : Soit $e = \gamma_1.....\gamma_k....$ une exécution maximale. Supposons que l'exécution e est finie. Alors le dernier état γ_l est un état final, un interblocage. Ainsi, l'horloge C_1 est stabilisée, car sinon il existerait un processus pour lequel l'action CA_1 ou RA_1 serait exécutable.
On suppose maintenant que e est infini. La projection e_1 de e sur les registres r_1 est une exécution de l'horloge C_1. Si e_1 est fini, alors le dernier état C_1 est stabilisé pour les mêmes raisons qu'au dessus. Si e_1 n'est pas fini, alors e_1 est une exécution infinie de C_1, donc daprès [BPV04] ou le chapitre 3 il y a un état qui est dans Γ_1^1.

□

Proposition 112 Les horloges C_2 convergent vers Γ_1^2 .

Preuve : Soit $e = \gamma_1.....\gamma_k....$ une exécution maximale. Grâce à la proposition 111 , on peut supposer que $\gamma_1 \in \Gamma_1^1$. Alors, pour chaque processus, tant que C_2 n'est pas stabilisé

Algorithme 20 ($SS-DC$) double horloge auto-stabilisante

Constante et variables :

$\mathcal{N}_p$: ensemble des voisins du processus p ;

$p.r_1 \in \chi_1$; $p.r_2 \in \chi_2$;

fonctions booléennes :

Pour l'horloge $i \in \{1,2\}$:

$ConvergenceStep_p^i \equiv p.r_i \in tail_{\varphi_i}^* \wedge (\forall q \in \mathcal{N}_p : (q.r_i \in tail_{\varphi_i}) \wedge (p.r_i \leq_{tail_{\varphi_i}} q.r_i))$;

$LocallyCorrect_p^i \equiv p.r_i \in stab_{\varphi_i} \wedge$
$\forall q \in \mathcal{N}_p, q.r \in stab_{\varphi_i} \wedge ((p.r_i = q.r_i) \vee (p.r_i = \varphi_i(q.r_i)) \vee (\varphi_i(p.r_i) = q.r_i))$;

$NormalStep_p^i \equiv p.r_i \in stab_{\varphi_i} \wedge (\forall q \in \mathcal{N}_p : (p.r_i = q.r_i) \vee (q.r_i = \varphi(p.r_i)))$;

$ResetInit_p^i \equiv \neg LocallyCorrect_p^i \wedge (p.r_i \notin init_{\varphi_i})$;

prédicat commun :

$NormalStep_p \equiv NormalStep_p^1 \wedge LocallyCorrect_p^2$;

Actions :

NA : $NormalStep_p \longrightarrow$ **if** $p.r_1 \equiv \delta - 1[\delta]$ **then**
Begin
$NormalStep_p^2 \wedge cond \rightarrow$ $<<$ **CS 2** $>>$;
if $cond_1$ **then** $p.r_2 := \varphi_2(p.r_2)$
$Initialization$;
End
else $Computation$;
$p.r_1 := \varphi_1(p.r_1)$;

For clock $i \in \{1,2\}$ **:**

CA_i : $ConvergenceStep_p^i \longrightarrow p.r_i := \varphi_i(p.r_i)$;

RA_i : $ResetInit_p^i \longrightarrow p.r_i := -\alpha_i$ **(reset)** ;

localement, C_1 n'exécute aucune action. Ainsi la projection e_2 de e sur les registres r_2 est une exécution de l'horloge $C\ _2$. Si e_2 est fini, alors le dernier état C_2 est stabilisé, ou sinon il existerait un processus pour lequel l'une des actions CA_2 ou RA_2 est exécutable. Si e_2 n'est pas fini, alors e_2 est une exécution infinie de C_2, donc d'après [BPV04] et le chapitre 3 il y a un état qui est dans Γ_1^2.

□

A partir des propositions 111 et 112 , nous déduisons ce corollaire :

Corollaire 1 La double horloge stabilise vers Γ_1.

Proposition 113 [Absence de famine pour l'horloge C_1] Une fois stabilisée, l'horloge C_1 s'incrémente indéfiniment .

Preuve : Soit $e = \gamma_1.....\gamma_k....$ une exécution maximale. Grâce au corollaire 1 de la proposition 112, on peut supposer que $\gamma_1 \in \Gamma_1$.
Supposons que pour un processus p, l'action NA n'est exécutée qu'un nombre fini de fois. Alors les horloges C_1 de toutes les processus ne sont exécutées qu'un nombre fini de fois, donc e est fini. Mais alors, dans le dernier état de e, les processus minimaux pour la relation de précédence sont activables pour l'action NA, ce qui est une contradiction.

□

3.2 Comparaison locale à distance ϱ

Dans le réseau, une fois que les doubles horloges sont toutes stabilisées, le retard entre deux processus relativement à la deuxième horloge définit un préordre sur l'ensemble des processus. Malheureusement, le retard est une notion globale. Pour pouvoir l'évaluer par un processus, il faut trouver une condition telle pour deux processus qui sont dans une même boule de rayon ϱ, il soit possible pour les deux de calculer le retard entre l'un et l'autre avec seulement la connaissance des valeurs des registres r_2 des deux processus. Pour pouvoir comparer deux processus se trouvant dans une même boule de rayon ϱ, Il suffit d'être capable de comparer les horloges esclaves de n'importe quel couple de processus à distance au plus 2ϱ l'un de l'autre. Ainsi, dans chaque boule B de rayon ϱ, nous serons capable de définir un préordre parmi les processus dans B en comparant les valeurs de leurs registres r_2. Bien sûr, il est important que le préordre total soit le même que celui défini par le retard des horloges r_2.
Pour atteindre cet objectif, il suffit d'utiliser une comparaison locale en prenant $M = 2\varrho$, voir chapitre 2, page 57. Pour que cette relation locale soit bien définie, il est nécessaire de prendre $K > 2M$, c'est pourquoi nous supposerons à partir de maintenant que $K > 4\varrho$. Rappelons que l'ordre local sur $\chi = \{0, ..., K-1\}$ est défini par :

$$\forall (a,b) \in \chi^2 :\ a \leq_l b \Leftrightarrow_{def} 0 \leq \overline{b-a} \leq 2\varrho. \tag{8.2}$$

Dans ces conditions, pour tout état $\gamma \in \Gamma_1$, deux processus qui sont dans une boule de rayon 2ϱ ont des registres dont les valeurs sont localement comparables. Cette comparaison coïncide avec l'ordre défini par la notion de retard définie par la deuxième horloge, comme le montre le lemme suivant :

Lemme 114 Pour tout état $\gamma \in \Gamma_1$, si p et q sont deux processus satisfaisant $d(p,q) \leq 2\varrho$, alors, en posant $a = p.r_2$ et $b = q.r_2$, le retard $\varrho_{p,q}$ est égal à $\overline{b-a}$ si $0 \leq \overline{b-a} \leq 2\varrho$ et est égal à $-\overline{a-b}$ sinon.

Preuve : De $d(p,q) \leq 2\varrho$, on déduit que $\varrho_{p,q} \in \{-2\varrho, ..., 2\varrho\}$.
De l'identité $\varrho_{p,q} \equiv \overline{b-a}\,[K]$ et du fait que $K > 4\varrho$, on déduit que :

1. si $0 \leq \overline{b-a} \leq 2\varrho$ alors $\varrho_{p,q} = \overline{b-a}$
2. sinon $\varrho_{p,q} = -\overline{a-b}$

□

Grâce à cette construction et grâce au lemme 114, pour tout couple de processus situés dans une boule B de rayon ϱ, nous avons accès à la notion de retard définie par les horloges esclaves des deux processus. Le problème que nous nous étions posé est maintenant résolu. Dans la suite de ce chapitre, nous supposerons que $\delta = \varrho + 1$, $K_1 = \delta K$ avec $K_1 \geq C_G - 1$ et $K_2 \geq \max(4\varrho + 1, C_G - 1)$. Ces hypothèses assurent que la double horloge est autostabilisante, que l'horloge maîtresse est calibrée pour définir un flux de $\varrho-$vaguelettes, et que le retard défini entre deux processus par les horloges esclaves est calculable à distance au plus 2ϱ avec la seule connaissance des contenus des registres des deux horloges.

3.3 Usage de la double horloge

Chaque processus p maintient trois registres en plus des registres d'horloge :

1. le registre $p.v$: Σ, où Σ est n'importe quel type de données. C'est la ressource demandée.
2. Deux registres $p.res_1$: $\chi_2 \times \Sigma$ et $p.res_2$: $\chi_2 \times \Sigma$, ils serviront à faire le pré-calcul d'infimum à distance ϱ durant chaque vaguelette.

On fait l'hypothèse qu'une relation d'ordre total $\preceq$ est définie sur l'ensemble Σ des ressources. On accède aux deux champs du registre $p.res_i$ par $p.res_i.r$ et $p.res_i.v$ respectivement, avec $i \in \{1, 2\}$.
Au premier temps de la vaguelette, c'est-à-dire avant le pré-calcul, $p.res_i$ est initialisé dela manière suivante :

$$Initialization \equiv p.res_1 := (p.r_2, p.v)\ ;\ p.res_2 := (p.r_2, p.v)$$

On définit maintenant un ordre $\varrho-$local sur $\chi_2 \times \Sigma$ par :

$$(r,v) \blacktriangleleft (r',v') \Leftrightarrow ou \begin{array}{l} r <_l r' \\ r = r' \text{ et } \bar{v} \preceq \bar{v}' \end{array}$$

Rappelons que $<_l$ est défini par la notion de retard, qui est un préordre total, ce préordre étant calculable à distance ϱ. On définit l'opérateur d'infimum $\varrho-$local associé de la manière suivante :

$$(r,v) \oplus (r',v') = \begin{array}{l}(r,v) \text{ si } (r,v) \blacktriangleleft (r',v') \\ (r',v') \text{ sinon}\end{array}$$

Maintenant nous calculons l'infimum sur toutes les boules en parallèle. Pour cela nous utilisons la méthode développée dans le chapitre 7 page 161 Ce pré-calcul de l'infimum à distance ϱ est défini par l'action :

$$Computation \equiv p.res_1 := p.res_2\,;\, p.res_2 := (p.r_2, p.v) \bigoplus \left\{q.res_{\nu(q)}, q \in \mathcal{N}_p\right\}$$

avec : si $q.r_2 = p.r_2$ alors $\nu(q) = 2$ et si $q.r_2 = \overline{p.r_2 + 1}$ alors $\nu(q) = 1$.
Le pré-calcul d'un infimum local désigne un "gagnant" q dans la boule de rayon ϱ centrée en p. Il est important de comprendre que ce gagnant est élu en p, et peut-être pas par un autre processus à distance ϱ de q. C'est l'infimum dans la boule de rayon ϱ centrée en p, mais peut-être pas l'infimum pour une autre boule de rayon ϱ contenant q.
On pourrait penser qu'il y a ici un risque de famine ou d'interblocage, mais il n'en est rien, comme nous allons le voir, du fait du préordre total managé par la deuxième horloge.
La condition *cond* du problème à résoudre, c'est la disjonction :

$$((p.r_2, p.v) = p.res_2) \vee cond_2$$

Où $(p.r_2, p.v) = p.res_2$ signifie que p est élu dans la boule de rayon ϱ centrée en p ; la condition $cond_2$ est là pour augmenter la concurrence, elle dépend du problème posé. Si p n'est pas élu par le calcul d'infimum $\varrho-$local et si $cond_2$ est vrai, il y a passage en section critique du processus p, mais l'incrémentation peut ne pas être désirée. La condition $cond_1$, qui conditionne l'incrémentation de l'horloge esclave, est définie par $p.r_2 = p.res_2.r$. Elle assure que la deuxième horloge reste synchronisée. Ce prédicat est important pour assurer la cohérence avec un maximum de concurrence.
Afin de prouver l'absence de famine et d'interblocage, nous allons utiliser la technique du relèvement détaillée au chapitre 2 page 62. Nous relevons l'horloge maîtresse en utilisant les mêmes notations que dans le chapitre 2, excepté le fait que le registre r devient le registre r_1.

Lemme 115 [Absence d'interblocage] Durant chaque phase $[C_{U\varrho}, C_{U\varrho+\varrho-1}]$, il y a au moins un processus qui est élu et qui incrémente son horloge esclave.

Preuve : L'ensemble des processus est fini. Pour la relation d'ordre $\preceq$, il y a un infimum parmi les couples $(p.r_2, p.v)$ quand p parcourt l'ensemble des processus V. Si le processus q est le processus qui réalise cet infimum, q est élu durant la phase $[C_{U\varrho}, C_{U\varrho+\varrho-1}]$ et comme alors $q.r_2 = q.res_2.r$, le prédicat $cond_1$ est vrai et $q.r_2$ s'incrémente.

□

Lemme 116 [Absence de famine] Tout processus incrémente son registre r_2 une infinité de fois, et a donc le "privilège" une infinité de fois .

Preuve : Il suffit de montrer que chaque processus p a le privilège au moins une fois. Supposons que le processus p n'ait jamais le privilège. Posons $Pot_p = \sum_{q \in V} \Delta_{pq}$, c'est la somme des retards du processus p par rapport à tous les autres processus du réseau.
Cette quantité est un entier relatif et d'après l'hypothèse, cette quantité est strictement croissante puisqu'il n'y a pas d'interblocage et qu'à chaque phase au moins un processus s'incrémente. Ainsi la quantité Pot_p n'est pas majorée. Or on sait qu'elle est aussi majorée par $|V|\, d$. Ce qui conduit à une contradiction ; le lemme en découle.

□

4 Synchronisation à distance ϱ auto-stabilisante

4.1 Algorithme d'exclusion mutuelle à distance ϱ

Dans cette partie, chaque processus p possède un identificateur noté $p.id$. La valeur de $p.id$ est dans un ensemble totalement ordonnés $\mathbb{S}$, par exemple l'ensemble des entiers. En fait, les identités peuvent correspondre seulement à une bonne coloration à distance 2ϱ c'est-à-dire une coloration telle que deux nœuds à distance inférieure ou égale à 2ϱ ; n'aient pas la même couleur. On définit pour chaque processus p le registre $p.v = p.id$. Le couple $(p.r_2, p.id)$ est défini pour chaque processus p. La condition $cond_2$ est définie le prédicat $false$ et bien sûr $cond_1$ est défini par le prédicat $\equiv true$. De la section 3.3, on déduit que l'algorithme est sans interblocage et sans famine. Il reste à prouver la conditionde de sûreté.

Lemme 117 [Sûreté] Si le processus p a le privilège, alors aucun processus situé à une distance inférieure ou égale à ϱ de p n'a le privilège simultanément.

Preuve : Supposons que p ait le privilège durant la phase $[C_{U\varrho}, C_{U\varrho+\varrho-1}]$. Il entre en section critique quand son registre satisfait $\widetilde{p}.r_1 = U\varrho + \varrho - 1$. Tout autre processus q à distance inférieure ou égale à ϱ de p n'a pas le privilège durant la phase $[C_{U\varrho}, C_{U\varrho+\varrho-1}]$. Ainsi, si $q \in B_\varrho(p)$ a le privilège simultanément, il ne peut l'avoir durant la même phase. La valeur absolue du retard, au sens de la première horloge (l'horloge maîtresse), entre deux processus est inférieure ou égale à ϱ. Mais quand p entre en section critique, et tant que p est en section critique : $\widetilde{q}.r_1 \in \{U\varrho + \varrho - 1 - \varrho, ..., U\varrho + \varrho - 1 + \varrho\} = \{U\varrho, ..., U\varrho + \varrho - 1 + \varrho\}$, et $U\varrho + \varrho - 1 + \varrho < (U+1)\,\varrho + \varrho - 1$. On en déduit que q ne peut entrer en section critique simultanément que si et seulement si $\widetilde{q}.r_1 = U\varrho + \varrho - 1$, c'est-à-dire durant la même phase que p, ce qui conduit à une contradiction.

□

De ces propriétés on conclut :

Théorème 118 L'algorithme $\varrho - LME$ résout de manière auto-stabilisante la ϱ−exclusion mutuelle locale.

Définition 58 On dit que la $\varrho - LME$ a pour indice d'équité k, si dans tout calcul, entre deux entrées en section critique d'un processus , n'importe quel autre processus n'entre pas en section critique plus de k fois. Le temps de service de l'algorithme de $\varrho - LME$ est le nombre maximal d'entrées en section critique des autres processus entre deux entrées en section critique d'un quelconque des processus. .

Notre algorithme de $\varrho - LME$ est une barrière de synchronisation à distance ϱ. On en déduit :

Proposition 119 Pour notre algorithme de $\varrho - LME$, l'indice d'équité est $\left\lceil \frac{d}{\varrho} \right\rceil$, et le temps de service est majoré par $\left\lceil \frac{n(n-1)}{\varrho} \right\rceil$

De la définition d'une phase, suivant [BPV04], et puisque K_1 est dans $O(\varrho d) \subset O(d^2)$, K_2 est dans $O(d)$, et donc $K_1 K_2$ est dans $O(d^3)$ respectivement. on déduit :

Proposition 120 Durant une phase, le nombre de communications entre processus est $2(\varrho + 1)|E|$ où $|E|$ est le nombre d'arêtes du réseau.
Le temps de convergence vers Γ_1 de notre algorithme de $\varrho - LME$ est dans $O(n)$ rondes.
La complexité en espace de notre algorithme de $\varrho - LME$ est dans $O(\log d)$.

Un algorithme de $\varrho - LME$ est un algorithme de LME. Une intéressante conséquence est que la $\varrho - LME$ est une technique pour réduire le temps de service et l'indice d'équité. Bien sûr, le prix à payer est un accroissement des communications entre les processeurs.

4.2 Deux autres synchronisations à distance ϱ

4.2.1 ϱ−exclusion mutuelle de groupe.

Soit Σ l'ensemble des ressources. Supposons qu'un préordre $\preceq$ soit défini sur Σ, un préordre arbitraire, par exemple un ordre de priorité.
La relation binaire sur Σ, définie par : $x \asymp y$ si et seulement si $x \preceq y$ et $y \preceq x$, est une relation d'équivalence. Les classes d'équivalence sont les groupes. L'ensemble des groupes est l'ensemble quotient $\frac{\Sigma}{\asymp}$; $p.v$ prend ses valeurs dans $\frac{\Sigma}{\asymp}$. Une autre manière de dire les choses est que $\preceq$ est un ordre total sur les groupes. Afin d'augmenter la concurrence, si p demande la ressource a et si le processeur élu à distance ϱ demande une ressource dans le même groupe, alors p entre en section critique. Le prédicat *cond* est défini par : $\overline{p.v} = \overline{p.res.v}$.
Remarquons que nous ne faisons aucune hypothèse sur l'existence d'identités sur les processus. Cependant, nous faisons l'hypothèse supplémentaire qu'il existe un ordre total sur les groupes de ressources. Par exemple, le problème de l'exclusion mutuelle locale est une instance de l'exclusion mutuelle de groupe où les ressources sont les processus. Nous faisons donc l'hypothèse de l'existence d'un ordre total sur les processus, ce qui revient à faire l'hypothèse de l'existence d'identités sur les processus.
D'après la section 3.3, notre algorithme est sans interblocage et sans famine. Nous devons prouver la propriété de sûreté. Par construction :

Lemme 121 [Sûreté] Si le processus p et le processus q ont le privilège simultanément, et sont à une distance inférieure ou égale à ϱ l'un de l'autre, les ressources demandées par les deux processus sont dans le même groupe.

Nous concluons que :

Théorème 122 Notre algorithme de $\varrho-$exclusion mutuelle de groupe résout le problème de la $\varrho-$*exclusion mutuelle de groupe* auto-stabilisante.

4.2.2 Le problème des $\varrho-$lecteurs-écrivain.

Afin de pouvoir comparer deux registres r à distance ϱ, on suppose (comme précédemment) $K \geq 4\varrho + 1$. Pour toute boule B de rayon ϱ , l'ordre local $\leq_l$ définit un préordre sur les registres r des processus dans B.
Nous supposons que chaque processus possède une identité $p.id$ entière (ou à valeur dans tout autre ensemble totalement ordonné). Chaque processus peut faire une des trois requêtes suivantes :

1. N : le processus de demande rien ;
2. R : le processus demande à pouvoir lire ;
3. W : le processus demande à pouvoir écrire .

Si un processus p fait la requête N ou R, alors le registre $p.v$ est initialisé par le constructeur constant F. Si p demande W, alors $p.v$ est initialisé par le constructeur non constant $Wp.id$.
Considérons maintenant le type "$F|W$ of int" . L'ordre $\preceq$ sur Σ l'ensemble des objets du type : $F|W\ of\ int$ est défini par :

$$v \preceq v' \Leftrightarrow \begin{array}{l}(v = F \text{ et } v' = F) \text{ ou } (v = Wn \text{ et } v' = F)\\ \text{ou } (v = W\ n \text{ et } v' = W\ n' \text{ avec } n \leq n')\end{array}$$

Pour chaque processus p, le prédicat *cond* est défini par le filtrage par cas sur $p.res$ suivant :

$$\begin{array}{ll}(r, F) \text{ when } r = p.r_2 & \rightarrow true\\ |\,(r, Wid) \text{ when } r = p.r_2 \text{ and } p \text{ askes } N & \rightarrow true\\ |\,(r, Wid) \text{ when } r = p.r_2 \text{ and } (p.r_2, p.id) = (r, id) & \rightarrow true\\ |\_ & \rightarrow false\end{array}$$

Pour la programmation par filtrage voir annexe A, page 208.

5 Remarques de conclusion

Dans ce chapitre, nous avons présenté un exposé unifié des problèmes de synchronisation, qui réunit les problèmes de synchronisation globaux et les problèmes de synchronisation locaux, ainsi que leur généralisation à la synchronisation à distance ϱ. Nous avons présenté un protocole auto-stabilisant générique, résolvant tous ces problèmes, tout en assurant une bonne concurrence entre les processus.

Nous avons montré comment le flux de vaguelettes est un bon outil pour résoudre de nombreux problèmes de synchronisation à distance ϱ. Pour cela, nous avons construit une double horloge auto-stabilisante ; l'horloge maîtresse définit le flux de vaguelettes et l'horloge esclave définit la barrière de synchronisation. Cette double horloge permet de définir la $\varrho-$ exclusion mutuelle locale, et ainsi de construire un transformateur qui permet de passer d'un démon $\varrho-$central vers un démon distribué inéquitable. Nous avons donné une analyse complète de complexité et généralisé la méthode à divers problèmes classiques de synchronisation locale à distance ϱ, comme la $\varrho-$ exclusion mutuelle de groupe et le problème des ϱ-lecteurs-écrivain.

Quatrième partie

Conclusion

Chapitre 9

Conclusion et perspectives

> L'un des problèmes essentiels de la science est que justement le progrès ne soit pas augmentation de volume par juxtaposition... mais révision perpétuelle des contenus par approfondissement et ratures. Il y a en lui plus de conscience – et ce n'est pas la même conscience.
>
> J. Cavaillès *Sur la logique et la théorie de la science* [Cav47] [1]

A la fin de la rédaction de ce mémoire, une chose est sûre : nous n'avons pas la même conscience de la notion d'unisson qu'au début de ce travail. Sans doute notre premier regard était-il très naïf. Les concepts introduits dans les différents chapitres, notamment les chapitres 2 et 7 structurent notre conscience et ont permis d'avancer. Cette conscience nouvelle, et les liens qu'elle tisse avec d'autres champs de l'algorithmique distribuée, nous amènent à penser que l'enquête que nous avons commencée avec ce travail n'est pas vraiment terminée. Ainsi, au-delà des algorithmes très simples et performants que nous proposons et justifions, l'essentiel du travail se trouve dans les concepts introduits, qui approfondissent et modifient notre compréhension de la synchronisation de phase.

L'unisson était un *déjà là* non totalement élucidé [Sal04]. Il a été introduit dans la version synchrone par Gouda et Herman [GH90]. Le premier unisson asynchrone est introduit par

[1] J. Cavaillès a été fusillé par les nazis le 17 février 1944 à Arras, il avait 40 ans. Dans la préface de [Cav47], Gaston Bachelard écrivait : "Loin des livres, dans la solitude héroïque d'une prison, Jean Cavaillès écrivit ce livre. C'est un livre de méditation...".

Couvreur, Francez et Gouda [CFG92] ; cet article présentait des faiblesses, la preuve de convergence était incomplète, et il n'y avait aucune étude de complexité. Dolev proposa dans [Dol00] un unisson asynchrone avec une preuve complète de convergence, celle-ci étant assurée par une réinitialisation globale. Les deux unissons asynchrones [GH90, CFG92] présentaient une exigence mystérieuse : l'occupation mémoire devait être supérieure à n^2 états, où n est le nombre de processus du réseau.

L'explication provient du fait que même dans un univers sans panne, il n'est pas toujours vrai qu'une synchronisation de phase localement correcte soit sans interblocage ou sans famine au sens où tous les processus exécuteront le code de l'application une infinité de fois. Nous avons montré que ces pathologies dépendent de l'initialisation et/ou de l'ordre de l'horloge de phase. Nous avons en particulier introduit une nouvelle constante topologique sur les graphes : la caractéristique cyclomatique, notée C_G, qui apparaît comme un seuil de bifurcation. Rappelons que M désigne l'écart de phase toléré entre deux processus voisins ; dans le cas où $M = 1$, si l'ordre K de l'horloge de phase est supérieur à C_G, alors la correction de l'algorithme est localement contrôlable (prendre $K > M.C_G$ si $M > 1$). Cette propriété permet d'effectuer une réinitialisation si la distribution des valeurs d'horloge n'est pas correcte. La caractéristique cyclomatique a le bon goût d'être majorée par le nombre n de processus dans le réseau. Quand l'ordre des horloges est supérieur à C_G, nous avons construit un relèvement sur $\mathbb{Z}$ des valeurs des horloges du réseau, et nous avons montré que l'intuition d'une horloge à valeur dans l'ensemble des entiers était une intuition correcte dans le cas d'une horloge bornée, si son ordre est minoré par la caractéristique cyclomatique du réseau. Cette intuition étant théorisée par le relèvement.

Si $K > M.C_G$, la correction d'une synchronisation de phase est localement contrôlable. Cela nous permet de définir un unisson tolérant aux fautes et aux changements de topologie. Une procedure de réinitialisation permet alors la convergence vers Γ_1, qui est l'ensemble des états où l'écart de phase entre deux voisins est majoré par 1. Nous avons étudié deux types de réinitialisation :

1. une *réinitialisation globale*, pour laquelle nous proposons l'algorithme $SS-RU$, dont la convergence est assurée en $O(d)$ rondes où d est le diamètre du réseau ; cet algorithme exige que chaque processus ait une identité distincte et que le démon soit faiblement équitable.
2. une *réinitialisation locale*, où aucune structure globale n'est utilisée explicitement.

Algorithmes proposés par réinitialisation locale

Sur l'arbre, nous avons donné des algorithmes d'unisson très performants :

1. En environnement asynchrone, $WU - Min$ converge en au plus d rondes.

2. En environnement synchrone, $SU - Min$ converge en au plus d pulsations.

L'ordre des horloges de ces deux protocoles peut être n'importe quel entier impair strictement plus grand que 1. Ces protocoles n'exigent aucune connaissance sur le réseau, excepté qu'il soit fini !
Sur le graphe général, nous avons proposé plusieurs protocoles d'unisson :

1. En environnement asynchrone, nous avons proposé une famille d'algorithmes à trois paramètres : $GAU\,(K, \alpha, M)$; dont l'algorithme $SSAU = GAU\,(K, \alpha, 1)$ qui converge en $O(n)$ si $K > C_G$ et $\alpha \geq T_G - 2$, où T_G est la longueur d'un plus grand trou dans le graphe. L' algorithme présenté dans [CFG92] est représenté par $GAU\,(N^2, 0, N)$ où N est un majorant du nombre de processus n, cet algorithme converge en $O(Nd)$. $SSAU = GAU\,(K, \alpha, 1)$ est donc sensiblement meilleur que celui de [CFG92].
2. En environnement synchrone, nous avons proposé SS_MinSU, qui converge en au plus $2d$ pulsations.

On voit qu'il est urgent, pour compléter ce tableau, de proposer un unisson asynchrone qui converge en $O\,(d)$, et cela par corrections locales sur un graphe anonyme quelconque. Une autre question intéressante, en particulier pour les réseaux dynamiques, est de chercher un unisson qui fasse de l'*endiguement de fautes* [GGHP96]. C'est-à-dire que la correction d'une seule faute locale n'entraîne pas une correction sur l'ensemble du réseau.

Nous avons donné une application intéressante de l'unisson, la *synchronisation locale*, et toutes ses variantes du type *allocation locale de ressources*. Nous avons généralisé la notion de *vague* en introduisant les notions de *vaguelettes* et de *vagues fortes*. Nous avons montré comment voir l'unisson comme un *courant de vagues*, *vaguelettes* et *vagues fortes*. Nous avons utilisé les vaguelettes pour donner un algorithme général permettant, à la demande, une *synchronisation locale*, une *synchronisation à distance* ρ, une *synchronisation globale*. C'est une solution unifiée des problèmes de synchronisation que nous proposons, du local au global. D'autres applications de l'unisson, comme courant de vagues, sont en cours de développement, notamment sur la détection de terminaison et la stabilisation instantanée. Une faiblesse de cette partie est le modèle. Il serait très intéressant de posséder un unisson performant dans le modèle à passage de messages ; ce qui permettrait, entre autre, de construire un transformateur passant directement des *modèles à état* ou *à réécriture à distance* ρ au modèle *à passage de messages*, tout en assurant une bonne concurrence.

Pour toutes ces raisons, et nous trouvant à la fin de ce présent travail, nous avons plus le sentiment d'être au milieu du gué que sur l'autre rive. Il reste beaucoup d'investigations à faire.
Amiens, le 10 juillet 2007.
Christian Boulinier

Cinquième partie

Annexes

Annexe A

Protocole d'unisson avec correction globale SS_RU

1 Réinitialisation globale

1.1 Le principe

L'idée est de construire un arbre en largeur d'abord enraciné en un noeud du réseau. Une fois cet arbre construit, un protocole utilise cet arbre pour réinitialiser le réseau en mettant la valeur 0 dans le registre de l'horloge de chaque processus. Une fois ce travail terminé, la racine lance un ordre de redémarrage de l'unisson.
Comme nous voulons définir un unisson qui soit tolérant aux changements de topologie et aux fautes transitoires, les protocoles utilisés doivent aussi être tolérants aux fautes transitoires et aux changements de topologie. Lors d'un changement de topologie, il se peut que la racine disparaisse, il est alors nécessaire de détecter la disparition de la racine et de déterminer une nouvelle racine pour construire un nouvel arbre couvrant en largeur d'abord. Nous sommes donc amenés à définir et composer équitablement trois protocoles auto-stabilisants :

1. un protocole auto-stabilisant, tolérant aux changements de topologie, déterminant une racine (élection d'un chef) ;
2. un protocole auto-stabilisant construisant un arbre couvrant en largeur d'abord, enraciné en la racine déterminée par le protocole précédent.
3. Un protocole d'unisson, utilisant l'arbre construit par le protocole précédent pour se réinitialiser en cas de faute.

Ces trois protocoles définis, il reste à les composer équitablement (voir définition page 40), c'est-à-dire à les faire s'exécuter en parallèle : une fois que le premier sera stabilisé, le second pourra se stabiliser, et une fois qu'il le sera, le troisième pourra se stabiliser et exécuter la réinitialisation, si il y a lieu. Afin que cette composition soit effectivement équitable et assure l'auto-stabilisation de la composition, nous utiliserons un démon faiblement équitable.

1.2 Recherche d'un bon candidat pour les deux premiers protocoles

Chaque processus p du réseau possède une identité unique $p.Id$, distincte de l'identité des autres processus. La détermination de la racine est basée sur la détermination du nœud ayant la plus petite identité. Dans un réseau dynamique, et à fortiori en auto-stabilisation, il se peut qu'il y ait des racines fantômes : si, par exemple, le processus racine disparaît, alors ce processus disparu devient une racine fantôme, puisque son identificateur (Id), qui était le plus petit du réseau, le reste pour les processuss qui ne savent pas qu'il a disparu. Dans un réseau non dynamique, le même effet peut provenir d'une faute transitoire qui introduit une fausse identité n'existant pas dans le réseau.
Dans ce qui suit, D est un majorant du diamètre. Pour faire disparaître les racines fantômes, on considère une deuxième variable, qui représente la distance estimée de la racine au processus. Si cette distance est supérieure à D, c'est que la racine est une racine fantôme, alors le processus l'élimine. Par ce moyen, les racines fantômes ont disparu en $O(D)$ rondes. La racine est ensuite déterminée par chacun des processus en au plus d rondes [APSV94, AKM$^+$93]. Pour une formulation avec les $r-$opérateurs, voir page 4.4.
Un grave inconvénient de cette approche est la majoration du diamètre, elle peut être trés élevée et donc le temps de stabilisation de cette méthode peut être trés mauvais en pratique [CRKGLA89, Awe90].
[AG94] propose une réinitialisation distribué auto-stabilisant, basé sur la construction d'un arbre couvrant, il requiert la connaissance d'une majoration N de la taille du réseau. Il converge en $O(N^2)$ rondes, selon [AK93].
[AKM$^+$93] propose un algorithme auto-stabilisant de recherche d'une bonne racine et de construction de l'arbre couvrant silencieux. C'est une version auto-stabilisante de l'algorithme de Bellman-Ford [Bel58], sa complexité en temps est dans $O(d)$, donc asymptotiquement optimale, et sa complexité en espace est dans $O(\ln(d))$, ce qui est asymptotiquement optimal d'après [DGS96]. Nous utilisrons cette dernière solution pour la suite ;

2 Convergence vers Γ_1 par réinitialisation globale

2.1 Programmation par filtrage

L'action de notre algorithme se définit par cas, en fonction des statuts du père et des fils. Une manière élégante de définir par cas une action est d'utiliser le filtrage de forme (pattern matching) avec gardes.
Si une forme (pattern) est reconnue (filtrée), alors l'action associée est exécutée. Si plusieurs formes sont reconnues, alors seulement la première de la liste de filtrage est exécutée. Si aucune forme n'est filtrée, alors aucune action n'est exécutée. Voir le listing Syntaxe 21
On peut ajouter à chaque forme une garde (guard, voir Syntaxe 22), qui correspond à l'évaluation d'une expression conditionnelle qui est évaluée juste après que la forme soit reconnue. Si l'évaluation retourne *vrai*, alors l'action associée est effectuée, sinon le filtrage

Syntaxe 21 Programmation par cas : le filtrage

Action :
Match *var* with
|*pattern*$_1$ $\longrightarrow$ *action*$_1$
|*pattern*$_2$ $\longrightarrow$ *action*$_2$
...
|*pattern*$_k$ $\longrightarrow$ *action*$_k$

continue sur la forme suivante. On peut, bien sûr, combiner le filtrage simple et le filtrage avec gardes. L'action elle-même peut se décomposer par un filtrage, comme ce sera le cas pour notre algorithme.

Syntaxe 22 Programmation par cas : le filtrage avec gardes

Action :
Match *var* with
|*pattern*$_1$ when *condition*$_1$ $\longrightarrow$ *action*$_1$
|*pattern*$_2$ when *condition*$_2$ $\longrightarrow$ *action*$_2$
...
|*pattern*$_k$ when *condition*$_2$ $\longrightarrow$ *action*$_k$

2.2 Le principe du protocole SS_RU

Dans cette section le réseau est avec identités. Nous considérons que les deux sous-couches de détermination d'une racine et de construction d'un arbre couvrant en largeur d'abord sont définies. Ces sous-couches sont auto-stabilisantes. Nous utiliserons l'algorithme défini dans [AKM$^+$93]pour réaliser ces sous-couches. Comme nous l'avons vu page 1.2, cet algorithme est optimal tant du point de vue de la complexité en temps de stabilisation, que du point de vue de la complexité en espace. Il combine les deux premières couches, il n'est pas basé sur la connaissance du nombre de processus, ni sur la connaissance du diamètre du réseau. Pour borner la mémoire, il utilise la connaissance d'une majoration D du diamètre. D est une constante qui n'est pas corruptible. D peut être très gand, mais cela n'affecte pas la complexité en temps de l'algorithme qui est dans $O(d)$). La complexité en espace est, elle, affectée logarithmiquement, elle est dans $O(ln(D)$.

Au dessus de l'algorithme [AKM$^+$93], nous allons définir un algorithme nommé RU qui contrôle si l'unisson est correct, et, le cas échéant, lance une réinitialisation par la racine de l'arbre sous-jacent. Cet algorithme assure l'exécution de l'unisson quand celui-ci est correct. Cet algorithme doit être auto-stabilisant, il suppose connus la racine courante du réseau et l'arbre couvrant défini par l'exécution de [AKM$^+$93]. Une fois cet algorithme défini, il restera à le composer équitablement avec l'algorithme defini dans [AKM$^+$93] grâce à un démon faiblement équitable. Cette composition définit l'algorithme d'unisson SS_RU .

2.3 Le principe du protocole *RU*

Le contrôle de la correction de l'unisson est facile si on prend une horloge d'ordre $K > MC_G$, où C_G est la caractéristique cyclomatique du graphe G définissant le réseau. Dans la suite nous supposerons toujours que cette inégalité est vérifiée. Alors, pour conrôler, il suffit que chaque nœud vérifie qu'il est correct par rapport à ses voisins. Chaque processus p maintient un registre supplémentaire, le registre *color* qui contient l'une des quatres couleurs $\{E, C, F_C, R\}$. L'algorithme a accés au registre *parent* qui pointe vers son père. Dans le cas d'une incorrection détectée, la réinitialisation se fait en 4 vagues :

1. Si un processus n'est pas correct, il prend la couleur E (Erreur). La couleur E se propage aux fils et au père, et ainsi jusqu'aux feuilles et jusqu'à la racine.
2. Une fois que la racine a tous ses voisins à la couleur E, la racine prend la couleur C (Correction). La correction se propage de la racine vers les feuilles.
3. Une fois qu'une feuille a la couleur C et que son père est à C, la feuille prend la couleur F_C (Fin de Correction) et met le registre de son horloge à 0. La fin de correction descend vers la racine : un processus prend la couleur F_C et met son registre d'unison à 0 si et seulement si ses fils sont à la couleur F_C et son père a la couleur C.
4. Une fois que la racine, qui à la couleur C, est entourée de fils à la couleur F_C, elle prend la couleur R (Repos) ; alors l'unisson peut redémarrer. Un processus prend la couleur R quand son père est à la couleur R, qu'il est à la couleur F_C et que ses fils aussi.

le protocole est constitué de trois types d'actions (i désigne un entier) :

1. les actions de mise en erreur, notées (EI) (mise en ereur initiale) et (Ei) (mise en erreur) ;
2. les actions de réinitialisation, notées (Ai) ;
3. les actions de gestion normale de l'unisson, (AN) et (AE).

2.3.1 Le cas d'un processus qui n'est pas la racine

Cet algorithme comporte des incompatibilités de coloration, la table A.1 présente ces incompatibilités et les règles de mise à la couleur E associées (règles (EI)). Les règles $(E1)$, $(E2)$ et $(E3)$ sont les règles de propagation de la couleur E.
Chaque fois qu'il y aura une incompatibilité constatée par un processus, ce processus prendra la couleur E par une action notée (EI). Cette couleur est invasive, sauf par rapport à la couleur C lors de la remontée dans l'arbre, action $(A1)$. Les actions de propagation de la couleur E sont les actions $(E1)$, $(E2)$, $(E3)$, et $(E4)$. La couleur C ne peut être détruite qu'à la redescente de la couleur F_C, c'est l'action $(A2)$. Une fois que la couleur F_C est redescendue à la racine, la couleur R remonte aux feuilles, c'est l'action $(A3)$ qui en même temps met le registre r à 0. Nous allons voir que ces règles suffisent à auto-stabiliser notre algorithme.

$p.parent \setminus p$	E	C	F_C	R
E	oui	$(E1)$	$(E2)$	$(E3)$
C	oui	oui	oui	(EI)
F_C	oui	(EI)	oui	(EI)
R	oui	(EI)	oui	oui

TAB. A.1 – Tableau des compatibilités de couleurs d'un processus p et des actions de correction associées aux incompatibilités.

La gestion normale de l'unisson est assurée par les actions (AN) (action normale) et (AE) (action de mise en erreur). Les prédicats associés sont :

$Correct_p(q) \equiv d(p.r, q.r) \leq M$;
$AllCorrect_p \equiv \forall q \in \mathcal{N}_p : Correct_p(q)$;
$NormalStep_p \equiv \forall q \in \mathcal{N}_p : p.r \leq_l q.r$;
$ResetInit_p \equiv \neg AllCorrect_p$;

Pour un processus qui n'est pas la racine, la gestion de l'unisson et des couleurs est formalisée par le tableau : action 23.

Action 23 Les trois types d'actions pour $p \neq root$ et gestion des couleurs

If $p.parent \neq \perp$ then :
Match p.color with :

$\|E$	$\longrightarrow$	Match p.parent.color with :			
		$\|C$ when $\forall q \in child_p, q.color = E$	$\longrightarrow$	$p.color := C$	$(A1)$
$\|C$	$\longrightarrow$	Match p.parent.color with :			
		$\|C$ when $\forall q \in child_p, q.color = F\_C$	$\longrightarrow$	$p.color := F\_C$	$(A2)$
		$\|E$	$\longrightarrow$	$p.color := E$	$(E1)$
		$\|F\_C \| R\_$	$\longrightarrow$	$p.color := E$	(EI)
$\|F\_C$	$\longrightarrow$	Match p.parent.color with :			
		$\|\_$ when $\neg(\forall q \in child_p, q.color = F\_C)$	$\longrightarrow$	$p.color := E$	
		$\|E$	$\longrightarrow$	$p.color := E$	$(E2)$
		$\|R$	$\longrightarrow$	$p.color := R; p.r := 0$	$(A3)$
$\|R$	$\longrightarrow$	Match p.parent.color with :			
		$\|R$ when $\neg(\forall q \in child_p, q.color \in \{F\_C, R\})$	$\longrightarrow$	$p.color := E$	(EI)
		$\|E$	$\longrightarrow$	$p.color := E$	$(E3)$
		$\|F\_C \| C$	$\longrightarrow$	$p.color := E$	(EI)
		$\|R$ when $NormalStep \wedge (\forall q \in \mathcal{N}_p, q.color = R$	$\longrightarrow$	$p.r := \varphi(p.r)$	(AN)
		$\|R$ when $ResetInit$	$\longrightarrow$	$p.color := E$	(AE)

Le protocole complet est présenté par l'algorithme 24.

2.3.2 Le cas de la racine

La racine n'a pas besoin du registre $root.parent$, ni de la couleur F_C. Les fils de la racine sont ses voisins. La racine passe à la couleur C et initialise son registre r à 0 quand tous

Algorithme 24 Unisson enraciné pour un processus $p \neq root$

Constante :

$\mathcal{N}_p$: l'ensemble des voisins du processus p ;

$child_p$: l'ensemble des enfants du processus p ;

Variables :

$p.color \in \{E, C, F\_C, R\}$;

$p.r \in \mathbb{Z}_K$;

$p.parent \in \mathcal{N}_p \cup \{\perp\}$;

Fonctions booléennes :

$Correct_p(q) \equiv d(p.r, q.r) \leq M$;
$AllCorrect_p \equiv \forall q \in \mathcal{N}_p : Correct_p(q)$;
$NormalStep_p \equiv \forall q \in \mathcal{N}_p : p.r \leq_l q.r$;
$ResetInit_p \equiv \neg AllCorrect_p$;

Actions :

If $p.parent \neq \perp$ then :
Match p.color with :

$\|E$	$\longrightarrow$	Match p.parent.color with :			
		$\|C$ when $\forall q \in child_p, q.color = E$	$\longrightarrow$	$p.color := C$	$(A1)$
$\|C$	$\longrightarrow$	Match p.parent.color with :			
		$\|C$ when $\forall q \in child_p, q.color = F\_C$	$\longrightarrow$	$p.color := F\_C$	$(A2)$
		$\|E\|F\_C\|R\_$	$\longrightarrow$	$p.color := E$	$(E1)$ (EI)
$\|F\_C$	$\longrightarrow$	Match p.parent.color with :			
		$\|\_$ when $\neg(\forall q \in child_p, q.color = F\_C)$	$\longrightarrow$	$p.color := E$	(EI)
		$\|R$	$\longrightarrow$	$p.color := R; p.r := 0$	$(A3)$
		$\|E$	$\longrightarrow$	$p.color := E$	$(E2)$
$\|R$	$\longrightarrow$	Match p.parent.color with :			
		$\|R$ when $\neg(\forall q \in child_p, q.color \in \{F\_C, R\})$	$\longrightarrow$	$p.color := E$	
		$\|E\|F\_C\|C$	$\longrightarrow$	$p.color := E$	$(E3)$ (EI)
		$\|R$ when $NormalStep \wedge (\forall q \in \mathcal{N}_p, q.color = R$	$\longrightarrow$	$<<$ *Action application* $>>$;	
				$p.r := \varphi(p.r)$	(AN)
		$\|R$ when $ResetInit$	$\longrightarrow$	$p.color := E$	(AE)

ses voisins sont à la couleur E, action $(A1)$. La racine passe à la couleur R quand tous ses voisins sont à la couleur F_C, action $(A2)$. Le tableau A.2 donne les incompatibilités de la racine.

Racine \ Fils	E	C	F_C	R
E	oui	oui	oui	oui
C	oui	oui	oui	(EI)
R	$(E4)$	(EI)	oui	oui

TAB. A.2 – Tableau des compatibilités de couleurs avec la racine et des actions de correction

L'algorithme 25 pour la racine et l'algorithme 24 pour les autres processus décrivent complètement les actions de la couche supérieure. Bien entendu le bon fonctionnement de notre algorithme enraciné est conditionné par la correction de l'arbre en largeur d'abord construit par l'algorithme défini dans [AKM+93] et implémenté sur la couche inférieure.

Algorithme 25 Unisson enraciné pour $p = root$

Constante :

$\mathcal{N}_{root}$: ensemble des voisins de $root$;

Variables :

$root.color \in \{E, C, R\}$;

$root.r \in \mathbb{Z}_K$;

Fonctions booléennes :

$Correct_{root}(q) \equiv d(root.r, q.r) \leq M$;
$AllCorrect_{root} \equiv \forall q \in \mathcal{N}_{root} : Correct_{root}(q)$;
$NormalStep_{root} \equiv \forall q \in \mathcal{N}_{root} : root.r \leq_l q.r$;
$ResetInit_{root} \equiv \neg AllCorrect_{root}$;

Actions :

Match root.color with :

$\|E$	when $\forall q \in \mathcal{N}_{root}, q.color = E$	$\longrightarrow$	$root.color := C$	$(A1)$
$\|C$	when $\forall q \in \mathcal{N}_{root}, q.color = F\_C$	$\longrightarrow$	$root.r := 0; root.color := R$	$(A2)$
$\|C$	when $\exists q \in \mathcal{N}_{root}, q.color = R$	$\longrightarrow$	$root.color := E$	(EI)
$\|R$	when $\exists q \in \mathcal{N}_{root}, q.color \in \{C, E\}$	$\longrightarrow$	$root.color := E$	(EI) $(E4)$
$\|R$	when $NormalStep \wedge (\forall q \in \mathcal{N}_r oot, q.color = R$	$\longrightarrow$	$<< Action\ application >>$; $root.r := \varphi(root.r)$	(AN)
$\|R$	when $ResetInit$	$\longrightarrow$	$root.color := E$	(AE)

2.4 La correction du protocole *RU*

Nous prouvons la convergence par la méthode des attracteurs successifs [GM91].
La preuve est relativement facile sous un démon faiblement équitable, ce qui est suffisant de notre point de vue, puisque nous ferons ensuite une composition équitable qui sera garantie par un démon faiblement équitable. L'intérêt d'une preuve sous un démon inéquitable est qu'elle garantit que les rondes sont uniformément bornées en nombre de transitions. Pour

qu'une preuve sous démon faiblement équitable conduise à une preuve sous un démon inéquitable, il suffit de montrer que si un processus ne fait plus rien alors le système conduit à un interblocage, alors le démon inéquitable choisira nécessairement ce processus. Cela signifie que dans ces circonstances, un démon inéquitable et un démon faiblement équitable, sont équivalents. Pour prouver la correction, nous commençons par prouver que lors d'une exécution du protocole, il n'y a qu'un nombre fini de passages en erreur. Nous présentons dans les figures A.1 et A.2 l'automate des transitions possibles dans le cas général pour la racine et les autres nœuds du réseau.

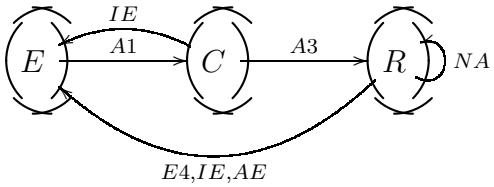


FIG. A.1 – Automate des changements de couleur, $p = root$

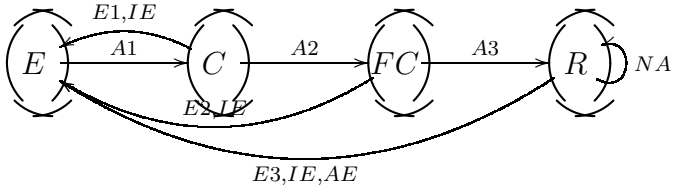


FIG. A.2 – Automate des changements de couleur, $p \neq root$

2.4.1 Disparition des actions de passage en erreur

L'objet de cette première partie est de montrer que quelle que soit l'exécution, le nombre de passages en erreur est fini. Pour cela nous devons introduire quelques concepts permettant l'analyse de la propagation des erreurs.
On appelle passage en erreur initial les actions IE et AE. Les actions $A1, A2, A3, A4$ et AN n'entraînent pas d'incompatibilité, les passages à la couleur E non initiaux non plus ; donc il n'y a qu'un nombre fini de passages en erreur initiaux (actions IE ou AE), un au plus par processus, dont c'est la première action.
Les passages en erreur autres qu'initiaux sont tous provoqués par un voisin avec le statut E (actions $E1, E2, E3, E4$). Nous considérons maintenant le Dag des passages en erreur, défini de la manière suivante :

Définition 59 Soit $\gamma_0 \rightarrow \gamma_1 \rightarrow ...$ une exécution.

1. Si le processus p a la couleur E en γ_0, alors on dit que $(p, 0)$ est un passage en erreur.

2. Si lors de la transition $\gamma_i \to \gamma_{i+1}$ le processus p prend la couleur E par l'action IE, alors $(p, i+1)$ est un passage en erreur initial.
3. Si lors de la transition $\gamma_i \to \gamma_{i+1}$ le processus p prend la couleur E par une action $E1, E2, E3$ ou $E4$, alors $(p, i+1)$ est un passage en erreur.

Remarquons que si un processus p prend la couleur E par une action $E1, E2, E3$ ou $E4$ lors de la transition $\gamma_i \to \gamma_{i+1}$, alors il existe un voisin q de p qui a le statut E en γ_i, donc il existe un passage en erreur (q, j) de q qui lui donne la couleur$E$, plus précisément :

$$\forall t \in [j, i-1]\,,\ q.color = E$$

Définition 60 Dag des passages en erreur.
On considère, sur l'ensemble des passages en erreur, la relation binaire $(q, j) \rightsquigarrow (p, i)$ définie par :

$$(q, j) \rightsquigarrow (p, i) \Leftrightarrow \begin{cases} q = p.parent \text{ ou } q \in child_p \\ j < i \\ \forall t \in [j, i-1]\,,\ q.color = E \end{cases}$$

$(q, j) \rightsquigarrow (p, i)$ implique $j < i$, donc le graphe défini par cette relation est sans cycle, c'est donc un Dag.

Lemme 123 Le Dag des passages en erreur ne comporte aucun béguaiement

1. Rappelons qu'un béguaiement est une séquence $(p, i_0) \rightsquigarrow (q, i_1) \rightsquigarrow (p, i_2)$. Une telle séquence est impossible ici, puisqu'il faudrait que p perde la couleur E à la date i_1, ce qui est impossible, puisqu'à la date $i_1 - 1$ le processus q n'est pas correct avec p.

Lemme 124 La longueur d'un chemin le long du Dag des passages en erreur est au plus d.

Preuve : Le Dag des passages en erreur est construit sur un arbre ; comme il n'y a pas de béguaiement, il ne peut y avoir de retour. La projection d'un chemin du Dag sur l'arbre est un chemin simple, donc de longueur au plus la hauteur de l'arbre, qui est majorée par d.

Proposition 125 Lors de n'importe quelle exécution du protocole, le nombre de passages en erreur est fini.

□

2.4.2 Absence d'interblocage

On prouve l'absence d'interblocage par étapes successives. On raisonne toujours en fonction de la topologie définie par l'arbre en largeur d'abor défini au dessous. La preuve se fait en distinguant trois cas :
Il y a au moins un processus qui a la couleur E dans le réseau.
Prenons un processus p ayant la couleur E et qui soit le plus proche de la racine :

1. Si $p = root$ alors deux cas sont possibles :
   (a) un fils n'a pas la couleur E, alors il peut faire l'action $E1$ ou $E2$ ou $E3$ suivant sa couleur.
   (b) Tous les fils sont à la couleur E, alors $root$ peut faire l'action $A1$
2. Si $p \neq root$ alors deux cas sont possibles :
   (a) Le père n'a pas la couleur C (il ne peut pas avoir la couleur E), alors le père peut effectuer l'action EI ;
   (b) le père a la couleur C, alors deux cas sont possibles :
      i. Tous les fils de p ont la couleur E, alors p peut faire l'action $A1$
      ii. Il existe un fils qui n'a pas la couleur E, alors ce fils peut faire l'action $E1$ ou $E2$ ou $E3$ suivant sa couleur.

On en déduit :

Proposition 126 Si un processus a la couleur E, alors le réseau n'est pas interbloqué.

Aucun processus n'a la couleur E, au moins un a la couleur C
Prenons un processus p ayant la couleur C et qui soit l'un des plus éloigné de la racine :

1. Si $p = root$ alors deux cas sont possibles :
   (a) Tous les fils ont la couleur FC, alors p peut faire l'action $A3$
   (b) Il existe un fils qui a la couleur R (les couleurs E et C sont impossibles), alors p peut faire l'action EI
2. Si $p \neq root$ alors deux cas sont possibles :
   (a) Le père de p a la couleur R ouFC, alors p peut effectuer la règle EI
   (b) Le père de p a la couleur C, alors deux cas sont possibles :
      i. Tous les fils de p ont la couleur FC, alors p peut effectuer la règle $A2$
      ii. Il existe un fils de p qui a la couleur R, alors p peut effectuer la règle EI

On en déduit :

Proposition 127 Si un processus a la couleur C, alors le réseau n'est pas interbloqué

Aucun processus n'a la couleur E ou C, et au moins un a la couleur FC
Prenons un des processus les plus proches de la racine ayant la couleur FC. Ce processus p est différent de $root$. Son père est nécessairement à la couleur R, donc p peut effectuer l'action $A3$.

On en déduit :

Proposition 128 Si un processus a la couleur FC, alors le réseau n'est pas interbloqué.

d'où :

Théorème 129 Si $K > MC_G$, alors le protocole est sans interblocage .

Preuve : le seul cas à examiner est celui où tous les processus sont à la couleur R :

1. Si le réseau est dans un état incorrect, alors cela est localement vérifiable, il y a donc un processus qui peut faire l'action AE.
2. Si le réseau est dans un état correct, il est en configuration d'unisson ; dans ce cas, le processus le plus en retard selon la relation de précédence peut effectuer l'action AE.

□

2.4.3 Absence de famine et auto-stabilisation

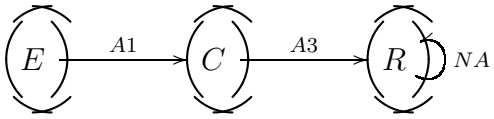


FIG. A.3 – Automate des changements de couleur sans passage en erreur, $p = root$

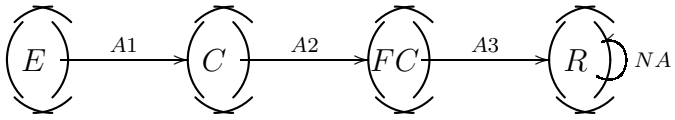


FIG. A.4 – Automate des changements de couleur sans passage en erreur, $p \neq root$

Considérons maintenant une exécution maximale e. Plaçons-nous maintenant dans un suffixe e' de cette exécution, qui ne contient pas de passage en erreur (cela est possible d'aprés la proposition 125). Nous présentons dans les figures A.3 et A.4 l'automate des transitions possibles dans le cas où il n'y a plus de passage en erreur. On montre dans cette partie que si un processus ne fait plus rien, alors le système est conduit à un interblocage. Le réseau étant fini, il sufft de prouver que les voisins du point de vue de l'arbre, c'est-à-dire le père et les fils, ne feront qu'un nombre fini d'actions. Supposons que le voisin q (pére ou fils) d'un processus p qui n'exécute plus aucune action, fasse une infinité de transitions ; d'après les automates de transition présentés par les figures A.3 et A.4 , ultimement le processus q exécutera une infinité de fois la transition NA, ce qui n'est possible qu'au plus deux fois, puisque le processus p ne fait aucune action. Donc le père (s'il existe) et les fils (s'ils existent) ne feront qu'un nombre fini d'actions. Ce qui conduit à un interblocge.
Nous venons de montrer que la famine implique l'interblocage, donc par contraposition, puisqu'il n'y a pas d'interblocage, il n'y a pas de famine.

Proposition 130 Le protocole est sans interblocage et sans famine.

Reprenons le suffixe e' : chaque processus fait une infinité de transitions, seules les trois premières au plus ne sont pas l'action NA, on en déduit qu'il existe un suffixe e" de e' où seules des actions NA sont effectuées. Le système est donc dans Γ_M, Il est stabilisé. De plus l'exécution de l'action de l'application se fait toujours selon la spécification de synchronisation. On en déduit :

Proposition 131 Le protocole $SS - RU$ est un unisson.

2.4.4 Temps de convergence

Notre protocole RU est sans famine, les rondes sont donc finies (on suppose ici l'arbre sous-jacent construit). Reprenons les définitions de la sous-section 2.4.1. On appelle passage en erreur initial les actions IE et AE. Les actions $A1, A2, A3, A4$ et AN n'entraînent pas d'incompatibilité, les passages à la couleur E non initiaux non plus (actions $E1, E2, E3$ et $E4$) ; donc il n'y a qu'un nombre fini de passages en erreur initiaux : un au plus par processus, dont c'est la première action. En particulier, après une faute transitoire ou un changement de topologie, au bout d'une ronde il n'y a plus de passage en erreur initial. On en déduit le lemme :

Lemme 132
A la fin de la première ronde, il n'y a plus d'action de type EI ou AE.

Reprenons le Dag des passages en erreurs (voir sous-section 2.4.1). La longueur d'un chemin le long du Dag des passages en erreur est au plus d.

Proposition 133 Après au plus $d+1$ rondes, plus aucun processus ne prend la couleur E.

Preuve : Il faut au plus une ronde pour que les passages en erreur s'effectuent. De plus, si $(q, j) \rightsquigarrow (p, i)$, alors p peut faire l'action de mise en erreur à tout instant entre $j + 1$ et i, donc il se déroule au plus une ronde entre les temps $j + 1$ et i. Donc le Dag est construit en au plus $d + 1$ rondes.

□

Lemme 134 Après $d + 1$ rondes, si un processus p a la couleur E, alors tous ses descendants sont à la couleur E.

Preuve : Supposons que le lemme est faux : soit q un descendant le plus proche de p qui n'est pas à la couleur E, alors son père (qui a la couleur E, et qui existe car q n'est pas la racine) est bloqué tant que q ne prend pas la couleur E ; donc, par l'absence de famine, q prendra la couleur E, ce qui est absurde puisque par hypothèse plus aucun processus ne prend la couleur E.

□

Les trois lemmes suivants sont faciles :

Lemme 135 Après $2d + 2$ rondes, plus aucun processus n'a la couleur E, et si un processus a la couleur C, alors tous ses ascendants ont la couleur C.

Lemme 136 Après $3d+3$ rondes , plus aucun processus n'a la couleur C, et si processus a la couleur F_C, alors tous ses descendants ont la couleur F_C.

Lemme 137 Après $4d+4$ rondes, tous les processuss sont a la couleur R et l'algorithme est stabilisé.

2.5 Conclusion

Nous venons de définir un algorithme d'unisson nommé SS_RU. Il est la composition équitable de l'algorithme de construction d'un arbre couvrant en largeur d'abord défini dans [AKM+93], appelé ici $OptSSBFS$, et de l'algorithme RU. Cette composition est équitable si le démon est faiblement équitable. La complexité en temps de $OptSSBFS$ est dans $O(d)$ et sa complexité en espace est dans $O(\ln N)$. Où N est un majorant de n connu du système. La complexité en temps de la convergence vers Γ_1 de RU est dans $O(d)$ et la complexité en espace est dans $O(\ln N)$. On en déduit que la complexité en temps de convergence vers Γ_1 de SS_RU est dans $O(d)$ et sa complexité en espace est dans $O(\ln N)$; la complexité en temps est comptée en nombre de rondes.

Théorème 138 Sous un démon faiblement équitable, le protocole SS_RU est un protocole d'unisson. Il converge vers Γ_1 en $O(d)$ rondes. Sa complexité en espace est dans $O(lnN)$, où N est un majorant de la taille du réseau.

Annexe B

Chercher la longueur du plus grand trou est NP-difficile

1 Introduction

L'idée de ce chapitre est née à la terrasse d'un café lors d'une conversation avec Alain Cournier.

1.1 Les trous

Un trou est un cycle sans corde. En théorie des graphes parfaits, la définition spécifie que les trous ont une longueur supérieure ou égale à 4. Dans ce mémoire, nous n'exigeons pas cette restriction, un triangle est un trou. Cela provient du fait que dans notre étude, les trous jouent un rôle d'objets minimaux au sens de la relation d'ordre définie par la décomposition des cycles. On pourrait découvrir ainsi une structure noethérienne sur l'ensemble des cycles et donc une structure adaptée à des raisonnements par induction. Si on regarde bien pourquoi les trous apparaissent dans notre étude, c'est précisément à cause de cette structure. Dans le chapitre 3 ainsi que dans [BPV04], nous avons montré le rôle important que jouent les trous dans la dynamique des réinitialisations locales de la famille de protocoles $GAU(K, \alpha, M)$, voir algorithme 7. Les cordes jouent le rôle de court-circuits. La question de la longueur d'un plus grand trou du graphe G, longueur notée T_G, est posée par le problème de la convergence $\Gamma \triangleright \Gamma_M$ pour les algorithmes $GAU(K, \alpha, M)$. Si le graphe est acyclique, nous avons posé par définition $T_G = 2$ (Voir définition 2.3.3).
Au chapitre 3, nous établissons que $\Gamma \triangleright \Gamma_M$ est garanti pour $GAU(K, \alpha, M)$ si et seulement si :

$$(\alpha = 0 \text{ et } M \geq T_G - 2) \text{ ou } (\alpha \geq T_G - 2)$$

La question de la longueur du plus grand trou dans un graphe apparait donc comme un problème lié à la complexité en espace de l'algorithmes 7 page 96. Pour autant que nous le sachions, c'est la première fois qu'une telle notion est utilisée dans le champ de l'algorithmique distribuée.

Deux conjectures sur les graphes parfaits de Claude Berge [Ber61] stimulèrent la recherche sur les trous. La conjecture faible fut résolue par Lovász [Lov72] en 1972 et la conjecture forte des graphes parfaits fut résolue par Chudnovski, Robertson, Seymour etThomas [CRST05]. La notion de trou fut trés étudiée dans le domaine des graphes parfaits, une question étant de savoir si un graphe G et son complément contiennent ou pas un trou d'ordre impair. Des résultats algorithmiques sont connus sur les trous, mais curieusement rien sur le calcul de la longueur du plus grand trou dans un graphe.

1.2 Un algorithme naïf

Un algorithme trivial par exploration exhaustive, répond à la question suivante :

Un graphe donné contient-il un trou de longeur k passant par un sommet donné ?

Cet algorithme est de complexité $O\left(n^{k-1}\right)$.
A partir de ce premier algorithme on obtient facilement un algorithme répondant à la question suivant :

Un graphe donné contient-il un trou de longueur k ?

Sa complexité en temps est $O\left(n^{k}\right)$, compexité polynomiale pour k fixé, mais pas polynômiale pour l'entrée (G, k).
On imagine clairement, à partir de l'algorithme précédent, comment répondre à la question :

Quelle est la longueur du plus grand trou d'un graphe donné

Malheureuseent la complexité d'un tel algorithme, sur l'entrée G est en $O(n^n)$.
Le seul résultat connu sur la complexité de propriétés sur les trous est , à ma connaissance le suivant :

Reconnaitre si un graphe contient un trou impair passant par un sommet donné

Ce problème est un problème NP-complet[Bie91].
L'objet de ce chapitre est de montrer que dans le cas général, la question de calculer la longueur du plus grand trou dans un graphe est un problème NP-difficile. Nous montrerons que ce problème reste NP-difficile dans certaines classes de graphes, notament les graphes bipartis, les graphes planaires et les grilles.

2 La transformation $\mathcal{T}$

On considère la transformation $\mathcal{T}$ sur la classe des graphes finis suivante :

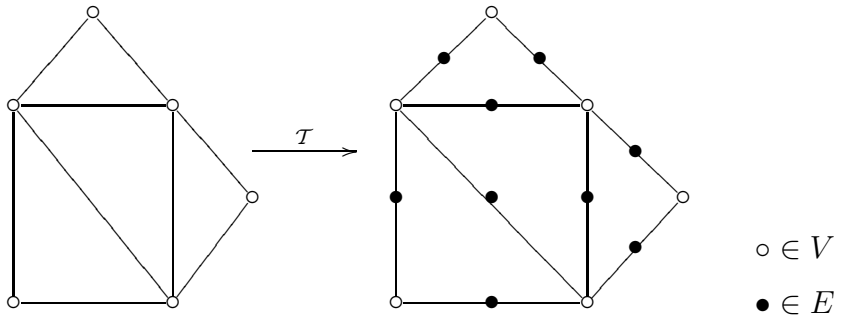


FIG. B.1 – La transformation $\mathcal{T}$

Définition 61 Soit $G = (V, E)$ un graphe. On note $\mathcal{T}(G)$ le graphe $\tilde{G} = \left(\tilde{V}, \tilde{E}\right)$ défini par :

$$\begin{aligned} \tilde{V} &= V \cup E \\ \tilde{E} &= \{\{v, e\}, v \in V, e \in E, v \in e\} \end{aligned}$$

Voir la figure B.1

Il est clair par construction que :

Proposition 139 $\tilde{G}$ est bibparti. Chaque noeud de $\tilde{G}$ dans E a un degré 2. Chaque chemin de $\tilde{G}$ est composé alternativement de noeuds de V et de E.

A cause de cette alternance de deux sortes de noeuds, on obtient le corollaire :

Corollaire 1 La longueur de tout cycle de $\tilde{G}$ pair.

Si une corde rejoind deux noeuds d'un cycle de $\tilde{G}$, un des sommets est dans V et l'autre est dans E et sont degré est strictement plus grand que 2. C'est une contradiction, donc par la proposition 139 :

Corollaire 2 $\tilde{G}$ est sans corde (i.e. tout cycle de $\tilde{G}$ est sans corde). En d'autres mots, tout cycle de $\tilde{G}$ est un trou.

Soit $x_1x_2...x_{2p}x_1$ un cycle de $\tilde{G}$, si $x_1 \in V$ alors par construction $x_1x_3...x_{2p-1}$ est un cycle de G, de même si $x_1 \notin V$ alors $x_2x_4...x_{2p}$ est un cycle de G. Donc :

Proposition 140 Pour tout cycle C de $\tilde{G}$ de longueur $2p$, il exists un cycle C_1 of G tel que $\tilde{C}_1$=C, et $|C_1| = p$

De la proposition 140 il suit le corollaire :

Corollaire 1 La longeur de tout cycle (trou) dans $\tilde{G}$ est plus petit ou égal à $2|G|$.

On obtient le corollaire important :

Corollaire 2 G est hamiltonien si et seulement si la longueur du plus grand trou de $\tilde{G}$ est $2|G|$

.

3 Trouver la longueur du plus grand trou

le problème (LLH) posé au début de ce chapitre et dans [BPV04] est le suivant :

INSTANCE : Le graphe $G = (V, E)$

ACTION : Retourner la longueur d'un trou de plus grande longueur de G, ou 2 si G est sans cycle.

Le problème de décision associé (HGK) est :

INSTANCE : le graphe $G = (V, E)$ et un entier k

ACTION : retourner $vrai$ si il existe un trou dans G avec une longueur plus grande ou égale à k, retourner $faux$ sinon.

Supposons qu'un algorithme $\mathcal{A}$ résolve le problème LLH pour out graphe G. le temps de construction de $\tilde{G}$ est une fonction linaire de $|E|$. G possède un cycle hamiltonien si et seulement si $\tilde{G}$ a un cycle passant par tous les sommets de G, c'est un trou maximal de $\tilde{G}$. Donc G possède un circuit hamiltonien si et seulement si $\mathcal{A}\left(\tilde{G}\right) = 2\,|G|$ (Corollaire 2).
Le problème général de décider si un graphe contient un circuit hamiltonien est NP-complet[Kar72].
Donc calculer la longueur du plus grand trou dans un graphe est au moins aussi difficile que de décider si un graphe a un circuit hamiltonien. En d'autres termes, la longueur du plus grand trou est un problème NP-difficile. Bien sûr le problème de décision HGK est NP-complet dans le cas général, sinon il permettrait de résoudre le problème de l'existence d'un circuit hamiltonien, problème qui est NP-complet.
Plus généralement nousz énonçons la proposition suivante :

> **Proposition 141** Si la classe de graphes $\mathcal{C}$ est stable pour la transformation $\mathcal{T}$, alors si le problème de savoir si un objet de cette classe $\mathcal{C}$ possède un circuit hamiltonien est NP-complet pour cette classe alors LLH est NP-difficile et HGK NP-complet pour $\mathcal{C}$.

Le problème de l'existence d'un circuit hamiltonien est NP-complet pour les classes de graphes suivantes : Graphes bipartis [Kri75] , graphes planaires [GJT76] et grilles [IPS82]. On en déduit :

> **Corollaire 1** LLH est NP-difficile et HGK NP-complet pour les classes de graphes finis : La classe des graphes bipartis, la classe des graphes planaires et la classe des grilles.

Annexe C

Notions élémentaires de théorie des graphes et d'algébre

1 Définitions générales sur les graphes

1.0.1 Graphe non orienté et graphe orienté, degré et demi-degré

Définition 62 Soit S un ensemble fini, A une partie de $\{\{a,b\},(a,b)\in S^2 \text{ et } a\neq b\}$ alors

$$G=(S,A)$$

est un graphe non orienté simple.
S est l'ensemble des *sommets*, A l'ensemble des *arêtes*.

Définition 63 Une arête est incidente à un sommet si celui-ci est l'une des extrémités de l'arête.
Le degré d'un sommet s est le nombre d'arêtes incidentes à ce sommet, on le note $d_G(s)$. On note $\omega_G(s)$ l'ensemble des arêtes incidentes de s

Dans ce qui suit et sauf indication contraire, graphe signifiera "graphe non orienté simple".

Définition 64 Soit S un ensemble fini, A une partie de $S^2\backslash\{(a,a),\ a\in S\}$ alors

$$G=(S,A)$$

est un graphe orienté simple.
S est l'ensemble des *sommets*, A l'ensemble des *arcs*.

Définition 65 Soit $G=(S,A)$ un graphe orienté. Un arc (s,s') est incident à un sommet si celui-ci est l'une des extrémités de l'arc, alors (s,s') est incident à s vers l'extérieur et incident à s' vers l'intérieur.
On note $d_G^+(s)$ le nombre d'arcs incidents vers l'extérieur à s (sortant de s), et $d_G^-(s)$ le nombre d'arcs incidents vers l'intérieur à s (entrant dans s).

On note $\omega_G^+(s)$ l'ensemble des arcs sortants de s, et $\omega_G^-(s)$ l'ensemble des arcs entrants dans s.
On pose $\omega_G(s) = \omega_G^+(s) + \omega_G^-(s)$ l'ensemble des arcs incidents à s.

Définition 66 Soit $G = (S, A)$ un graphe. Un chemin (ou une chaîne) sur G est une suite finie de sommets

$$s_0, s_1, ..., s_p$$

telle que $p > 0$ et :

$$\forall i \in \{0, ..., p-1\}, \ \{s_i, s_{i+1}\} \in A$$

La longueur du chemin est p. Un chemin est simple (ou élémentaire) si il ne visite pas deux fois le même sommet.

Définition 67 La distance d'un sommet s à un sommet s' est la longueur d'un plus court chemin allant de s à s'. On note cette distance par $dist(s, s')$. Si il n'existe pas de chemin de s à s', alors $dist(s, s') = \infty$.

Définition 68 Soit $G = (S, A)$ un graphe. Un cycle est une chaîne $s_0, s_1, ..., s_p$ telle que $s_0, s_1, ..., s_{p-1}$ est un chemin élémentaire et $s_0 = s_p$.
Une corde du cycle $s_0, s_1, ..., s_p$ est une arête $\{s_i, s_j\}$ avec $i < j - 1$ ou $j < i - 1$

Remarquons qu'un cycle est au moins de longueur 3.

Définition 69 Soit $G = (S, A)$ un graphe, il est connexe si et seulement si pour tout couple de sommets (s, s') il existe un chemin allant de s à s'.
Alors il existe un chemin élémentaire allant de s à s'.

Définition 70 Soit $G = (S, A)$ un graphe, c'est un arbre si il est connexe et sans cycle.

Définition 71 un graphe sans cycle est une forêt.

Définition 72 Soit $G = (S, A)$ un graphe, $G' = (S', A')$ est un sous-graphe de G si et seulement :

$$\begin{aligned} S' &\subset S \ et \ A' \subset A \\ \forall \{s, s'\} &\in A', \{s, s'\} \in S' \times S' \end{aligned}$$

et on dit que G' est un sous-graphe induit de G si et seulement :

$$S' \subset S \ et \ A' = A \cap S' \times S'$$

2 Relations binaires sur un ensemble E

2.1 Définitions générales

Définition 73 Une **relation binaire $\mathcal{P}$ sur un ensemble** E est un couple (E, Γ), où Γ est une partie de E^2. Γ est le *graphe* de la relation binaire.

$$\forall x, y \in E, \quad x\mathcal{R}y \iff (x, y) \in \Gamma.$$

Remarque

$\mathcal{R}$ correspond à la définition d'un prédicat à deux variables sur E que l'on donne souvent directement.

Exemples

1. La relation $\leq$ sur $\mathbb{R}$.
2. La relation « divise » définie sur $\mathbb{N}$ (resp. $\mathbb{Z}$) par :

$$\forall b, a \in \mathbb{N}\,(\text{resp. } \mathbb{Z}), \quad b \mid a \iff \exists q \in \mathbb{N}\,(\text{resp. } \mathbb{Z}), a = bq.$$

3. La relation $\subset$ sur $\mathcal{P}(E)$.
4. La relation de congruence modulo θ sur les réels ou les entiers relatifs :

$$x \equiv y\,[\theta] \Leftrightarrow_{def} \exists k \in \mathbb{Z}, y = x + k\theta$$

Définition 74 Soit $\mathcal{R}$ une relation binaire sur E.

1. $\mathcal{R}$ est **réflexive**, si :
$$\forall x \in E, \quad x\mathcal{R}x.$$
2. $\mathcal{R}$ est **symétrique**, si :
$$\forall (x, y) \in E^2, \quad x\mathcal{R}y \Longrightarrow y\mathcal{R}x.$$
3. $\mathcal{R}$ est **antisymétrique**, si :
$$\forall (x, y) \in E^2, \quad (x\mathcal{R}y \text{ et } y\mathcal{R}x) \Longrightarrow x = y.$$
4. $\mathcal{R}$ est **transitive**, si :
$$\forall (x, y, z) \in E^3, \quad (x\mathcal{R}y \text{ et } y\mathcal{R}z) \Longrightarrow x\mathcal{R}z.$$

Remarque

Attention : antisymétrique n'est pas le contraire de symétrique.
$\mathcal{R}$ n'est pas symétrique $\iff \exists (x, y) \in E^2, x\mathcal{R}y$ et $y\not\mathcal{R}x$.

2.2 Relations d'équivalence

Définition 75 Soit $\mathcal{R}$ une relation binaire dans E.

$\mathcal{R}$ est une relation d' *équivalence*, si $\mathcal{R}$ est réflexive, symétrique, et transitive.

Exemples

1. La relation de congruence modulo n dans $\mathbb{Z}$.
2. La relation de parallélisme dans l'ensemble des droites du plan (ou de l'espace).
3. La relation d'égalité dans un ensemble E.

Définition 76 Soit E un ensemble muni d'une relation d'équivalence $\mathcal{R}$.

1. Pour tout $x \in E$, la **classe d'équivalence** de x est :

$$\mathrm{cl}(x) = \{y \in E; x\mathcal{R}y\}.$$

   De plus,

$$\forall y \in \mathrm{cl}(x), \quad \mathrm{cl}(y) = \mathrm{cl}(x).$$

   On dit que $y \in \mathrm{cl}(x)$ est un **représentant** de la classe de x.
2. L'ensemble $E/\mathcal{R}$ des classes d'équivalences de E pour la relation $\mathcal{R}$ est une partition de E. C'est l'**ensemble quotient** de E par $\mathcal{R}$.
3. On note souvent la classe de x par $\dot{x}$.

Exemple

1. L'ensemble quotient de $\mathbb{Z}$ par la relation de congruence modulo n noté usuellement $\mathbb{Z}/n\mathbb{Z}$ est :
$$\mathbb{Z}/n\mathbb{Z} = \{\dot{0}, \dot{1}, \ldots, n \dot{-} 1\}.$$
2. pour simplifier les notations, nous confondrons $\mathbb{Z}/n\mathbb{Z}$ avec l'ensemble
$$\{0, 1, \ldots, n-1\}.$$
3. Pour tout $x \in \mathbb{Z}$, on note $\overline{x}$ l'unique entier dans $\{0, ..., n-1\}$ tel que $x \equiv \bar{x}\,[n]$.

2.3 Relations d'ordre

2.3.1 Préordre, ordre, ordre total, ordre partiel

Définition 77 Soit $\mathcal{R}$ une relation binaire sur E.

1. $\mathcal{R}$ est une relation de *préordre*, si $\mathcal{R}$ est réflexive, et transitive.
2. $\mathcal{R}$ est une relation d' *ordre*, si $\mathcal{R}$ est réflexive, antisymétrique, et transitive.

Exemples

1. $\leq$ est un ordre sur $\mathbb{R}$.
2. $|$ est un ordre sur $\mathbb{N}$, mais seulement un préordre sur $\mathbb{Z}$ (la relation n'est pas anti-symétrique sur $\mathbb{Z}$).
3. $\subset$ est un ordre sur $\mathcal{P}(E)$.
4. Ordre **lexicographique** sur $\mathbb{R}^2$:

$$(x,y)\mathcal{L}(z,t) \iff ((x<z) \text{ ou } (x=z \text{ et } y\leq t)).$$

Remarques

1. Un ensemble E muni d'une relation d'ordre, notée $\preceq$ ou plus simplement $\leq$, est dit **ordonné**.
2. • $x \geq y \iff y \leq x$.
   • $x < y \iff (x \leq y \text{ et } x \neq y)$.
   • $x > y \iff y < x$.

Définition 78 Soit $\leq$ une relation de préordre sur E.
• $\leq$ est une relation de préordre *total* si :

$$\forall (x,y) \in E^2, \quad x \leq y \text{ ou } y \leq x.$$

Exemples

1. $\leq$ sur $\mathbb{R}$, l'ordre lexicographique sur $\mathbb{R}^2$, sont des relations d'ordre total.
2. $|$ sur $\mathbb{N}$, $\subset$ sur $\mathcal{P}(E)$ sont des relations d'ordre partiel.
3. $|$ sur $\mathbb{Z}$, est une relation de préordre partiel.

2.4 Préordres et ordres

Définition 79 Soit $\preceq$ un préordre sur un ensemble E et R une relation d'équivalence sur E. On dit que la relation de préordre est compatible avec la relation d'équivalence R si et seulement si

$$\forall\, (x,y,x',y') \in E^2 : xRx' \text{ et } yRy' \text{ et } x \preceq y \text{ alors } x' \preceq y'$$

Proposition 142 Si $\preceq$ est un préordre sur un ensemble E et R une relation d'équivalence sur E. Si la relation de préordre $\preceq$ est compatible avec la relation d'équivalence R alors le préordre définit un préordre $\dot{\preceq}$ sur E/R par :

$$\forall\, (\dot{x},\dot{y}) \in (E/R)^2 : \ \dot{x}\dot{\preceq}\dot{y} \Leftrightarrow_{def} x \preceq y$$

Soit $\preceq$ un préordre sur un ensemble E. On définit sur E la relation binaire ρ définie par :

$$\forall\,(x,y)\in E^2,\ x\rho y \Leftrightarrow x\preceq y \text{ et } y\preceq x$$

Proposition 143
La relation binaire ρ est une relation d'équivalence sur E, elle est compatible avec la relation de préordre $\preceq$.

Corollaire 1 Dans ces conditions, $\dot{\preceq}$ est une relation d'ordre sur E/ρ.En particulier, si $\preceq$ est un préordre total sur E alors $\dot{\preceq}$ est un ordre total sur E/ρ. On appelle cet ordre, l'*ordre quotient* défini par le préordre $\preceq$.

2.4.1 Eléments remarquables d'un ensemble préordonné

Définition 80 Soit $(E,\leq)$ un ensemble préordonné, et A une partie de E.
- Soit $M\in E$. M est un **majorant** de A si :

$$\forall x\in A,\quad x\leq M.$$

- Soit $m\in E$. m est un **minorant** de A si :

$$\forall x\in A,\quad m\leq x.$$

- A est une partie **majorée** de E, si elle admet un majorant.
  A est une partie **minorée** de E, si elle admet un minorant.
  A est **bornée** si A est majorée et minorée.

Exemples

1. Soit $(\mathbb{R},\leq)$ et $A=[0,1[$.
   -1 est un minorant de A et 2 est un majorant de A.
   0 est un minorant de A, mais 1 n'est pas un majorant de A.
2. Dans $\mathbb{R}^2$ muni de l'ordre lexicographique, (-1,5) est un minorant de $\mathbb{R}_+^2$.
3. Dans $\mathbb{N}$ ordonné par la relation divise, 6 est un majorant de $\{1,2,3\}$.

Définition 81 Soit $(E,\leq)$ un ensemble préordonné, et A une partie de E.
- Si A possède un majorant $M\in A$, M est un **plus grand élément** de A.
- Si A possède un minorant $m\in A$, m est un **plus petit élément** de A.

Il n'y a pas en général uncité d'un plus grand élément ou d'un plus petit élément, mais si la relation de préordre définit un ordre sur E alors il y a uncité du plus grand élément, s'il existe et du plus petit élément s'il existe.

Définition 82 Soit $(E,\leq)$ un ensemble ordonné, et A une partie de E.
- Si A possède un majorant $M\in A$, M est unique : c'est le **plus grand élément** de

A, noté $\max A$.

$$M = \max A \iff M \in A \text{ et } \forall x \in A, \quad x \leq M.$$

- Si A possède un minorant $m \in A$, m est unique : c'est le **plus petit élément** de A, noté $\min A$.

$$m = \min A \iff m \in A \text{ et } \forall x \in A, \quad m \leq x.$$

Index

Bibliographie

[AAG87] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 358–370, New York, 1987. IEEE.

[ABDT98] L. O. Alima, J. Beauquier, A. K. Datta, and S. Tixeuil. Self-stabilization with global rooted synchronizers. In *ICDCS '98 : Proceedings of the The 18th International Conference on Distributed Computing Systems*, pages 102–109, Washington, DC, USA, 1998. IEEE Computer Society.

[ACT00] Marcos Kawazoe Aguilera., Wei Chen, and Sam Toueg. On quiescent reliable communication. *SIAM J. Comput.*, 29(6) :2040–2073, 2000.

[ADG91] A Arora, S Dolev, and MG Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1 :11–18, 1991.

[AG85] B Awerbuch and R G Gallager. Distributed BFS algorithms. In *26th Symposium on Foundations of Computer Science (FOCS ' 85)*, pages 250–256, 1985.

[AG87] B Awerbuch and R G Gallager. A new distributed algorithm to find breadth first search trees. *IEEE Transactions on Information Theory*, IT33(3) :315–322, 1987.

[AG94] A Arora and MG Gouda. Distributed reset. *IEEE Transactions on Computers*, 43 :1026–1038, 1994.

[AGLP89] B Awerbuch, A V Goldberg, M Luby, and S A Plotkin. Network decomposition and locality in distributed computation. In *IEEE Symposium on Foundations of Computer Science (FOCS '89)*, pages 364–369, 1989.

[AK93] S. Aggarwal and S. Kutten. Time optimal self-stabilizing spanning tree algorithm. In *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, Berlin, 1993. Springer-Verlag.

[AKM+93] B Awerbuch, S Kutten, Y Mansour, B Patt-Shamir, and G Varghese. Time optimal self-stabilizing synchronization. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC '93)*, pages 652–661, 1993.

[AKY90] Y Afek, S Kutten, and M Yung. Memory-efficient self-stabilization on general networks. In Springer-Verlag, editor, *WDAG90 Distributed Algorithms 4th International Workshop Proceedings*, volume 486 of *Lecture Notes in Computer Science*, pages 15–28, 1990.

[AP90] B Awerbuch and D Peleg. Network synchronization with polylogarithmic overhead. In *IEEE Symposium on Foundations of Computer Science (FOCS '90)*, pages 514–522, 1990.

[APSV91] B Awerbuch, B Patt-Shamir, and G Varghese. Self-stabilization by local checking and correction. In *FOCS91 Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 268–277, 1991.

[APSV94] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Bounding the unbounded (extended abstract). In *Proceeding 13th IEEE INFOCOM*, pages 773–783, 1994.

[Asp03] James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2–3) :165–175, September 2003.

[Aum02] Olivier Aumage. *Madeleine : une interface de communication performante et portable pour exploiter les interconnexions hétérogènes de grappes.* Thèse de doctorat, spécialité informatique, École normale supérieure de Lyon, 46, allée d'Italie, 69364 Lyon cedex 07, France, September 2002. 154 pages.

[Awe85] B. Awerbuch. Complexity of network synchronization. *Journal of the Association of the Computing Machinery*, 32(4) :804–823, 1985.

[Awe89] B Awerbuch. Distributed shortest paths algorithms. In *21st Annual ACM Symposium on Theory of Computing (STOC '89)*, pages 490–500, 1989.

[Awe90] B Awerbuch. Shortest paths and loop-free routing in dynamic networks. In *Proceedings of the Annual ACM SIGCOMM Symposium on Communication Architectures and Protocols*, Philadelphia, Pennsylvania, 1990.

[BCPV03] C Boulinier, S Cantarell, F Petit, and V Villain. A note on a self-stabilizing local mutual exclusion algorithm. Technical Report RR03-02, LaRIA, University of Picardie Jules Verne, 2003. Available at `http://www.laria.u-picardie.fr/~petit/publi/TR2003-02.ps.gz`.

[BDGM00] J Beauquier, A.K. Datta, M Gradinariu, and F Magniette. Self-stabilizing local mutual exclusion and daemon refinement. In *Proceedings of the 14th International Conference on Distributed Computing (DISC 2000), Springer-Verlag LNCS :1914*, pages 223–237, 2000.

[BDPV99] A Bui, AK Datta, F Petit, and V Villain. State-optimal snap-stabilizing PIF in tree networks. In *Proceedings of the Forth Workshop on Self-Stabilizing Systems (WSS'99)*, pages 78–85. IEEE Computer Society Press, 1999.

[Bel58] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1) :87–90, 1958.

[Ber61] C Berge. Färbung von graphen, deren sämtliche bzw. deren ungeraden kreise satrr sind (zusammenfassung). *Wiss. Z. Marin Luther Univ. Halle Wittenberg*, 10 :114–115, 1961.

[Ber83] C Berge. *Graphes*. Gauthier-Villars (Paris), 1983.

[Bie91] D. Bienstock. On complexity of testing for odd holes and induced odd paths. *Discrete Mathematics*, 90 :85–92, 1991.

[Bor93] Jorge Luis Borges. *Oeuvres complètes*. Gallimard, 1993.

[Bou06] C Boulinier. Unison as a self-stabilizing wave stream algorithm in asynchronous anonymous networks. Technical Report RR06-9, LaRIA, University of Picardie Jules Verne, 2006. Available at `http://www.laria.u-picardie.fr/~boulinie/`.

[BPV04] C Boulinier, F Petit, and V Villain. When graph theory helps self-stabilization. In *PODC '04 : Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 150–159, 2004.

[BPV05] C Boulinier, F Petit, and V Villain. Synchronous vs. asynchronous unison. In *7th Symposium on Self-Stabilizing Systems (SSS'05), LNCS 3764*, pages 18–32, 2005.

[BPV06] C Boulinier, F Petit, and V Villain. Toward a time-optimal odd phase clock unison in trees. In Springer-Verlag, editor, *Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'06)*, Lecture Notes in Computer Science, 2006.

[BV97] P Boldi and S Vigna. Self-stabilizing universal algorithms. In *Proceedings of the Third Workshop on Self-Stabilizing Systems*, pages 141–156. Carleton University Press, 1997.

[Can03] S Cantarell. *Group Mutual Exclusion and Self-Stabilization*. PhD thesis, LRI, Université de Paris-Sud, Orsay, France, 2003.

[Cav47] J. Cavaillès. *Sur la logique et la théorie de la science*. P.U.F., Paris, 1947.

[CDP03] S Cantarell, AK Datta, and F Petit. Self-stabilizing atomicity refinement allowing neighborhood concurrency. In Springer-Verlag, editor, *DSN SSS'03 Workshop : 6th Symposium on Self-Stabilizing Systems (SSS '03)*, volume 2704 of *Lecture Notes in Computer Science*, pages 102–112, 2003.

[CDPV02] A Cournier, AK Datta, F Petit, and V Villain. Snap-stabilizing PIF algorithm in arbitrary networks. In *IEEE 22nd International Conference on Distributed Computing Systems (ICDCS 02)*, pages 199–206. IEEE Computer Society Press, 2002.

[CFG92] JM Couvreur, N Francez, and M Gouda. Asynchronous unison. In *Proceedings of the 12th IEEE International Conference on Distributed Computing Systems (ICDCS'92)*, pages 486–493, 1992.

[CGH95] DM Chickering, D Geiger, and D Heckerman. On finding a cycle basis with a shortest maximal cycle. *Information Processing Letters*, 54 :55–58, 1995.

[CH73] C.C. Chang and H.J.Keisler. *Model theory*. North-Holland Publ. Co, 1973.

[CHP71] P.J. Courtois, F. Heymans, and D.L. Parnas. Concurrent control with readers and writers. *Communications of the Association of the Computing Machinery*, 14(10) :667–668, 1971.

[CHT92] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. In Maurice Herlihy, editor, *Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing (PODC'92)*, pages 147–158, Vancouver, BC, Canada, 1992. ACM Press.

[CRKGLA89] C Cheng, R Riley, S Kumar, and J Garcia-Luna-Aceves. A loop-free extended bellman-ford routing protocol without bouncing effect. In *Proceedings of the Annual ACM SIGCOMM Symposium on Communication Architectures and Protocols*, pages 224–236, Austin, Texas, 1989.

[CRST05] M Chudnovski, N Robertson, P Seymour, and R Thomas. The strong perfect graph theorem. to appear in Annals of Math., 2005.

[CT91] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for asynchronous systems. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC'91)*, pages 325–340, Montreal, Quebec, Canada, 1991. ACM Press.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2) :225–267, 1996.

[DDT03] Sylvie Delaet, Bertrand Ducourthial, and Sébastien Tixeui. Self-stabilization with r-operators in unreliable directed networks. Technical Report 1361, LRI, University of Paris XI, 2003.

[Des37] R. Descartes. *Discours de la méthode*. Par l'imprimeur Ian Maire, Leyde, Hollande, 1637.

[DGF98] C. Delporte-Gallet and H. Fauconnier. Synchronized phase systems. Technical report, Université Denis Diderot, LIAFA, Paris, France, 1998.

[DGF99] Carole Delporte-Gallet and Hugues Fauconnier. Real-time fault-tolerant atomic broadcast. In *Symposium on Reliable Distributed Systems*, pages 48–55, 1999.

[DGS96] S. Dolev, MG Gouda, and M Schneider. Memory requirements for silent stabilization. In *podc96*, pages 27–34, 1996.

[Dij65] E.W. Dijkstra. Solution to a problem in concurrent programming control. *Communications of the Association of the Computing Machinery*, 8(9) :569, 1965.

[Dij68] E Dijkstra. *Cooperating Sequential Processes*. Academic Press, 1968.

[Dij74] EW Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17 :643–644, 1974.

[DIM93] S Dolev, A Israeli, and S Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, 7 :3–16, 1993.

[DIM97] S Dolev, A Israeli, and S Moran. Uniform dynamic self-stabilizing leader election. *IEEE Transactions on Parallel and Distributed Systems*, 8(4) :424–440, 1997.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2) :288–323, 1988.

[DNT06] P Danturi, M Nesterenko, and S Tixeuil. Self-stabilizing philosophers with generic conflicts. In *8th International Symposium on Stabilizing, Safety, and Security of Distributed Systems (SSS'06)*, 2006.

[Dol00] S Dolev. *Self-Stabilization*. The MIT Press, 2000.

[DT01] B. Ducourthial and S. Tixeuil. Self-stabilization with r-operators. *Distributed Computing*, 14 :147–162, 2001.

[Duc98] B. Ducourthial. New operators for computing with associative nets. In *The 5th International Colloquium On Structural Information and Communication Complexity Proceedings (SIROCCO'98)*, pages 51–65. Carleton University Press, 1998.

[Duc06] B. Ducourthial. r-semi-groups : an algebra for computing on networks. Technical report, Heudiasyc, Université de Technologie de Compiègne, 2006.

[Fin79] SG Finn. Resynch procedures and a fail-safe network protocol. In *IEEE Trans. on Commun.*, pages 840–845, 1979.

[FLP85] MJ Fischer, NA Lynch, and MS Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the Association of the Computing Machinery*, 32 :374–382, 1985.

[FRT04] E Frometin, M Raynal, and F Tronel. About classes of problems in asynchronous distributed systems with process crashes. Technical Report Publication interne 1178, IRISA, University of Rennes, 2004.

[Gal82] R G Gallager. Distributed minimum hop algorithms. Technical Report LIDS-P-1175, M.I.T., Laboratory for Information and Decision Systems, 1982.

[GGHP96] S Ghosh, A Gupta, T Herman, and S.V Pemmaraju. Fault-containing self-stabilizing algorithms. In *Symposium on Principles of Distributed Computing (PODC)*, pages 45–54, 1996.

[GH90] MG Gouda and T Herman. Stabilizing unison. *Information Processing Letters*, 35 :171–175, 1990.

[GHJT06] W. Goddard, S.T. Hedetniemi, D.P. Jacobs, and V. Trevisian. Distance-k information in self-stabilizing algorithms. In *The 13th International Colloquium On Structural Information and Communication Complexity Proceedings (SIROCCO'06)*, 2006.

[GJT76] Michael R Garey, David S Johnson, and Robert E Tarjan. The planar hamiltonian circuit problem is np-complete. *SIAM J. Comput*, 5 :704–714, 1976.

[GM91] M.G. Gouda and N.J. Multari. Stabilizing communication protocols. *IEEE Transactions on Computers*, 40(4) :448–458, 1991.

[GM01] E. Godard and Y. Métivier. A characterization of classes of graphs recognizable by local computations with initial knowledge. In *International colloquium on structural information and communication complexity (SIROCCO)*, pages 179–193. Carleton Scientific Press, 2001.

[GM02] E. Godard and Y. Métivier. Equivalence of structural knowledges in distributed algorithms. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 149–164. Carleton scientific press, 2002.

[GMM00] Emmanuel Godard, Yves Métivier, and Anca Muscholl. The power of local computations in graphs with initial knowledge. In *TAGT'98 : Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations*, pages 71–84, London, UK, 2000. Springer-Verlag.

[GMM04] E. Godard, Y. Métivier, and A. Muscholl. Characterizations of classes of graphs recognizable by local computations. *Theory of Computing Systems*, 37 :249–293, 2004.

[HC92] ST Huang and NS Chen. A self-stabilizing algorithm for constructing breadth-first trees. *Information Processing Letters*, 41 :109–117, 1992.

[Her91] T Herman. *Adaptativity through distributed convergence.* Ph.d. thesis, Dept Computer Science, University of Texas, Austin, USA, 1991.

[Her01] T Herman. A phase clock tutorial. Draft, 2001.

[HG95] T Herman and S Ghosh. Stabilizing phase-clocks. *Information Processing Letters*, 54 :259–265, 1995.

[HG05] F. Furman Haddix and Mohamed G. Gouda. A general alternator. In *IASTED PDCS*, pages 409–413, 2005.

[HL98] Shing-Tsaan Huang and Tzong-Jye Liu. Four-state stabilizing phase clock for unidirectional rings of odd size. *Inf. Process. Lett.*, 65(6) :325–329, 1998.

[HL01] S.T. Huang and T.J. Liu. Phase synchronization on asynchronous uniform rings with odd size. In *IEEE Transactions on Parallel and Distributed System*, volume 12(6), pages 638–652, 2001.

[IPS82] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamiltonian paths in grid graphs. *SIAM J. Comput.*, 23 :676–686, November 1982.

[Joh97] Colette Johnen. Memory efficient, self-stabilizing algorithm to construct bfs spanning trees. In *PODC '97 : Proceedings of the sixteenth annual ACM*

symposium on Principles of distributed computing, page 288, New York, NY, USA, 1997. ACM Press.

[Jou98] Y.-J. Joung. Asynchronous group mutual exclusion (extended abstract). In ACM Press, editor, *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing (PODC 98)*, pages 51–60, 1998.

[Jou00] Yuh-Jzer Joung. Asynchronous group mutual exclusion. *Distrib. Comput.*, 13(4) :189–206, 2000.

[KA97] S. Kulkarni and A. Arora. Multitolerant barrier synchronization. *Information Processing Letters*, 64(1) :29–36, 1997.

[KA98a] S. Kulkarni and A. Arora. Low-cost fault-tolerance in barrier synchronizations. In *ICPP : 27th International Conference on Parallel Processing*, 1998.

[KA98b] S. Kulkarni and A. Arora. Multitolerance in distributed reset. *Chicago Journal of Theoretical Computer Science, Special Issue on Self-Stabilization*, 1998.

[Kan87] E. Kant. *Critique de la raison pure*. G.F., Paris, 1787. Traduction de Jules Barni.

[Kar72] Rabin M. Karp. Reducibility among combinatorial problems. In E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[KMM02] D Kondou, H Masuda, and T Masuzawa. A self-stabilizing protocol for pipelined pif in tree networks. In *icdcs02*, pages 181–183. IEEE Computer Society, 2002.

[Kri75] M. S. Krishnamoorthy. An np-hard problem in bipartite graphs. In *SIGACT*, pages 1–26, 1975.

[Kru79] HSM Kruijer. Self-stabilization (in spite of distributed control) in tree-structured systems. *Information Processing Letters*, 8 :91–95, 1979.

[KY02] H Kakugawa and M Yamashita. Self-stabilizing local mutual exclusion on networks in which process identifiers are not distinct. In *Proceedings of the 21st Symposium on Reliable Distributed Systems (SRDS2002)*, pages 202–211, 2002.

[LAD+96] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St Pierre, David S. Wells, Monica C. Wong-Chan, Shaw-Wen Yang, and Robert Zak. The network architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2) :145–158, 1996.

[Lam74] L. Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8) :453–455, 1974.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2) :133–169, 1998.

[Lam01] Butler Lampson. The abcd's of paxos. In *PODC '01 : Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, page 13, New York, NY, USA, 2001. ACM Press.

[Lov72] L Lovász. A characterization of perfect graphs. *J. Combin. Theory*, B,13 :95–98, 1972.

[LS95] C Lin and J Simon. Possibility and impossibility results for self-stabilizing phase clocks on synchronous rings. In *wss95*, pages 10.1–10.15, 1995.

[Lyn96a] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[Lyn96b] N. Lynch. *Distributed algorithms*. Morgan Kaufmann, Chichester, UK, 1996.

[Maz97] Antoni W. Mazurkiewicz. Distributed enumeration. *Information Processing Letters*, 61(5) :233–239, 1997.

[MDN+06] Christine Morin, Alexandre Denis, Raymond Namyst, Olivier Aumage, and Renaud Lottiaux. *Encyclopédie de l'informatique et des systèmes d'information*, chapter Des réseaux de calculateurs aux grilles de calcul. Number ISBN : 2-7117-4846-4. Vuibert, December 2006.

[Mis91] J Misra. Phase synchronization. *Information Processing Letters*, 38(2) :101–105, 1991.

[NA02] M Nesterenko and A Arora. Stabilization-preserving atomicity refinement. *Journal of Parallel and Distributed Computing*, 62(5) :766–791, 2002.

[Nol02] F Nolot. *Self-stabilizing phase clock in distributed systems*. PhD thesis, LaRIA, Université de Picardie Jules Verne, Amiens, France, 2002. Dissertation in French.

[NS93] Moni Naor and Larry Stockmeyer. What can be computed locally ? In *STOC '93 : Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 184–193, New York, NY, USA, 1993. ACM Press.

[NV01] F Nolot and V Villain. Universal self-stabilizing phase clock protocol with bounded memory. In *IPCCC '01, 20th IEEE International Performance, Computing, and Communications Conference*, pages 228–235, 2001.

[Pas56] B. Pascal. *les provinciales*. Par l'imprimeur Pierre Le Petit, Paris, France, 1656.

[PV00] F Petit and V Villain. Optimality and self-stabilization in rooted tree networks. *Parallel Processing Letters*, 10(1) :3–14, 2000.

[Ray90] M. Raynal. *La communication et le temps dans les réseaux et les systèmes répartis*. Eyrolles, Paris, F, 1990.

[Ray92] M. Raynal. *Synchronisation et état global dans les systèmes répartis*. Eyrolles, Paris, F, 1992.

[Ray04] M Raynal. A short introduction to failure detectors for asynchronous distributed systems. Technical Report Publication interne 1613, IRISA, University of Rennes, 2004.

[RH88] M. Raynal and JM Helary. *Synchronisation et contrôle des systèmes et programmes répartis*. Eyrolles, Paris, F, 1988.

[RH90] M. Raynal and JM Helary. *Synchronization and control of distributed Systems and programs*. John Wiley and Sons, Chichester, UK, 1990.

[RM05] L Rilling and C Morin. Partage de données transparent et tolérant aux fautes pour la grille. In *RENPAR'16 Journées Composants, Le croisic, France*, 2005.

[Sal04] J. M. Salanskis. *L'herméneutique formelle*. Editions du CNRS., Paris, 2004.

[SRR95] SK Shukla, DJ Rosenkrantz, and SS Ravi. Observations on self-stabilizing graph algorithms for anonymous networks. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 7.1–7.15, 1995.

[Tel90] Tel. Total algorithms. *ALCOM : Algorithms Review, Newsletter of the ESPRIT II Basic Research Actions Program Project no. 3075 (ALCOM)*, 1, 1990.

[Tel91] G. Tel. *Topics in Distributed Algorithms*. Cambridge University Press, 1991.

[Tel94] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[Tel00] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.

[Tix02] S Tixeuil. *Auto-stabilisation Efficace*. Thèse de doctorat, spécialité informatique, University of Paris-Sud XI, January 2002.

[Tix06] S Tixeuil. Vers l'autostabilisation des systèmes à grande échelle. Technical Report RR LRI 1452, LRI, University of Paris-Sud, 2006.

[Var93] G Varghese. Self-stabilization by local checking and correction (Ph.D. thesis). Technical Report MIT/LCS/TR-583, MIT, 1993.