HAL
open science

Problèmes de communication dans les systèmes distribués: ruptures et corruptions.

Antoine Gaillard

To cite this version:

Antoine Gaillard. Problèmes de communication dans les systèmes distribués: ruptures et corruptions.. Computer Science [cs]. Ecole Polytechnique X, 2009. English. NNT : . pastel-00004991

HAL Id: pastel-00004991

https://pastel.hal.science/pastel-00004991

Submitted on 21 Apr 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Problèmes de communication dans les systèmes distribués: ruptures et corruptions

Antoine Gaillard

Introduction

Un système distribué est un ensemble d'entités de calcul indépendantes, appelées *ordinateurs*, *processeurs*, ou encore *processus*, selon le niveau d'abstraction, qui communiquent et collaborent par l'intermédiaire d'un *médium de communication*. On distingue plusieurs types de systèmes distribués qui diffèrent par leur médium de communication ainsi que leur degré de synchronisme. Dans les systèmes à *mémoire partagée*, les processus partagent une même structure de données qu'ils peuvent lire et dans laquelle ils peuvent écrire ; dans les systèmes à *échanges de messages*, les processus s'envoient des messages le long de canaux de communication. Les systèmes sont dits *synchrones* s'il existe des bornes sur le délai d'acheminement des messages et les vitesses relatives des processus, respectivement, ou *asynchrones* si on ne fait aucune référence au temps physique.

La collaboration entre les processus d'un système pour l'accomplissement d'une tâche commune, par exemple se mettre d'accord sur une valeur unique, est en grande partie fondée sur leur capacité à communiquer. Une des questions importantes que l'on peut se poser est de savoir ce qu'il advient de cette possibilité de collaboration si les communications ne sont pas fiables, et par quels moyens on peut la garantir. Nous distinguons deux approches principales qui ont pour but de répondre à cette question. La première approche, adoptée par la communauté du *Calcul distribué tolérant aux pannes*, consiste à considérer que les communications imparfaites sont partie intégrante de l'environnement dans lequel les processus sont censés collaborer, et à tenter de décrire des solutions valables *en dépit* de cette absence de fiabilité. La seconde approche suppose que le graphe de communication doit satisfaire certaines propriétés afin que les processus puissent collaborer. La question est alors de déterminer dans quelle mesure et par quels moyens il est possible de faire en sorte que ces propriétés soient satisfaites. C'est l'approche communément utilisée pour l'étude des *Réseaux mobiles ad hoc*, dans lesquels la

topologie du graphe de communication est susceptible d'être modifiée par les mouvements des nœuds du réseau.

Calcul distribué tolérant aux pannes

La quasi-totalité des travaux en Calcul distribué tolérant aux pannes ont associé les erreurs de transmission à des défaillances des composants du système, que ce soit les processus (arrêt prématuré, définitif ou non, omissions d'envoi ou de récéption de messages, déviations arbitraires par rapport au code), ou le médium de communication (pertes, créations, duplications ou corruptions de messages). D'autre part, le type de pannes et le degré de synchronisme ont toujours été considérés comme des paramètres *indépendants* qui déterminent un système particulier.

À notre connaissance, les premiers à se demander s'il était nécessaire, ou simplement utile, d'incriminer systématiquement les composants du système ont été Santoro et Widmayer [42]. Ils ont remarqué que le fait d'associer les erreurs de transmission à des défaillances des processus ou du médium de communication pouvait conduire à des conclusions malheureuses, en particulier dans le cas d'erreurs dynamiques ou transitoires. Ils ont ainsi développé un modèle en rounds, appelé *Transmission Faults Model*, qui prend en compte la localisation des erreurs de transmission sans en spécifier la cause, et ont montré que ce modèle était adapté à l'étude des systèmes distribués sujets à des pannes dynamiques ou transitoires. Ils ont, en particulier, démontré qu'il était possible de résoudre des problèmes d'accord en tolérant ce type de pannes, alors que c'est impossible dans les modèles où, par exemple, les pannes sont des omissions d'envoi ou de réception. En effet, dans ces derniers, la perte d'un unique message par chacun des processus au cours du calcul rend fautif le système tout entier. La notion de *composant fautif* n'existe donc pas dans le modèle Transmission Faults. Cependant, le type de pannes et le degré de synchronisme sont, dans [42], toujours considérés comme des paramètres distincts qui caractérisent un système distribué.

En 1998, Gafni a développé dans [22] un autre modèle en rounds, le *Round-by-Round Fault Detector Model* (RRFD), pour l'étude des systèmes distribués. L'idée de Gafni était d'étudier comment évoluaient les calculs *round après round*, et d'abstraire les propriétés des communications entre les processus, ainsi que les garanties fournies par le système, par une unique entité abstraite, le module *Round-by-Round Fault Detector*. A chaque round r

et pour chaque processus p, le module RRFD fournit un ensemble de processus *suspects* dont p ne reçoit pas de messages à ce round. Seule l'absence de transmission est prise en compte dans le modèle ; la raison pour laquelle un processus q est suspecté - arrêt prématuré ou retard dans l'envoi du message - n'est pas spécifiée : d'une certaine manière, le module RRFD capture ici à la fois le type de pannes et le degré de synchronisme. Cependant, la notion de *composant fautif* est toujours présente. En effet, le médium de communication est supposé fiable. Si un processus p ne reçoit pas de message provenant d'un processus q à un round donné, alors q est implicitement désigné comme responsable de l'erreur de transmission.

Récemment, Charron-Bost et Schiper [17] se sont, eux aussi, penchés sur la pertinence de ces deux hypothèses fondamentales de la plupart des modèles. S'inspirant des travaux de Gafni et de Santoro et Widmayer, ils ont développé un modèle en rounds pour l'étude des systèmes distribués en présence de panne *bénignes*[1]. Dans ce modèle, qu'ils ont appelé modèle *Heard-Of*, ou modèle HO, les calculs s'exécutent en rounds, *structures logiques fermées pour les communications*, en ce sens que tout message envoyé à un round ne peut être reçu qu'à ce round. À chaque round, chaque processus envoie un message à chacun des autres, puis attend de recevoir un message des autres processus. Pour chaque round r et chaque processus p, $HO(p, r)$ représente l'ensemble des processus dont p a reçu un message au round r et est appelé *ensemble d'écoute*. Une erreur de transmission bénigne de q à p au round r est caractérisée par le fait que q n'appartient pas à $HO(p, r)$. Le degré de synchronisme et le type de pannes sont capturés par une même entité abstraite de haut niveau, un prédicat sur les collections d'ensembles d'écoute appelé *prédicat de communication* ; la notion de composant fautif est totalement absente.

Comme l'indiquent Charron-Bost et Schiper, le modèle HO et le modèle RRFD sont similaires sur plusieurs points. D'une part, l'un comme l'autre ne spécifie pas le médium de communication. Les processus échangent des morceaux d'information, appelés *messages*, mais la manière dont les messages sont envoyés et reçus par les pocessus n'est pas précisée. D'autre part, tout module RRFD fournit, à chaque round r et pour chaque processus p, le complémentaire dans Π de l'ensemble $HO(p, r)$. En revanche, les modèles RRFD et HO diffèrent sur la manière d'interpréter les prédicats :

[1] Ce type de pannes regroupe les crashs, les omissions d'envoi et de réception, ainsi que les pertes de messages au cours de leur acheminement.

dans le modèle RRFD, l'hypothèse de fiabilité du médium de communication réintroduit la notion de composant fautif. Dans le modèle HO, au contraire, cette notion est absente et les prédicats caractérisent uniquement des propriétés vérifiées par les communications.

Contribution

Le modèle HO a été décrit initialement pour l'étude des systèmes distribués sujets à des pannes bénignes, qui ont pour effet une absence de communication. Or, certains modèles "classiques" considèrent des pannes qui ont pour effet la transmission d'une information erronée, par exemple la corruption d'un message au cours de son acheminement. Le premier chapitre de la présente thèse reprend le travail présenté dans [13] et généralise le modèle HO au cas où les processus sont susceptibles de recevoir des messages différents de ceux qui auraient dû leur être envoyés, c'est-à-dire au cas des *erreurs de transmission par valeurs*, ou *pannes par valeurs*. Nous introduisons l'ensemble $SHO(p, r)$ qui représente l'ensemble des processus dont p a reçu, au round r, un message identique à celui qui aurait dû lui être envoyé à ce round, que nous appelons *ensemble d'écoute sûr*. Une erreur de transmission par valeurs de q à p au round r est alors caractérisée par le fait que q appartient à $HO(p, r)$ mais qu'il n'appartient pas à $SHO(p, r)$. Nous généralisons la notion de prédicat de communication que nous définissons à présent comme un prédicat sur les collections de paires d'ensembles $(HO(p, r); SHO(p, r))_{p\in\Pi, r>0}$.

Le modèle HO se situe à un haut niveau d'abstraction. Son but étant de permettre une description unifiée des différents systèmes "classiques", il est fondamental d'étudier dans quelle mesure il est possible d'exprimer leurs caractéristiques par des prédicats de communication. Dans [17], Charron-Bost et Schiper présentent des prédicats de communication qui, selon eux, "correspondent naturellement" à certains systèmes à échanges de messages sujets à des pannes bénignes que l'on rencontre dans la littérature classique. Dans le Chapitre 2, nous poursuivons cette démarche et la généralisons aux systèmes à mémoire partagée sujets à des pannes bénignes, ainsi qu'aux systèmes sujets à des erreurs de transmission par valeurs.

Une autre question que l'on peut se poser consiste à déterminer les prédicats de communication qui permettent de résoudre un problème particulier. Charron-Bost et Schiper [17] se sont attachés à l'étude du problème du Consensus, dans lequel tous les processus sont censés se mettre d'accord sur une valeur *unique*. Ils ont caractérisé le plus faible prédicat de communication per-

mettant la résolution de Consensus dans des systèmes sujets à des pannes bénignes et qui garantissent qu'à chaque round, tous les ensembles d'écoute sont deux à deux d'intersection non-vide. Dans le Chapitre 2, nous mettons ce résultat en relation avec les prédicats présentés et retrouvons des résultats précédemment établis relatifs à la résolubilité de Consensus dans certains types de systèmes.

Dans [17], Charron-Bost et Schiper ont présenté quatre algorithmes qui résolvent Consensus en présence d'erreurs de transmissions bénignes. Dans le Chapitre 3, nous adaptons ces algorithmes en sorte qu'ils tolèrent des erreurs de transmission par valeurs, et nous présentons, pour chacun d'eux, un prédicat de communication qui garantit la sûreté de chaque exécution et un prédicat suffisant pour assurer la vivacité.

Réseaux mobiles ad hoc

Un réseau mobile ad hoc peut être considéré comme un type de système distribué particulier, constitué de nœuds mobiles dans l'espace qui communiquent en s'envoyant des messages *via* des canaux de communication sans fil. Deux processus ne peuvent communiquer directement que s'ils se trouvent dans leur *rayon de transmission* respectif. La possible mobilité des nœuds peut donc causer des changements dans la topologie du graphe de communication, et rend par conséquent problématique la collaboration entre les nœuds. L'un des problèmes fondamentaux dans les réseaux mobiles ad hoc est celui du *Routage*, dont le but est de maintenir des chemins le long desquels les nœuds du réseau transmettent l'information qu'ils doivent échanger.

En 1981, Gafni et Bertsekas [23] ont introduit une technique algorithmique élégante, appelée *renversements de liens*, pour le routage des messages dans des réseaux sujets à de fréquents changements de topologie. Dans le problème considéré, le réseau contient une destination unique, et une direction virtuelle est associée à chacun des canaux de communication. Les directions des canaux doivent assurer que chaque nœud du réseau dispose d'un chemin dirigé vers la destination, c'est-à-dire que le graphe de communication soit *orienté vers la destination*, et que le graphe soit acyclique. Les messages peuvent ainsi être acheminés de tout nœud du réseau vers la destination en suivant les directions associées aux canaux de communication. Si, pour une raison quelconque[2], un nœud ne dispose plus d'un chemin vers la destination,

[2]Arrêt du fonctionnement d'un processus, changement dans la topologie du graphe de

le réseau doit s'auto-ajuster afin de restaurer cette propriété. Pour ce faire, certains nœuds *renversent* les directions associées à un ou plusieurs de leurs liens incidents. Les nœuds doivent, de plus, être capables de déterminer de manière autonome, en se fondant uniquement sur une information locale, s'il leur est nécessaire d'exécuter de tels renversements.

Gafni et Bertsekas [23] ont restreint leur étude au cas des graphes acycliques et ont présenté deux algorithmes, chacun avec une description abstraite ainsi qu'une implémentation. Dans chacune des descriptions abstraites, chacun des nœuds (autre que la destination) qui devient un *puits* (c'est-à-dire dont tous les liens incidents sont entrants) renverse certains de ses liens incidents. Les deux algorithmes diffèrent dans le choix des liens renversés par les puits. Dans l'algorithme *full reversal* (FR), tous les liens incidents sont renversés, tandis que dans l'algorithme *partial reversal* (PR), un puits ne renverse que les liens qui n'ont pas été renversés depuis la dernière fois qu'il a été un puits, s'il en existe, et les renverse tous sinon. Dans les deux implémentations proposées par Gafni et Bertsekas, les nœuds mettent à jour une variable, appelée *hauteur*, à valeurs dans un ensemble totalement ordonné, telle que deux nœuds distincts ont des hauteurs différentes. Un lien entre deux nœuds est orienté du nœud de plus grande hauteur vers le nœud de plus basse hauteur. Les renversements des liens sont alors implémentés par l'accroissement des hauteurs des nœuds considérés. Gafni et Bertsekas ont démontré que leurs deux implémentations étaient correctes, en ce sens que toutes deux terminent et que, de plus, le graphe final est acyclique et orienté vers la destination.

L'approche adoptée par Gafni et Bertsekas, fondée sur l'utilisation de hauteurs deux à deux distinctes à valeurs dans un ensemble totalement ordonné, présente de nombreux avantages. Tout d'abord, elle permet d'unifier de manière élégante les expressions des deux algorithmes. La convergence des deux algorithmes est alors prouvée en ne considérant que des propriétés abstraites des hauteurs et des fonctions qui les modifient. De plus, en variant la représentation des hauteurs ainsi que des fonctions, on peut s'attendre à obtenir d'autres algorithmes. Enfin, la structure totalement ordonnée de l'ensemble des valeurs prises par les hauteurs, combinée avec le fait que les hauteurs sont deux à deux distinctes, garantit que tous les graphes engendrés au cours d'une exécution sont acycliques. Cependant, plusieurs inconvénients apparaissent lors d'une étude plus approfondie. En

communication, etc.

premier lieu, la détermination distribuée des hauteurs initiales peut se révéler problématique. Ensuite, la correction de l'approche dépend du fait que les hauteurs sont non-bornées, ce qui, d'un point de vue pratique, limite fortement l'utilité de l'approche. Enfin, malgré l'apparente flexibilité résultant des différents types de hauteurs pouvant être considérés, la description d'algorithmes dont les exécutions diffèrent de celles des implémentations de *FR* et de *PR* n'est pas aisée.

Contribution

Le Chapitre 4 reprend le travail de [14]. Nous y présentons une nouvelle formalisation des algorithmes de renversements de liens, fondée uniquement sur un étiquetage binaire des *liens*. En utilisant cette approche plus directe, nous définissons un algorithme simple destiné à établir des routes dans des graphes acycliques, et nous déterminons des conditions suffisantes sur l'étiquetage du graphe d'entrée pour assurer la correction de l'algorithme. Les algorithmes *FR* et *PR* de Gafni et Bertsekas peuvent alors être vus comme des cas particuliers de notre algorithme pour des étiquetages initiaux uniformes. De plus, en considérant des étiquetages initiaux non-uniformes, notre algorithme capture plus que les seuls *FR* et *PR*. Pour terminer, cette formalisation nous permet de fournir une preuve simple du fait que notre algorithme résout le problème de l'orientation vers la destination. À notre connaissance, c'est la seule preuve directe du fait que l'algorithme abstrait *PR*, qui ne suppose pas l'existence de hauteurs totalement ordonnées, préserve l'acyclicité des graphes engendrés par une exécution.

Notre formalisation de l'algorithme général nous permet d'exprimer le nombre *exact* de pas de calcul effectués par chaque nœud au cours d'une exécution donnée. L'expression dépend uniquement du graphe étiqueté initial, et donne lieu à des formules simples dans les cas particuliers de *FR* et *PR*. Le fait d'avoir une formule exacte facilite la détermination des graphes *pire cas* et *meilleur cas*. Un autre sujet naturel d'étude est la *complexité en temps* qui peut être définie comme le nombre de rounds nécessaires à la convergence, au cours desquels plusieurs puits sont susceptibles d'effectuer simultanément un pas de calcul. On note que la complexité en travail globale correspond à la complexité en temps d'une exécution séquentielle, au cours de laquelle un seul puits prend un pas de calcul à chaque round.

Une autre question intéressante consiste à déterminer combien de rounds sont nécessaires pour qu'une exécution *complète* converge, c'est-à-dire une

exécution au cours de laquelle, à chaque round, *tous les puits* prennent un pas de calcul. Notre approche nous a permis de déterminer une formule exacte pour la complexité en temps de FR sur une chaîne. A l'aide de cette formule, nous avons déduit une borne inférieure sur la complexité en temps de toute exécution de FR, quel que soit le graphe initial.

Chapitre 1

Le modèle Heard-Of (HO)

Dans ce premier chapitre, nous présentons le modèle de calcul qui servira de cadre d'étude dans toute la première partie de la présente thèse.

Comme nous l'avons expliqué dans l'introduction, les calculs, dans ce modèle, sont composés de rounds, structures logiques *fermées pour les communications*, en ce sens que tout message envoyé lors d'un round donné ne peut être reçu qu'à ce round. La description technique des calculs est similaire à celles qu'ont présentées Dwork, Lynch et Stockmeyer dans [19] et Gafni dans [22] ; le modèle généralise ainsi la notion classique de rounds synchrones développée pour les systèmes synchrones [33].

Le modèle HO a été défini par Charron-Bost et Schiper [17] dans le but, entre autres, de disposer d'un outil d'étude des systèmes distribués sujets à des pannes *bénignes*[1] qui 1°) ne considère plus le type de pannes et le degré de synchronisme comme des paramètres indépendants qui déterminent un système particulier, et 2°) ne fait plus référence au *composant responsable de la panne*, en ne considérant plus que la notion d'*erreur de transmission*.

Nous avons généralisé le modèle HO afin de pouvoir étudier des systèmes dans lesquels les processus sont susceptibles de recevoir des messages différents de ceux qui auraient dû leur être envoyés [13] ; on parle alors d'*erreurs de transmission par valeurs*.

Dans le cas des pannes bénignes, le modèle est fondé sur la notion d'ensemble *Heard-Of* (HO) ou *ensemble d'écoute*, qui permet de caractériser les messages reçus par un processus à un round, et donc évalue la vivacité des

[1] On peut citer les omissions d'envoi ou de réception d'un message, ou encore l'arrêt prématuré et définitif d'un processus, communément appelé *crash*.

communications. Pour les erreurs de transmission par valeurs, nous introduisons ici la notion d'ensemble *Safe Heard-Of* (SHO) ou *ensemble d'écoute sûr*, qui caractérise la sûreté des communications, c'est-à-dire les messages reçus par un processus à un round donné dont le contenu correspond bien à celui qui *aurait dû* lui être envoyé.

1.1 Ensembles d'écoute. Prédicats de communication.

On suppose que l'on dispose d'un ensemble fini non vide Π de cardinal n, d'un ensemble de messages M, ainsi que d'un marqueur *null* n'appartenant pas à M indiquant le message vide. À chaque élément p de Π, nous associons un *processus*, formé des composants suivants : (a) un ensemble d'états noté $states_p$, (b) un sous-ensemble $init_p$ d'états initiaux et, pour tout entier positif r appelé *numéro de round*, (c) une fonction d'envoi de message S_p^r qui associe à chaque élément de $states_p \times \Pi$ un unique élément de $M \cup \{null\}$ et (d) une fonction de transition d'états T_p^r qui détermine le nouvel état du processus p en fonction de son état au début du round r et du vecteur partiel (indexé par Π) des messages reçus par p au round r. Plus formellement, on a :

$$S_p^r : states_p \times \Pi \longrightarrow M \cup \{null\},$$

et

$$T_p^r : states_p \times \left((M \cup \{null\})^{\Pi}\right)^* \longrightarrow states_p,$$

où $\left((M \cup \{null\})^{\Pi}\right)^*$ représente l'ensemble des vecteurs partiels, indexés par Π, d'éléments de $M \cup \{null\}$. La collection des processus est appelée *algorithme sur* Π.

1.1.1 Ensembles d'écoute

Les *calculs* s'effectuent en rounds successifs. À chaque round r, chaque processus p envoie à chaque processus q le message $S_p^r(s, q)$, où s est l'état de p au début du round r, puis, pour un certain sous-ensemble $HO(p, r)$ de Π représentant l'ensemble des processus desquels p reçoit un message au round r, troque son état courant contre le nouvel état $T_p^r(s, \vec{\mu}_p^r)$, où $\vec{\mu}_p^r$ est

le vecteur partiel des messages reçus par p à ce round (le support de $\vec{\mu}_p^r$ est donc $HO(p,r)$).

On suppose que tout calcul consiste en une infinité de rounds. Cette hypothèse est fondamentale, en ce sens qu'elle garantit la *vivacité des calculs* en assurant que chaque processus effectue une infinité de transitions d'états.

Il est, d'autre part, important d'insister ici sur le fait que, pour chaque processus p de Π, la fonction S_p^r est définie, pour tout round r, sur $states_p \times \Pi$. En d'autres termes, à chaque round, tous les processus sont tenus d'envoyer un message à chacun des autres. Cette hypothèse est, elle aussi, fondamentale : elle résulte du fait que nous tenons à séparer les aspects sémantiques, c'est-à-dire liés à l'algorithme, des aspects opérationnels liés aux hypothèses sur le système. Plus précisément, nous voulons pouvoir différencier le cas dans lequel un processus p ne reçoit pas de message d'un processus q au round r, qui se traduit alors par $q \notin HO(p,r)$, et celui dans lequel p reçoit de q le message $\langle null \rangle$. Dans le premier cas, le transfert d'information de q à p, ou, en d'autres termes, la communication de q vers p, est "physiquement" impossible au round r ; dans le deuxième cas, le processus q a la possibilité de communiquer avec p d'un point de vue opérationnel, mais n'a pas d'information à lui transmettre. L'ensemble $HO(p,r)$ représente ainsi l'ensemble des processus desquels p a la capacité physique de recevoir de l'information au cours du round r.

Dans tout calcul, nous ne considérons que de possibles erreurs de transmission, et en aucun cas des erreurs de transition d'états. En d'autres termes, pour tout processus p et pour tout round r, la fonction T_p^r est respectée.

Pour chaque processus p de Π, et pour chaque round r, nous introduisons deux sous-ensembles de Π :

- L'ensemble d'écoute, noté $HO(p,r)$, qui est le support de $\vec{\mu}_p^r$:

$$HO(p,r) = \{q \in \Pi \ : \ \vec{\mu}_p^r[q] \text{ est défini}\}.$$

- L'ensemble d'écoute sûr, noté $SHO(p,r)$, et défini par

$$SHO(p,r) = \{q \in \Pi \ : \ \vec{\mu}_p^r[q] = S_q^r(s_q, p)\},$$

  où s_q est l'état du processus q au début du round r.

Par définition, on a

$$\forall r > 0, \forall p,\ SHO(p,r) \subseteq HO(p,r).$$

Ces deux ensembles permettent de caractériser la différence qui existe éventuellement entre ce qui *aurait dû être envoyé* au round r et ce *qui est effectivement reçu* à ce round. Comme annoncé dans l'introduction, aucune hypothèse n'est avancée quant aux raisons pour lesquelles, à un round donné r, $\vec{\mu}_p^r[q] \neq S_q^r(p, s_q)$. Nous ne considérons que l'effet des erreurs de transmission sur le calcul, sans chercher à en déterminer la cause : l'erreur peut être due à un non-respect de S_q^r par q, à une réception incorrecte par p, ou encore à une corruption du message par le lien de q à p.

Alors qu'au round r, tout processus p est capable de déterminer $HO(p,r)$, nous supposons que son ensemble $SHO(p,r)$ lui est inaccessible. En effet, les erreurs de transmission par valeurs ne sont, en général, pas détectables par les processus destinataires des messages corrompus. Dans le cas contraire, on peut se ramener au cas des erreurs bénignes en forçant tous les processus à ne considérer que les messages provenant des processus appartenant à leur ensemble d'écoute sûr.

A partir des ensembles $HO(p,r)$ et $SHO(p,r)$, nous formons l'*ensemble d'écoute altéré*, noté $AHO(p,r)$, de la manière suivante :

$$AHO(p,r) = HO(p,r) \setminus SHO(p,r).$$

L'ensemble $AHO(p,r)$ représente l'ensemble des processus q de Π desquels, au round r, p reçoit un message ($q \in HO(p,r)$) mais celui-ci ne correspond pas à celui que q aurait dû lui envoyer à ce round ($q \notin SHO(p,r)$).

Etant donné un calcul C , nous considérons la collection indexée par $\Pi \times \mathbb{N}$

$$\big(HO(p,r)\ ;\ SHO(p,r)\big)_{p\in\Pi, r>0}$$

induite par C et nous définissons, pour chaque round r de C, le *noyau* (resp. *noyau sûr*) :

$$K(r) = \bigcap_{p\in\Pi} HO(p,r), \qquad\qquad SK(r) = \bigcap_{p\in\Pi} SHO(p,r)$$

Intuitivement, le noyau d'un round correspond à l'ensemble des processus qui sont entendus par tous à ce round, alors que le noyau sûr désigne l'ensemble des processus dont les messages sont reçus correctement par tous les

processus. Nous pouvons généraliser cette définition à l'ensemble du calcul :

$$K = \bigcap_{r>0} K(r) \qquad\qquad SK = \bigcap_{r>0} SK(r)$$

Qu'ils soient restreints à un round particulier ou globaux à tout le calcul, nous omettons la distinction entre noyau et noyau sûr lorsqu'il n'y a pas de confusion possible, comme, par exemple, dans le cas des erreurs de transmissions bénignes où les ensembles d'écoute et les ensembles d'écoute sûrs sont égaux.

De manière similaire, la *partie altérée* d'un round (resp. d'un calcul) représente l'ensemble des processus qui ont envoyé un message ayant été reçu corrompu *par un processus quelconque* à ce round (resp. dans tout le calcul) :

$$AS(r) = \bigcup_{p\in\Pi} AHO(p,r) \qquad\qquad AS = \bigcup_{r>0} AS(r)$$

On notera que dans le cas des erreurs de transmission bénignes, ces ensembles sont toujours vides.

1.1.2 Prédicats de communication

Un *prédicat de communication* est un prédicat sur les collections indexées par $\Pi \times \mathbb{N}$

$$\big(HO(p,r); SHO(p,r)\big)_{p\in\Pi, r>0},$$

qui n'est pas le prédicat constant "faux".

Nous ne manipulerons que des prédicats invariants par rapport au temps, c'est-à-dire qui ont la même valeur de vérité pour toute collection

$$\big(HO(p,r+i); SHO(p,r+i)\big)_{p\in\Pi, r>0},$$

où i est un entier naturel quelconque. Cette hypothèse d'invariance par rapport au temps résulte du fait que nous ne voulons pas que la correction d'un algorithme, décrit dans ce modèle, dépende de l'instant auquel son exécution débute.

1.2 Machines HO et problèmes

Nous définissons à présent les notions de *machine HO*, de *problème*, et de *résolubilité* d'un problème par une machine HO.

1.2.1 Machines HO

Une machine *Heard-Of* (on parlera dans la suite indifféremment de machine HO ou tout simplement de machine) pour un ensemble Π est une paire $M = \big(\mathcal{A}, \mathcal{P}\big)$, où $\mathcal{A}$ est un algorithme sur Π et $\mathcal{P}$ est un prédicat de communication. Par exemple, nous serons amenés à considérer des machines HO avec comme prédicat de communication :

$\mathcal{P}_{ref}$	$\forall r > 0,\ \forall p \in \Pi\ :\ p \in HO(p,r)$
$\mathcal{P}_{nekrounds}$	$\forall r > 0\ :\ K(r) \neq \emptyset$
$\mathcal{P}_{sym}$	$\forall r > 0,\ \forall p, q \in \Pi^2\ :\ p \in HO(q,r)\ \wedge q \in HO(p,r)$
$\mathcal{P}_{\alpha},\ \alpha \in \mathbb{N}$	$\forall r > 0,\ \forall p \in \Pi\ :\ \lvert AHO(p,r)\rvert \leq \alpha$
$\mathcal{P}_{sp_unif}$	$\forall r > 0,\ \forall p, q \in \Pi^2\ :\ HO(p,r) = HO(q,r)$

Une *exécution* d'une machine $M = \big(\mathcal{A}, \mathcal{P}\big)$ est définie par les collections

$$\big(\sigma_p^{(0)}\big)_{p\in\Pi}\ ,\quad \big(\sigma_p^{(r)}\big)_{p\in\Pi, r>0}\ \text{ et }\ \big(HO(p,r); SHO(p,r)\big)_{p\in\Pi, r>0}$$

qui satisfont les conditions suivantes :

1. Pour tout processus p, $\sigma_p^{(0)}$ est un état initial de p, c'est-à-dire $\sigma_p^{(0)} \in init_p$,

2. pour tout entier $r > 0$ et pour tout processus p, $\sigma_p^{(r)}$ est l'état de p à la fin du round r, obtenu en appliquant T_p^r à $\sigma_p^{(r-1)}$ et au vecteur de messages reçus par p au round r, et
3. la collection $\big(HO(p,r); SHO(p,r)\big)_{r>0,p\in\Pi}$ satisfait le prédicat de communication $\mathcal{P}$.

On remarque que, dans le cas des pannes bénignes, où on ne considère pas de déviation par rapport aux fonctions d'envoi de messages et de transition, puisque ces fonctions sont déterministes, une exécution est entièrement déterminée par les deux collections

$$\big(\sigma_p^{(0)}\big)_{p\in\Pi} \quad \text{et} \quad \big(HO(p,r)\big)_{r>0,p\in\Pi}$$

Dans le cas des erreurs de transmission par valeurs, au contraire, l'appartenance d'un processus q à l'ensemble d'écoute altéré d'un processus p à un round r ne suffit pas à déterminer le message, provenant de q, reçu par p à ce round ni, par suite, l'état de p à la fin de ce round. C'est la raison pour laquelle nous considérons aussi la collection des états atteints par les processus, round après round, qui dépend du contenu sémantique des messages reçus par chacun des processus à chaque round.

1.2.2 Problème. Résolubilité d'un problème.

Comme on vient de le voir, on associe à chaque exécution d'une machine HO donnée la collection $(\sigma_p^{(r)})_{r\geq 0,p\in\Pi}$ des états (un par processus et par round) atteints par les processus au cours de cette exécution, où $\sigma_p^{(0)}$ est l'état initial de p dans cette exécution. De la même manière, pour toute variable X_p locale au processus p, on notera $X_p^{(r)}$ la valeur de X_p à la fin du round r et $X_p^{(0)}$ sa valeur initiale.

Définition 1.2.1 *Un problème Σ pour Π est un prédicat sur les collections d'états des processus.*

Définition 1.2.2 *Une machine $M = \big(\mathcal{A}, \mathcal{P}\big)$ résout un problème Σ si, pour chacune de ses exécutions, la collection d'états associée $(\sigma_p^{(r)})_{r\geq 0,p\in\Pi}$ satisfait Σ.*

Définition 1.2.3 *On dit que le problème Σ est résoluble sous $\mathcal{P}$ s'il existe un algorithme $\mathcal{A}$ tel que la machine $M = \big(\mathcal{A}, \mathcal{P}\big)$ résout Σ.*

Nous avons précisé un peu plus haut que nous ne considérions que des prédicats de communication invariants par rapport au temps. Puisque, de plus, les numéros de rounds ne sont pas partie intégrante de l'état d'un processus, la résolubilité d'un problème Σ par une machine $\big(\mathcal{A}, \mathcal{P}\big)$ ne dépend pas de l'instant auquel l'exécution de $\mathcal{A}$ démarre. Pour tout entier i, si on définit l'algorithme ${}^i\mathcal{A}$ par les fonctions d'envoi et de transition ${}^iS_p^r$ et ${}^iT_p^r$ telles que, pour tout entier r, $1 \leq r \leq i$,

$$\begin{aligned} &\forall s \in states_p,\ \forall q \in \Pi,\quad {}^iS_p^r(s;q) = null \quad \text{et} \\ &\forall s \in states_p,\ \forall \vec{\mu} \in \big(M \cup \{null\})^{\Pi}\big)^*,\quad {}^iT_p^r(s,\vec{\mu}) = s \end{aligned}$$

et, pour tout entier r, $r > i$,

$${}^iS_p^r = S_p^{r-i} \quad \text{et} \quad {}^iT_p^r = T_p^{r-i},$$

alors :

Proposition 1.2.4 *Si la machine $M = \big(\mathcal{A}, \mathcal{P}\big)$ résout le problème Σ, alors pour tout entier i la machine ${}^iM = \big({}^i\mathcal{A}, \mathcal{P}\big)$ résout Σ.*

En Tolérance aux pannes, l'un des problèmes les plus étudiés est le *Consensus*. De manière informelle, il s'agit de parvenir à ce que les processus se mettent d'accord sur une valeur unique. En effet, pour assurer la tolérance aux pannes, on a traditionnellement recours à la technique de *réplication*. Cette dernière conduit naturellement à considérer un système distribué (composé des différents réplicats) dans un contexte de pannes. Il s'agit alors de fournir une vue unique de ce système, ce qui revient à résoudre le problème du Consensus.

Plus formellement, dans ce problème, chaque processus se voit attribuer une valeur initiale appartenant à un ensemble fixé V et doit, en un temps fini, décider de façon irrévocable la valeur initiale d'un des processus. Ainsi, chacune des valeurs v de l'ensemble V correspond à un état initial s_p^v du processus p, signifiant que v est la valeur initiale de p :

$$\sigma_p^{(0)} = s_p^v.$$

Le processus p dispose aussi d'ensembles disjoints Σ_p^v d'états particuliers, appelés *états de décision*, un pour chaque valeur v de V. Un processus p se trouve dans un état de Σ_p^v s'il a décidé la valeur v. Le problème du Consensus

est alors défini par les quatre prédicats suivants :

Irrévocabilité. Lorsqu'un processus a décidé une valeur, il ne peut pas en décider une autre ultérieurement :

$$\forall p \in \Pi,\ \forall v \in V,\ \forall r > 0\ :\ (\sigma_p^{(r)} \in \Sigma_p^v) \Rightarrow (\forall r' \geq r\ :\ \sigma_p^{(r')} \in \Sigma_p^v).$$

Accord. Deux processus quelconques ne peuvent décider différemment :

$$\forall p, q \in \Pi,\ \forall v, w \in V,\ \forall r, r' > 0\ :\ (\sigma_p^{(r)} \in \Sigma_p^v \wedge \sigma_q^{(r')} \in \Sigma_q^w) \Rightarrow (v = w).$$

Validité. Toute valeur décidée par un processus est la valeur initiale d'un processus :

$$\forall p \in \Pi,\ \forall v \in V,\ \forall r > 0\ :\ (\sigma_p^{(r)} \in \Sigma_p^v) \Rightarrow (\exists q \in \Pi\ :\ \sigma_q^{(0)} = s_q^v).$$

Terminaison. Tous les processus finissent par décider :

$$\forall p \in \Pi,\ \exists v \in V,\ \exists r_p > 0\ :\ \sigma_p^{(r_p)} \in \Sigma_p^v.$$

On trouve dans la littérature classique des spécifications du Consensus dans lesquelles la clause d'Accord n'est pas uniforme, en particulier dans le cas des pannes byzantines, où certains processus sont susceptibles de se comporter de manière totalement arbitraire. En revanche, aucune des spécifications ne comporte de clause de Terminaison uniforme, comme celle que nous considérons ici.

Cette uniformité que nous requérons peut paraître trop forte pour deux raisons. D'une part, parce que la spécification résultante du Consensus peut se révéler non-résoluble sous un prédicat $\mathcal{P}$ alors que la spécification classique est résoluble dans des systèmes correspondant à $\mathcal{P}$. Charron-Bost et Schiper ont montré dans [17] que cette objection ne tient pas.

D'autre part, on peut s'interroger sur la possibilité d'implémenter, dans un système réel composé de *processeurs* sujets à des pannes crash, un algorithme HO qui assure que *tous les processus* décident. Il est, en effet, clair qu'un processeur qui a arrêté de fonctionner ne peut pas prendre de décision. Cependant, le processus HO correspondant au processeur fautif n'est alors plus entendu et n'a, par conséquent, plus aucun impact sur le reste du calcul. Ainsi, une implémentation d'une machine HO qui résout la spécification uniforme du Consensus dans un système réel sujet à des pannes crash ne

peut simplement pas implémenter la capacité physique à décider d'un processus HO qui correspond à un processeur ayant arrêté prématurément de fonctionner.

On voit ainsi que, dans les modèles traditionnels, la définition de la résolubilité du Consensus dépend fortement de la manière dont on impute les pannes aux différents composants du système, ce qui illustre comment une même spécification syntaxique peut conduire à différentes conditions sémantiques dès lors que la spécification fait référence au modèle de pannes. Dans le modèle HO, nous évitons ce problème en ne considérant que des spécifications complètement uniformes, avec les mêmes conditions pour *tous* les processus.

Enfin, d'un point de vue pratique, il est déraisonnable d'exempter un processus de prendre une décision pour la seule raison qu'il a été rendu responsable d'une erreur de transmission transitoire, ou de considérer des spécifications dont la clause d'Accord n'est pas uniforme si l'on ne considère que des déviations par rapport aux fonctions d'envoi de messages.

1.2.3 Machines HO coordonnées

De nombreux algorithmes de Consensus utilisent un ou plusieurs processus distingués, communément appelés *coordinateurs*, dont le rôle consiste à collecter des informations transmises par les autres processus du système, puis à effectuer un arbitrage sur les informations reçues (la nature de cet arbitrage est spécifiée par le code de l'algorithme), et à diffuser le résultat de cet arbitrage au sein du système ; parmi les plus connus, on trouve des algorithmes proposés par Dwork, Lynch et Stockmeyer [19], par Chandra et Toueg [11], ou encore Lamport avec l'algorithme Paxos [28].

La correction de ces algorithmes est garantie par certaines propriétés sur les coordinateurs : par exemple, la Terminaison de Paxos dans le cas des pannes bénignes nécessite que le coordinateur d'une des phases soit entendu par tous à cette phase. Pour de tels algorithmes, nous introduisons les *machines HO coordonnées* (indistinctement dénommées machines coordonnées ou machines CHO dans la suite) dans lesquelles 1°) les algorithmes font référence à ces coordinateurs, et 2°) les prédicats ne concernent plus uniquement les ensembles d'écoute, mais concernent aussi les coordinateurs.

Nous introduisons ici la notion d'*algorithme coordonné* sur un ensemble

Π. La définition d'un algorithme coordonné est semblable à celle d'un algorithme, excepté le fait que, pour tout entier $r > 0$ et pour tout processus p, on associe à p son *coordinateur au round* r, que l'on note $Coord(p, r)$. Les fonctions d'envoi et de transition sont alors définies de la manière suivante :

$$S_p^r : \Pi \times states_p \times \Pi \longrightarrow M \cup \{null\},$$

et

$$T_p^r : \Pi \times states_p \times \big((M \cup \{null\})^{\Pi}\big)^* \longrightarrow states_p,$$

où $\big((M \cup \{null\})^{\Pi}\big)^*$ représente l'ensemble des vecteurs partiels, indexés par Π, d'éléments de $M \cup \{null\}$. Les fonctions $(S_p^r)_{r>0}$ et $(T_p^r)_{r>0}$ définissent le *processus coordonné* p, et la collection des processus coordonnés est appelée *algorithme coordonné* sur Π.

À chaque round r, chaque processus p envoie à chaque processus q le message $S_p^r(Coord(p, r), s, q))$, où s est l'état de p au début du round r, puis troque son état courant contre le nouvel état $T_p^r(Coord(p, r), s, \vec{\mu}_p^r)$, où $\vec{\mu}_p^r$ est le vecteur partiel des messages reçus par p au round r.

Nous dirons qu'un round r est *uniformément coordonné* si

$$\forall\ p, q \in \Pi\ :\ Coord(p, r)\ =\ Coord(q, r)$$

et que r est *bien coordonné* si

$$\forall\ p \in \Pi\ :\ Coord(p, r) \in HO(p, r).$$

Un calcul est *uniformément coordonné à partir du round* r_0 si tout round r tel que $r \geq r_0$ est uniformément coordonné ; un calcul est *uniformément coordonné* s'il est uniformément coordonné à partir du premier round.

Un *prédicat de communication-coordinateur* est un prédicat sur les collections indexées par $\Pi \times \mathbb{N}$

$$\big(HO(p, r); SHO(p, r); Coord(p, r)\big)_{p \in \Pi, r > 0}$$

comme,par exemple,

$$\mathcal{P}_{coord_correct} :: \left| \begin{array}{l} \forall r > 0,\ \forall p, q \in \Pi^2,\ \ Coord(p, r) = Coord(q, r)\ \ \text{et} \\ \forall r > 0,\ \forall p \in \Pi,\ \ Coord(p, r) \notin AS(r) \end{array} \right.$$

Une *machine HO coordonnée* pour un ensemble Π est une paire $\big(\mathcal{A}, \mathcal{P}\big)$, où $\mathcal{A}$ est un algorithme coordonné sur Π et $\mathcal{P}$ est un prédicat de communication-coordinateur. Comme pour les machines HO ordinaires, nous ne considérons que des prédicats de communication-coordinateur invariants par rapport au temps. La notion de résolubilité d'un problème par une machine ne change pas.

De même que pour les ensembles $HO(p, r)$, nous ne spécifions pas la façon dont les processus déterminent leurs coordinateurs : cela peut être le résultat d'un calcul (la machine coordonnée est simulée à l'aide d'une machine ordinaire), ou alors les processus ont accès à des ressources extérieures au système (ressources physiques ou oracles) qui sont en mesure de fournir à chaque processus le nom de son coordinateur pour un round donné.

L'une des stratégies les plus communément employées est celle qui est dite *"du coordinateur tournant"*. Elle consiste à sélectionner, pour tout processus p de $\Pi = \{p_1, \cdots, p_n\}$, et pour tout round $r > 0$:

$$Coord(p, r) = p_{(1+r) \bmod n}.$$

Avec cette stratégie, puisqu'il n'y a pas de déviation par rapport aux fonctions de transition, l'accord sur l'identité du coordinateur d'un round donné est trivialement garanti. En d'autres termes, tous les rounds d'une exécution d'une machine coordonnée pour laquelle la stratégie du coordinateur tournant est adoptée sont uniformément coordonnés.

1.3 Translations dans le cas bénin

Nous nous intéresserons, dans le Chapitre 2, à un grand nombre de systèmes classiques et aux prédicats de communication que chacun peut garantir. Nous donnerons une nouvelle preuve d'équivalence de deux de ces systèmes, dans le cas des pannes bénignes, fondée sur la notion de *simulation* d'un prédicat de communication $\mathcal{P}'$ par une machine $M = \big(\mathcal{A}, \mathcal{P}\big)$. Nous utiliserons, pour cela, la notion de *translation d'un prédicat $\mathcal{P}$ en un prédicat $\mathcal{P}'$*, initialement définie par Charron-Bost et Schiper dans [17].

1.3.1 Translations uniformes

Soit k un entier positif quelconque, et soit $\mathcal{A}$ un algorithme sur Π tel que chaque processus p met à jour une variable $NewHO_p$ qui contient un sous-ensemble de Π. On appelle *macro-round* ρ la suite des k rounds consécutifs $k(\rho-1)+1, \cdots, k\rho$. On note $NewHO_p^{(\rho)}$ la valeur de la variable $NewHO_p$ à la fin du macro-round ρ.

On dit alors que la machine $M = \big(\mathcal{A}, \mathcal{P}\big)$ *simule* le prédicat de communication $\mathcal{P}'$ en k rounds si, pour tout exécution de M, on a :

E1 : Si un processus q appartient à $NewHO_p^{(\rho)}$, alors il existe un entier l dans $\{1, \cdots, k\}$, une chaîne de $l+1$ processus $p_0, \cdots, p_l$ de $p_0 = q$ à $p_l = p$ et une sous-suite de l rounds $r_1, \cdots, r_l$, avec $r_1 < r_2 < \cdots < r_l$, du macro-round ρ tels que pour tout indice i, $1 \leq i \leq l$, on a :

$$p_{i-1} \in HO(p_i, r_i),$$

E2 : La collection $\big(NewHO_p^{(\rho)}\big)_{p\in\Pi, r>0}$ satisfait le prédicat $\mathcal{P}'$.

La condition E1 impose qu'un processus q qui appartient à l'ensemble $NewHO_p$ à la fin du macro-round ρ a effectivement été entendu, directement ou par l'intermédiaire d'autres processus, par le processus p. On rappelle que l'ensemble d'écoute d'un processus p, à un round donné, consiste en l'ensemble des processus desquels p a la capacité physique de recevoir de l'information à ce round. Il est donc crucial que les ensembles d'écoute simulés vérifient, eux aussi, cette propriété opérationnelle, ce qui ne serait pas nécessairement le cas si on autorisait les processus à les déterminer arbitrairement.

La condition E2, quant à elle, impose que les ensembles $NewHO_p$ simulent des ensembles d'écoute qui satisfont $\mathcal{P}'$.

S'il existe un algorithme $\mathcal{A}$ tel que la machine $M = \big(\mathcal{A}, \mathcal{P}\big)$ simule $\mathcal{P}'$ en k rounds, on dira que $\mathcal{A}$ *translate* $\mathcal{P}$ en $\mathcal{P}'$ *en k rounds*, ou encore que $\mathcal{A}$ est une *translation en k rounds* de $\mathcal{P}$ à $\mathcal{P}'$, et on notera $\mathcal{P} \succeq_k \mathcal{P}'$.

On peut remarquer que dans le cas où $\mathcal{P} \Rightarrow \mathcal{P}'$, l'algorithme qui impose à chaque processus p d'écrire la valeur de $HO(p, r)$ dans $NewHO_p$ à la fin de chaque round r est une translation en 1 round de $\mathcal{P}$ à $\mathcal{P}'$, et donc que $\mathcal{P} \succeq_1 \mathcal{P}'$.

1.3.2 Translations générales

On généralise à présent la définition précédente au cas des translations qui ne prennent pas un nombre constant de rounds, en ce sens que deux macro-rounds distincts peuvent ne pas être composés du même nombre de rounds, ou encore qu'un macro-round donné peut être composé d'un certain nombre de rounds pour un processus p et d'un nombre différent de rounds pour un processus q. Bien entendu, les deux notions ne sont pas incompatibles.

Chaque processus p met à jour une variable supplémentaire $MacroRound_p$ initialisée à 0. Lorsque p met à jour sa variable $NewHO_p$, il incrémente $MacroRound_p$ de 1. Lorsque p envoie un message, il l'étiquette avec la valeur courante de $MacroRound_p$. De plus, p ignore tout message étiqueté avec une autre valeur que la valeur courante de $MacroRound_p$.

On obtient une définition générale de la notion de translation si on reformule la condition E1 comme suit :

E1 : Si un processus q appartient à $NewHO_p(\rho)$, alors il existe une chaîne de processus $p_0, p_1, \cdots, p_l$ de $p_0 = q$ à $p_l = p$ et une suite de l rounds $r_1, \cdots, r_l$, avec $r_1 < r_2 < \cdots < r_l$, telles que pour tout indice i, $1 \leq i \leq l$:

$$MacroRound_{p_i}^{(r_i)} = \rho, \text{ et } p_{i-1} \in HO(p_i, r_i).$$

S'il existe un algorithme $\mathcal{A}$ tel que la machine $M = \big(\mathcal{A}, \mathcal{P}\big)$ simule le prédicat $\mathcal{P}'$, on dira que $\mathcal{A}$ *translate* $\mathcal{P}$ en $\mathcal{P}'$ et on notera $\mathcal{P} \succeq \mathcal{P}'$. De manière évidente, la relation $\succeq$ contient toutes les relations $\succeq_k$.

Etant donné une simulation d'un prédicat $\mathcal{P}'$ par une machine $\big(\mathcal{A}, \mathcal{P}\big)$, tout problème résoluble sous $\mathcal{P}'$ l'est aussi sous $\mathcal{P}$ (cf. [17]). Nous verrons dans le chapitre suivant que, si on peut implémenter $\mathcal{P}$ grâce à un système classique S, alors on peut aussi implémenter $\mathcal{P}'$ grâce à S.

La relation $\succeq$ est transitive et permet donc d'ordonner les prédicats en fonction de leur capacité à résoudre des problèmes ou à capturer les propriétés de systèmes classiques. Si on a à la fois $\mathcal{P} \succeq \mathcal{P}'$ et $\mathcal{P}' \succeq \mathcal{P}$, on dira que $\mathcal{P}$ et $\mathcal{P}'$ sont *équivalents* et on notera $\mathcal{P} \simeq \mathcal{P}'$.

Chapitre 2

Systèmes classiques et prédicats de communication

Comme nous avons pu le voir dans le chapitre précédent, le modèle HO se situe à un haut niveau d'abstraction. Il est donc important d'étudier quels prédicats de communication peuvent être implémentés dans quels systèmes, et de tenter de déterminer quel prédicat capture le mieux les propriétés des communications d'un système donné. D'autre part, en s'inspirant de l'approche de Charron-Bost et Schiper [17], on peut se poser la question de savoir quel prédicat de communication caractérise un problème particulier. Étant donné un problème Σ, on veut identifier les prédicats des machines HO qui permettent de résoudre Σ.

La première partie du présent chapitre est consacrée à l'analyse d'une large gamme de systèmes classiques et des prédicats de communication qui, selon nous, leur correspondent, en ce sens que chacun d'eux est le plus fort prédicat implémentable dans le système considéré. Cette notion de *correspondance* que nous utilisons est tout à fait informelle : elle reflète l'intuition qu'on peut avoir sur les correspondances "naturelles" entre systèmes et prédicats de communication.

Nous reviendrons, par la suite, sur le résultat de [17] concernant les prédicats de communication des machines HO qui résolvent Consensus, et le mettrons en relation avec les prédicats présentés.

2.1 Mémoire partagée dans le cas bénin

Dans un système à *mémoire partagée*, les processus communiquent *via* des *objets partagés*, ou encore *variables partagées*. Une des classes majeures d'objets partagés est celle des objets *Read-Modify-Write* (RMW) qui, à notre connaissance, ont été introduits dans leur forme générale par Kruskal *et al.* [26].

2.1.1 Registres SWMR et objets Atomic-Snapshot

Nous nous concentrons, tout d'abord, sur deux types particuliers d'objets RMW : 1°) les *registres atomiques Single-Writer/Multi-Reader* (SWMR) et 2°) les *objets Atomic-Snapshot*. Il a été montré ([4, 20, 3]) que ces deux types d'objets étaient équivalents dans le contexte des pannes bénignes, en ce sens que l'on peut implémenter les uns dans un système disposant des autres. Nous discutons la caractérisation des systèmes SWMR donnée par Gafni [22] et présentons, pour chacun de ces systèmes, un prédicat de communication qui lui correspond naturellement.

Dans un système SWMR, chaque processus p d'un ensemble Π dispose d'une infinité de registres R_p sur lesquels peuvent être exécutées deux opérations (1) $Write(R_p, v)$, qui est l'écriture dans R_p d'une valeur v appartenant à un ensemble fixé V, et (2) $Read(R_p)$, qui est la lecture du contenu de R_p. Chaque processus $p \in \Pi$ peut lire dans tous les registres R_q, mais ne peut écrire que dans R_p. Nous considérons des registres *atomiques* au sens de la définition donnée par Lamport dans [27], c'est-à-dire pour lesquels les opérations *Read* et *Write* peuvent se chevaucher[1] mais s'exécutent comme si elles se produisaient de façon instantanée. En d'autres termes, toute exécution du système est équivalente à une exécution dans laquelle les opérations sur chaque registre sont séquentielles et les valeurs retournées par les lectures sont en accord avec la spécification des registres.

Un objet Atomic-Snapshot est composé de n registres, un par processus. Chaque processus peut, soit écrire une valeur dans la composante de l'objet qui lui a été attribuée - on dira alors qu'il *met à jour l'objet*[2]-, soit obtenir

[1] On peut, par exemple, citer le cas de figure dans lequel un processus p commence à lire la valeur d'un registre R_q alors que q est en train d'y écrire.

[2] Puisque chaque processus ne peut mettre à jour que sa propre composante dans un objet Atomic-Snapshot, on omet de préciser qu'un processus met à jour *sa composante* de

l'ensemble des valeurs des composantes de l'objet - on dira alors qu'il *scanne* l'ojet. Comme indiqué par son nom, l'accès à un objet Atomic-Snapshot par un processus est atomique, qu'il s'agisse d'une mise à jour ou d'un scan : la notion d'atomicité est la même que celle qui est donnée pour les registres SWMR, c'est à dire que les opérations sur l'objet (mise à jour ou scan) peuvent se chevaucher, mais s'exécutent comme si elles se produisaient de façon instantanée.

Dans un système Atomic-Snapshot, les processus ont accès à une infinité d'objets Atomic-Snapshot.

Quels prédicats pour ces objets ?

Comme nous l'avons précisé dans l'introduction, Gafni a défini dans [22] un modèle de calcul en rounds, appelé *Round-by-Round Fault Detector* (RRFD), pour l'étude des systèmes distribués. Dans ce modèle, les calculs s'effectuent en rounds successifs, et les propriétés des communications entre processus, que ce soit *via* des objets partagés ou par échanges de messages, sont capturées par un *module* appelé *Round-by-Round Fault Detector module*, qui fait partie intégrante du système. A chaque round r et pour chaque processus p, le module fournit un ensemble de processus dont p ne reçoit pas de messages.

Comme annoncé dans [17], les modèles RRFD et HO sont similaires sous plusieurs aspects. D'une part, l'un comme l'autre ne spécifie pas le médium de communication. Les processus échangent des morceaux d'information, appelés *messages*, mais la manière dont les messages sont envoyés et reçus par les processus n'est pas précisée. D'autre part, tout module RRFD fournit, à chaque round r et pour chaque processus p, le complémentaire dans Π de l'ensemble $HO(p, r)$. En revanche, les modèles RRFD et HO diffèrent sur la manière d'interpréter les prédicats : dans le modèle RRFD, le médium de communication est supposé fiable ; le module RRFD fournit donc, à chaque round, un ensemble de processus possiblement fautifs. Dans le modèle HO, au contraire, la notion de composant fautif est inexistante et les prédicats représentent uniquement des propriétés vérifiées par les communications.

Gafni a étudié de manière informelle la capacité du modèle RRFD à couvrir un grand nombre de systèmes distribués. En particulier, il a introduit deux modules RRFD qu'il estime "correspondre naturellement" aux systèmes Atomic-Snapshot et SWMR, respectivement. Nous analysons ici ses résultats.

---

l'objet

Algorithme 2.1 Algorithme *AtSn* : code du processus p

```
1:  Initialisation
2:    r_p ∈ ℕ ; initialement 1
3:    v_p ∈ V, initialement v_p = id_p, où id_p est l'identifiant du processus p
4:    HO_p ⊆ Π ; initialement ∅

5:  Pour tout r_p > 0 :
6:    HO_p := ∅
7:    Write(AS^{r_p}[p], v_p)
8:    Scan(AS^{r_p})
9:    HO_p := {ensemble des valeurs distinctes de ⊥ retournées par Scan(AS^{r_p})}
10:     r_p := r_p + 1
```

En ce qui concerne les systèmes Atomic-Snapshot sujets à des pannes bénignes, la nature même d'un objet Atomic-Snapshot le conduit à considérer le prédicat suivant, que nous notons $\mathcal{P}^*_{\mathcal{RD}}$ (pour *Russian Dolls*) :

$$\mathcal{P}^*_{\mathcal{RD}} = \mathcal{P}_{\mathcal{RD}} \ \wedge \ \mathcal{P}_{ref},$$

où $\mathcal{P}_{\mathcal{RD}}$ assure qu'à chaque round les ensembles d'écoute sont totalement ordonnés par l'inclusion :

$$\mathcal{P}_{\mathcal{RD}} \ :: \ \forall r > 0, \ \forall p, q \in \Pi \ \ : \ \ HO(p,r) \subseteq HO(q,r) \ \vee \ HO(q,r) \subseteq HO(p,r)$$

et $\mathcal{P}_{ref}$ garantit qu'à chaque round, chaque processus entend parler de lui-même :

$$\mathcal{P}_{ref} \ :: \ \forall r > 0, \ \forall p \in \Pi \ \ : \ \ p \in HO(p,r).$$

Pour se convaincre qu'il est possible de garantir $\mathcal{P}^*_{\mathcal{RD}}$ dans un système Atomic-Snapshot en présence de pannes bénignes, considérons l'algorithme *AtSn*, donné comme Algorithme 2.1, censé être exécuté dans un système où, pour tout entier $r > 0$, tous les processus de Π ont accès à un objet Atomic-Snapshot A^r dont toutes les composantes sont initialisées avec la valeur $\perp$. Nous décomposons les exécutions de *AtSn* en rounds. Pour tout entier $r > 0$, chaque processus p met à jour A^r avec son identifiant, puis scanne A^r. Il met ensuite à jour une variable HO_p en lui affectant l'ensemble des processus q tels que id_q appartient à l'ensemble de valeurs retourné par $Scan(A^r)$.

Proposition 2.1.1 *Il est possible de garantir $\mathcal{P}^*_{\mathcal{RD}}$ dans un système Atomic-Snapshot en présence de pannes bénignes.*

Démonstration: Soit S un système Atomic-Snapshot. Considérons une exécution quelconque de $AtSn$ dans S.

Soit $r > 0$ un entier. Supposons qu'il existe un processus p qui met à jour sa variable HO_p au round r. Le code implique que p effectue $Scan(A^r)$. Puisque p ne scanne A^r qu'après y avoir écrit, il lit nécessairement dans sa composante une valeur différente de $\perp$. Par la ligne 9, on a $p \in HO_p$.

Supposons qu'il existe un processus q, distinct de p, qui met à jour sa variable HO_q au round r. L'atomicité de A^r implique que l'un des deux, disons p, effectue l'opération $Scan(AS^r)$ avant l'autre. Par la ligne 9, on a alors $HO_p \subseteq HO_q$. □

Pour les systèmes SWMR en présence de pannes bénignes, Gafni propose le prédicat $\mathcal{P}_{nekrounds}$ qui assure qu'à chaque round, il existe un processus entendu par tous les autres :

$$\mathcal{P}_{nekrounds} \ :: \ \forall r > 0, \ K(r) \neq \emptyset.$$

Algorithme 2.2 Algorithme $sWmR$: code du processus p

1: **Initialisation**
2: $r_p \in \mathbb{N}$; initialement 1
3: $v_p \in V$, initialement $v_p = id_p$, où id_p est l'identifiant unique du processus p
4: $HO_p \subseteq \Pi$

5: **Pour tout** $r_p > 0$
6: $HO_p := \emptyset$
7: $Write(R_p^{r_p}, v_p)$
8: Pour tout $q \in \Pi$, $Read(R_q^{r_p})$
9: $HO_p := \{q \ : \ Read(R_q^{r_p}) \neq \perp\}$
10: $r_p := r_p + 1$

Considérons l'algorithme $sWmR$, donné comme Algorithme 2.2, censé être exécuté dans un système où, pour tout entier $r > 0$, chaque processus $p \in \Pi$ dispose d'un registre atomique R_p^r initialisé avec la valeur $\perp$. L'algorithme $sWmR$ est simplement obtenu en remplaçant le scan de l'algorithme $AtSn$ (ligne 8) par une lecture séquentielle des n registres SWMR. Comme c'était le cas pour $AtSn$, nous décomposons les exécutions de $sWmR$ en rounds.

Proposition 2.1.2 *Tout système SWMR sujet à des pannes bénignes peut garantir le prédicat* $\mathcal{P}_{nekrounds}$.

Démonstration: Considérons une exécution de $sWmR$. Soit $r > 0$ un entier. Puisque les registres sont atomiques, on peut distinguer le processus qui

écrit le "premier" dans son registre R_p^r, s'il existe. On le note p. Supposons qu'il existe un processus q qui met à jour son ensemble HO_q. Puisque q ne commence à lire les registres qu'après avoir écrit dans le sien, et donc après que p a écrit dans R_p^r, on en déduit $p \in HO_q$. □

On remarque que, dans toute exécution de $sWmR$, pour tout $r > 0$, chaque processus p ne commence à lire les registres R_q^r qu'après avoir écrit dans R_p^r ; on en déduit que si p met à jour HO_p au round r, alors $p \in HO_p$. On en déduit que tout système SWMR sujet à des pannes bénignes peut garantir le prédicat $\mathcal{P}_{ref}$. Il est donc plus naturel de considérer le prédicat $\mathcal{P}^*_{nekrounds}$, conjonction des deux prédicats $\mathcal{P}_{nekrounds}$ et $\mathcal{P}_{ref}$, pour décrire les systèmes SWMR en présence de pannes bénignes.

$\mathcal{P}^*_{nekrounds}$ est trop faible pour les sytèmes SWMR.

La discussion autour de l'algorithme $sWmR$ pour les systèmes SWMR dans le cas des pannes bénignes montre qu'il est très naturel de considérer le prédicat $\mathcal{P}^*_{nekrounds}$ comme leur correspondant. Cependant, l'hypothèse d'atomicité des registres implique aussi que, dans toute exécution de $sWmR$, pour tous processus p et q, pour tout entier $r > 0$, soit p peut lire la valeur inscrite par q dans R_q^r, soit q peut lire la valeur inscrite par p dans R_p^r. Cette constatation nous conduit à considérer le prédicat suivant, lui aussi introduit par Gafni, que nous notons $\mathcal{P}_{sym}$:

$$\mathcal{P}_{sym} \ :: \quad \forall r > 0,\ \forall p, q \in \Pi \quad : \quad p \in HO(q, r) \quad \vee \quad q \in HO(p, r)$$

qui peut ainsi être garanti par tout système SWMR. On remarque que, de manière évidente, $\mathcal{P}_{sym}$ implique $\mathcal{P}_{ref}$. Nous nous proposons, ici, de démontrer que $\mathcal{P}_{sym}$ est strictement plus fort que $\mathcal{P}^*_{nekrounds}$, et donc que $\mathcal{P}^*_{nekrounds}$ est trop faible pour correspondre aux systèmes SWMR sujets à des pannes bénignes.

Nous présentons tout d'abord l'algorithme $\mathcal{STN}$, donné comme Algorithme 2.3. Nous démontrons que $\mathcal{STN}$ est une translation en 2 rounds de $\mathcal{P}_{sym}$ en $\mathcal{P}^*_{nekrounds}$, et donc que $\mathcal{P}_{sym}$ est au moins aussi fort que $\mathcal{P}^*_{nekrounds}$. Il est intéressant de remarquer que $\mathcal{STN}$ est la première translation présentée par Charron-Bost et Schiper (donnée dans [17] comme *"Algorithm 1 : translation for increasing kernels"*), qui l'utilisent pour simuler des macro-rounds dont les noyaux sont de taille plus importante que la taille des noyaux des rounds originaux.

Algorithme 2.3 Algorithme $\mathcal{STN}$: code du processus p

1: **Initialisation :**
2: $NewHO_p \in 2^{\Pi}$, initialement vide

3: **Round $r = 2\rho - 1$:**
4: S_p^r :
5: Envoyer $\langle id_p \rangle$ à tous les processsus

6: T_p^r :
7: $NewHO_p := \emptyset$

8: **Round $r = 2\rho$:**
9: S_p^r :
10: Envoyer $\langle HO(p, r-1) \rangle$ à tous les processus

11: T_p^r :
12: $NewHO_p := \bigcup_{q \in HO(p,r)} HO(q, r-1)$

Considérons une exécution de $\mathcal{STN}$. Soit ρ un macro-round et soit $A_\rho = (a_{i,j})_{i,j \in [\![1;n]\!]}$, $B_\rho = (b_{i,j})_{i,j \in [\![1;n]\!]}$ et $M_\rho = (m_{i,j})_{i,j \in [\![1;n]\!]}$ les matrices définies par :

* $a_{i,j} = 1$ si $p_i \in HO(p_j, 2\rho - 1)$, 0 sinon

* $b_{i,j} = 1$ si $p_i \in HO(p_j, 2\rho)$, 0 sinon

* $M_\rho = A_\rho \cdot B_\rho$

On note que, pour tous $i, j \in [\![1;n]\!]$, on a $m_{i,j} > 0$ si et seulement si le processus p_i appartient à $NewHO_{p_j}^{(\rho)}$. Par conséquent, pour démontrer que le noyau du macro-round simulé est non-vide, il suffit de vérifier qu'il existe un indice $i \in [\![1;n]\!]$ tel que, pour tout indice $j \in [\![1;n]\!]$, on ait $m_{i,j} > 0$. De la même manière, pour démontrer que tout processus p_i appartient à $NewHO_{p_i}^{(\rho)}$, il suffit de démontrer que, pour tout indice $i \in [\![1;n]\!]$, on a $m_{i,i} > 0$.

Lemme 2.1.3 *Soit n un entier tel que $n \geq 2$. Soit $A = (a_{i,j})_{i,j \in [\![1;n]\!]}$ et $B = (b_{i,j})_{i,j \in [\![1;n]\!]}$ deux matrices dans $\mathcal{M}_{n \times n}(\{0,1\})$.*

Si A et B satisfont

- $\forall i, j \in [\![1;n]\!],\ a_{i,j} + a_{j,i} > 0$

- $\forall i, j \in [\![1;n]\!],\ b_{i,j} + b_{j,i} > 0$

et si la matrice $M = (m_{i,j})_{i,j\in[\![1;n]\!]}$ *est définie par* $M = A \times B$*, alors* M *satisfait :*

$$\exists i \in [\![1;n]\!] : \ \forall j \in [\![1;n]\!], \ m_{i,j} > 0, \quad et \tag{2.1}$$
$$\forall i \in [\![1;n]\!], \quad m_{i,i} > 0. \tag{2.2}$$

Démonstration: Nous commençons par démontrer (2.1), par récurrence sur la taille n, $n \geq 2$, des matrices A et B.

• *Cas de base* $n = 2$.

Soit $A = (a_{i,j})_{i,j\in\{1,2\}}$ et $B = (b_{i,j})_{i,j\in\{1,2\}}$ dans $\mathcal{M}_{2\times 2}(\{0,1\})$.
Si A satisfait

$$\forall i, j \in \{1,2\}, \ a_{i,j} + a_{j,i} > 0,$$

alors il existe un indice $i_0 \in \{1,2\}$ tel que $a_{i_0,1} > 0$ et $a_{i_0,2} > 0$.

Si B satisfait $\forall i, j \in \{1,2\}, \ b_{i,j} + b_{j,i} > 0$, alors on a $b_{1,1} > 0$ et $b_{2,2} > 0$.
Par conséquent, si $M = A \times B$ alors $m_{i_0,1} > 0$ et $m_{i_0,2} > 0$.

• *Etape de récurrence* $n > 2$.

Supposons à présent que le résultat soit vérifié pour des matrices de taille $n-1$.
Soit $A = (a_{i,j})_{i,j\in[\![1;n]\!]}$ et $B = (b_{i,j})_{i,j\in[\![1;n]\!]}$ deux matrices dans $\mathcal{M}_{n\times n}(\{0,1\})$ qui satisfont

- $\forall i, j \in [\![1;n]\!], \ a_{i,j} + a_{j,i} > 0$

- $\forall i, j \in [\![1;n]\!], \ b_{i,j} + b_{j,i} > 0$

Soit $M = (m_{i,j})_{i,j\in[\![1;n]\!]}$ la matrice définie par $M = A \times B$. Raisonnons par contradiction et supposons que M satisfasse

$$\forall i \in [\![1;n]\!], \ \exists j \in [\![1;n]\!] : \ m_{i,j} = 0$$

Pour tout $k = 1, \cdots, n$, soit A_k, B_k dans $\mathcal{M}_{(n-1)\times(n-1)}(\{0,1\})$ et M_k dans $\mathcal{M}_{(n-1)\times(n-1)}(\mathbb{N})$ les matrices définies par

$$A_k = (a_{i,j})_{i,j\neq k}, \ B_k = (b_{i,j})_{i,j\neq k} \text{ et } M_k = A_k \times B_k.$$

Notons que pour tout $k = 1, \cdots, n$, A_k et B_k satisfont

- $\forall i, j \neq k,\ a_{i,j} + a_{j,i} > 0$

- $\forall i, j \neq k,\ b_{i,j} + b_{j,i} > 0$

Considérons A_1 et B_1. L'hypothèse de récurrence implique qu'il existe un indice $i_1 \neq 1$ tel que, pour tout indice $j = 2, \cdots, n$, on a

$$\sum_{l=2}^{n} a_{i_1,l} \cdot b_{l,j} > 0.$$

De plus, pour tout indice $j = 2, \cdots, n$,

$$m_{i_1,j} = \sum_{l=1}^{n} a_{i_1,l} \cdot b_{l,j}.$$

On obtient ainsi que, pour tout indice $j = 2, \cdots, n$:

$$m_{i_1,j} \geq \sum_{l=2}^{n} a_{i_1,l} \cdot b_{l,j} \quad > 0.$$

Puisque nous avons supposé que

$$\forall i \in [\![1; n]\!],\ \exists j \in [\![1; n]\!] : \quad m_{i,j} = 0,$$

on en conclut $m_{i_1,1} = 0$.

Considérons à présent A_2 et B_2. Par le même argument, nous montrons qu'il existe un indice $i_2 \neq 2$ tel que $m_{i_2,2} = 0$ et $\forall j \neq 2$, $m_{i_2,j} > 0$. On vient de montrer $m_{i_1,1} = 0$; on en conclut donc $i_2 \neq i_1$.

En répétant le même argument pour A_k et B_k, $k = 3, \cdots, n$, on démontre que

- pour tout $k = 1, \cdots, n$, il existe un indice $i_k \in [\![1; n]\!]$ tel que

$$m_{i_k,k} = 0 \text{ et } \forall l \in \{1, \cdots, n\} \setminus \{k\},\ m_{i_k,l} > 0$$

- $\forall l, k \in [\![1; n]\!] \quad k \neq l \Rightarrow i_k \neq i_l$.

Nous pouvons ainsi définir une application $h : [\![1;n]\!] \longrightarrow [\![1;n]\!]$ telle que, pour tout indice $j \in [\![1;n]\!]$, on a $h(j) = i_j$. Par construction, l'application h est bijective.

Soit maintenant $k \in [\![1;n]\!]$. Puisque $h : j \to h(j)$ est bijective, nous avons

$$m_{h(k),h(h^{-1}(k))} = 0.$$

Le fait que $b_{h(h^{-1}(k)),h(h^{-1}(k))} > 0$ implique alors $a_{h(k),h(h^{-1}(k))} = 0$.

De plus, pour tous $i, j \in [\![1;n]\!]^2$, on a $a_{i,j} + a_{j,i} > 0$, et donc $a_{h(h^{-1}(k)),h(k)} > 0$.

Par définition de l'application h, on a $m_{h(h^{-1}(k)),h^{-1}(k)} = 0$, qui est équivalent à $a_{h(h^{-1}(k)),h(k)} \cdot b_{h(k),h^{-1}(k)} = 0$.

On en déduit $b_{h(k),h^{-1}(k)} = 0$, ce qui, par définition de B, implique

$$b_{h^{-1}(k),h(k)} > 0.$$

Par le même argument on obtient $a_{h(h(k)),h^{-1}(k)} = 0$, et donc $a_{h^{-1}(k),h(h(k))} > 0$.

Par conséquent, on a $b_{h(h(k)),h^{-2}(k)} = 0$, ce qui implique $b_{h^{-2}(k),h(h(k))} > 0$, et donc $a_{h^3(k),h^{-2}(k)} = 0$.

En généralisant, on obtient

$$\forall l, \quad a_{h^{-l}(k),h^{l+1}(k)} > 0, \tag{2.3}$$

où h^r est la r-ième itérée, pour la composition, de h.

Puisque $h : [\![1;n]\!] \to [\![1;n]\!]$ est bijective, il existe un entier $l_k \geq 1$ tel que $h^{l_k}(k) = k$. En posant $l_k = l + 1$ dans l'équation (3.23), on obtient $a_{i(k),k} > 0$, une contradiction. On en conclut donc que M satisfait bien la condition (2.1).

Pour démontrer que M satisfait aussi le condition (2.2), il suffit de remarquer que, pour tout indice i, $i \in [\![1;n]\!]$, on a

$$a_{i,i} > 0 \quad \text{et}$$
$$b_{i,i} > 0.$$

Or, par définition de M, on a, pour tout indice i, $i \in [\![1;n]\!]$,

$$m_{i,i} \geq a_{i,i} + b_{i,i}.$$

On en conclut donc que M satisfait bien (2.2).

□

Nous sommes à présent en mesure d'énoncer le résultat annoncé :

Proposition 2.1.4 *L'algorithme $\mathcal{STN}$ est une translation en 2 rounds de $\mathcal{P}_{sym}$ en $\mathcal{P}^*_{nekrounds}$, et donc*

$$\mathcal{P}_{sym} \succeq_2 \mathcal{P}^*_{nekrounds}.$$

Démonstration: Soit une exécution de la machine $(\mathcal{STN}, \mathcal{P}_{sym})$.

Le fait que la condition E1 de la définition d'une translation est bien vérifiée est une conséquence directe de la manière dont $\mathcal{STN}$ fonctionne. En effet, pour tout macro-round $\rho > 0$, le code (ligne 12) assure que si un processus q appartient à $NewHO_p^{(\rho)}$, alors p a effectivement entendu q, peut-être par l'intermédiaire d'un autre processus, au cours du macro-round ρ.

Il nous reste à montrer que la condition E2 est, elle aussi, vérifiée, c'est-à-dire que la collection $\left(NewHO_p^{(\rho)}\right)_{p\in\Pi,\rho>0}$ satisfait $\mathcal{P}^*_{nekrounds}$.

Soit $\rho > 0$ un macro-round et soit $A_\rho = (a_{i,j})_{i,j\in[\![1;n]\!]}$, $B_\rho = (b_{i,j})_{i,j\in[\![1;n]\!]}$ et $M_\rho = (m_{i,j})_{i,j\in[\![1;n]\!]}$ les matrices définies par :

* $a_{i,j} = 1$ si $p_i \in HO(p_j, 2\rho - 1)$, 0 sinon
* $b_{i,j} = 1$ si $p_i \in HO(p_j, 2\rho)$, 0 sinon
* $M_\rho = A_\rho \cdot B_\rho$

Comme précisé plus haut, pour tous $i, j \in [\![1;n]\!]^2$, on a $m_{i,j} > 0$ si et seulement si le processus p_i appartient à $NewHO_{p_j}^{(\rho)}$. Ainsi, la condition E2 est satisfaite si et seulement si M_ρ vérifie

$$\exists i \in [\![1;n]\!],\ \forall j \in [\![1;n]\!],\ m_{i,j} > 0 \ \text{ et} \tag{2.4}$$

$$\forall i \in [\![1;n]\!],\ \ m_{i,i} > 0. \tag{2.5}$$

Comme $\mathcal{P}_{sym}$ est satisfait, les matrices A_ρ and B_ρ vérifient

- $\forall i, j \in [\![1; n]\!],\ a_{i,j} + a_{j,i} > 0,$

- $\forall i, j \in [\![1; n]\!],\ b_{i,j} + b_{j,i} > 0.$

Le Lemme 2.1.3 assure alors que M_ρ vérifie (2.4) et (2.5) ; par conséquent, la condition E2 est satisfaite, ce qui achève de démontrer que l'algorithme $\mathcal{STN}$ est une translation en 2 rounds de $\mathcal{P}_{sym}$ en $\mathcal{P}^*_{nekrounds}$. □

Nous démontrons à présent que $\mathcal{P}^*_{nekrounds}$ n'est pas au moins aussi fort que $\mathcal{P}_{sym}$. De manière évidente, si $|\Pi| = 2$, $\mathcal{P}^*_{nekrounds}$ implique $\mathcal{P}_{sym}$. On suppose donc $|\Pi| \geq 3$ dans la suite.

Proposition 2.1.5 *Il n'existe pas de translation de $\mathcal{P}^*_{nekrounds}$ en $\mathcal{P}_{sym}$:*

$$\mathcal{P}^*_{nekrounds} \not\preceq \mathcal{P}_{sym}.$$

Démonstration: Supposons qu'il existe un algorithme $\mathcal{A}$ qui translate $\mathcal{P}^*_{nekrounds}$ en $\mathcal{P}_{sym}$. Considérons une exécution de la machine $(\mathcal{A}, \mathcal{P}^*_{nekrounds})$ telle que :

$$\exists q \in \Pi\ :\ \ \forall r > 0,\ \forall p \in \Pi,\ \ HO(p, r) = \{p; q\}.$$

Soit $\rho > 0$ un macro-round et soit p et p' deux processus distincts de q tels que $p \neq p'$. Puisque $\mathcal{A}$ est une translation de $\mathcal{P}^*_{nekrounds}$ en $\mathcal{P}_{sym}$, la condition E2 de la définition d'une translation assure que

$$p \in NewHO^{(\rho)}_{p'} \quad \vee \quad p' \in NewHO^{(\rho)}_{p}.$$

Sans perte de généralité, supposons que

$$p \in NewHO^{(\rho)}_{p'}.$$

La condition E1 implique alors qu'il existe une chaîne de processus $p_0, \cdots, p_l$ de $p_0 = p$ à $p_l = p'$ et une suite de l rounds $r_1 < \cdots < r_l$ telles que, pour tout indice i, $1 \leq i \leq l$:

$$MacroRound^{(r_i)}_{p_i} = \rho,\ \text{ et }\ p_{i-1} \in HO(p_i, r_i).$$

Or, pour tout processus $p'' \in \Pi$ et pour tout round $r > 0$ tel que $MacroRound_{p'}^{(r)} = \rho$, on a $p'' \notin HO(p', r)$, une contradiction. Par conséquent, $\mathcal{A}$ n'est pas une translation de $\mathcal{P}^*_{nekrounds}$ en $\mathcal{P}_{sym}$. $\square$

La combinaison des Propositions 2.1.4 et 2.1.5 nous autorise à énoncer le résultat principal de cette section :

Théorème 2.1.6 *$\mathcal{P}_{sym}$ est strictement plus fort que $\mathcal{P}^*_{nekrounds}$:*

$$\mathcal{P}_{sym} \succ \mathcal{P}^*_{nekrounds}.$$

Nous avons, dans cette section, présenté le prédicat $\mathcal{P}_{sym}$ que l'on peut garantir dans un système SWMR sujets à des pannes bénignes, et démontré que $\mathcal{P}_{sym}$ est strictement plus fort que $\mathcal{P}^*_{nekrounds}$. On en conclut donc que $\mathcal{P}^*_{nekrounds}$ est trop faible pour correspondre aux systèmes SWMR dans le cas des pannes bénignes.

$\mathcal{P}_{sym}$ est équivalent à $\mathcal{P}_{\mathcal{RD}}$

Les prédicats $\mathcal{P}_{sym}$ et $\mathcal{P}^*_{\mathcal{RD}}$ correspondent donc aux systèmes SWMR et aux systèmes Atomic-Snapshot, respectivement, dans le cas des pannes bénignes. Nous démontrons, dans cette section, que ces deux prédicats sont équivalents au sens de l'ordre défini dans le Chapitre 1, ce qui correspond à l'équivalence, démontrée par exemple dans [4, 20, 3], entre systèmes SWMR et systèmes Atomic-Snapshot sujets à des pannes bénignes.

Théorème 2.1.7 *On a*

$$\mathcal{P}^*_{\mathcal{RD}} \Rightarrow \mathcal{P}_{sym},$$

et donc

$$\mathcal{P}^*_{\mathcal{RD}} \succeq_1 \mathcal{P}_{sym}.$$

Démonstration: Soit $\big(HO(p,r)\big)_{p\in\Pi, r>0}$ une collection d'ensembles d'écoute qui satisfait $\mathcal{P}^*_{\mathcal{RD}}$.

Soit $r > 0$ et soient p et q deux pocessus de Π. Puisque $\mathcal{P}^*_{\mathcal{RD}}$ implique $\mathcal{P}_{ref}$, on a

$$p \in HO(p,r) \quad \wedge \quad q \in HO(q,r).$$

Si $p \notin HO(q,r)$ on en déduit $q \neq p$ et

$$HO(p,r) \not\subseteq HO(q,r),$$

ce qui, sous $\mathcal{P}^*_{\mathcal{RD}}$, implique

$$HO(q,r) \subseteq HO(p,r).$$

Par conséquent, on a bien $q \in HO(p,r)$. $\square$

Algorithme 2.4 Algorithme $\mathcal{SRD}$: code du processus p

```
1:  Initialisation
2:    D_p ⊆ Π ; initialement {p}
3:    NewHO_p ⊆ Π ; initialement ∅
4:    Known_p ⊆ Π ; initialement ∅
5:    k_p ∈ {0, 1, 2} ; initialement 0
6:    N_p ∈ ℕ ; initialement 0
7:    MacroRound_p ∈ ℕ ; initialement 1

8:  S_p^r :

9:    if r ≡ 1[n^2 + n] then
10:       Envoyer ⟨p⟩ à tous les processus
11:   else
12:       if NewHO_p = ∅ then
13:           Envoyer ⟨D_p⟩ à tous les processus
14:       else
15:           Envoyer ⟨null⟩ à tous les processus

16: T_p^r :

17:   if r ≡ 1[n^2 + n] then
18:       NewHO_p := ∅
19:       Known_p := ∅
20:       D_p := HO(p, r)

21:       if D_p ≠ {p} then
22:           k_p := 0
23:       else
24:           k_p := 1

25:       N_p := 1

26: if r ≢ 1[n^2 + n] then

27:   if NewHO_p = ∅ then
28:       Known_p := HO(p, r) \ {q : p reçoît ⟨null⟩ de q au round r}

29:       if ⋃{D_q : q ∈ Known_p} ⊆ D_p then
30:           k_p := k_p + 1
31:       else
32:           k_p = 0

33:       D_p := D_p ∪ ⋃{D_q : q ∈ Known_p}

34:       if k_p = 1 then
35:           N_p := |Known_p|
36:       if k_p = 2 then
37:           k_p := 1 ; N_p := N_p − 1
38:           if N_p = 0 then
39:               NewHO_p := D_p
```

Il reste à démontrer que $\mathcal{P}_{sym}$ est au moins aussi fort que $\mathcal{P}^*_{\mathcal{RD}}$. On remarque que si $|\Pi| = 2$, toute collection

$$\big(HO(p,r)\big)_{p\in\Pi, r>0}$$

satisfait $\mathcal{P}^*_{\mathcal{RD}}$. Dans ce cas, on a donc

$$\mathcal{P}_{sym} \Rightarrow \mathcal{P}^*_{\mathcal{RD}}$$

et donc

$$\mathcal{P}_{sym} \succeq_1 \mathcal{P}^*_{\mathcal{RD}}.$$

On suppose donc $|\Pi| \geq 3$. Nous présentons l'algorithme $\mathcal{SRD}$, donné comme Algorithme 2.4, que nous prouvons être une translation de $\mathcal{P}_{sym}$ en $\mathcal{P}^*_{\mathcal{RD}}$. Les exécutions sont décomposées en macro-rounds, composés chacun de $n^2 + n$ rounds consécutifs, où n est le cardinal de Π.

De manière informelle, cet algorithme fonctionne de la manière suivante. À chaque macro-round, chaque processus p met à jour une variable $NewHO_p$ qui représente l'ensemble d'écoute simulé de p. Aussi longtemps qu'un processus p n'a pas déterminé son ensemble $NewHO_p$, il met à jour, à chaque round r, une variable D_p qui consiste en l'ensemble des processus dont p a entendu parler au cours de ρ, directement ou par l'intermédiaire d'autres processus. Tant que p n'a pas déterminé $NewHO_p$, il envoie la valeur courante de D_p ($\{p\}$ au premier round) aux autres processus et met à jour sa variable en fonction des messages qu'il reçoit. Si la valeur de D_p ne change pas pendant un assez grand nombre (qui peut varier d'un macro-round à l'autre) de rounds consécutifs de ρ - c'est-à-dire si p exécute la ligne 37 pendant un assez grand nombre de rounds consécutifs de ρ -, alors p met à jour $NewHO_p$ en lui affectant la valeur D_p. Une fois que p a défini son ensemble $NewHO_p$, il envoie $\langle null \rangle$ à chacun des rounds suivants et ne modifie plus $NewHO_p$.

Nous commençons notre démonstration par deux lemmes élémentaires concernant l'évolution des variables D_p au cours d'un macro-round. On rappelle que pour tout processus p, pour tout round r (resp. macro-round ρ), et toute variable x_p locale à p, on note $x_p^{(r)}$ (resp. $x_p^{(\rho)}$) la valeur de x_p à la fin du round r (resp. du macro-round ρ), et $x_p^{(0)}$ sa valeur initiale.

Lemme 2.1.8 *Soit une exécution de la machine* $(\mathcal{SRD}, \mathcal{P}_{sym})$ *et soit* $\rho > 0$ *un macro-round. Pour tous rounds* r *et* r' *de* ρ *tels que* $r < r'$ *et pour tout processus* $p \in \Pi$ *on a*

$$D_p^{(r)} \subseteq D_p^{(r')}.$$

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$ et soit $\rho > 0$ un macro-round. Soit r et r' deux rounds de ρ tels que $r < r'$ et soit p un processus de Π. On raisonne par récurrence sur $r' - r \geq 1$.

- *Cas de base* $r' = r + 1$. Par définition de r', on a $r' \not\equiv 1[n^2 + n]$. Si p n'exécute pas la ligne 33 de son code au round r', alors $D_p^{(r')} = D_p^{(r)}$. Sinon, on a

  $$D_p^{(r')} = D_p^{(r)} \cup \bigcup \{D_q^{(r)} \ : \ q \in Known_p\},$$

  et donc $D_p^{(r)} \subseteq D_p^{(r')}$.

- *Étape de récurrence* $r' > r + 1$. Supposons que, pour tout r", $r < r" < r'$, on a

  $$D_p^{(r)} \subseteq D_p^{(r")}.$$

  Par le même argument que dans le cas de base, on a

  $$D_p^{(r'-1)} \subseteq D_p^{(r')}.$$

  En posant $r" = r' - 1$, on en conclut $D_p^{(r)} \subseteq D_p^{(r')}$.

□

Lemme 2.1.9 *Soit une exécution de la machine* $(\mathcal{SRD}, \mathcal{P}_{sym})$. *Pour tout round* $r > 0$ *et pour tout processus* $p \in \Pi$ *on a*

$$p \in D_p^{(r)}.$$

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$. Soit$\rho > 0$ un macro-round et soit $p \in \Pi$ un processus.

Soit r_0 le premier round de ρ. Le code (ligne 20) assure $D_p^{(r_0)} = HO(p, r_0)$. Puisque $\mathcal{P}_{sym}$ implique $\mathcal{P}_{ref}$, on en déduit $p \in D_p^{(r_0)}$. Le Lemme 2.1.8 implique que, pour tout round r de ρ, on a

$$D_p^{(r_0)} \subseteq D_p^{(r)}.$$

On en conclut donc que, pour tout round r de ρ, on a

$$p \in D_p^{(r)}.$$

□

Lemme 2.1.10 *Soit une exécution de la machine* $(\mathcal{SRD}, \mathcal{P}_{sym})$*. Au cours de chaque macro-round, chaque processus exécute au plus une fois la ligne 39 de son code.*

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$ et soit $\rho > 0$ un macro-round. Supposons qu'il existe un processus p qui exécute la ligne 39 au cours de ρ.

Soit r_p le premier round de ρ au cours duquel p exécute la ligne 39. Si r_p est le dernier round de ρ, le résultat est immédiat. Sinon soit $r > r_p$ un round de ρ. Démontrons que p n'exécute pas la ligne 39 au round r. On raisonne par récurrence sur r.

- *Cas de base* $r = r_p + 1$. Puisque p exécute la ligne 39 au round r_p, on a $NewHO_p^{(r_p)} = D_p^{(r_p)}$. Le Lemme 2.1.9 implique alors $NewHO_p^{(r_p)} \neq \emptyset$. Comme r n'est pas le premier round de ρ, p n'exécute pas la ligne 18 au round r. On en conclut donc que la garde de la ligne 27 est fausse à ce round et donc que p n'exécute pas la ligne 39.

- *Étape de récurrence* $r > r_p + 1$. Supposons que, pour tout r', $r_p < r' < r$, p n'exécute pas la ligne 39 au round r'. Puisque, pour tout r', $r_p < r' < r$, le round r' n'est pas le premier round de ρ, p n'exécute pas la ligne 18 au round r' et donc $NewHO_p^{(r')} = NewHO_p^{(r'-1)}$. En particulier, on a $NewHO_p^{(r-1)} = D_p^{(r_p)}$. Le Lemme 2.1.9 implique alors $NewHO_p^{(r-1)} \neq \emptyset$. Comme p n'exécute pas la ligne 18 au round r, on en déduit que la garde de la ligne 27 est fausse à ce round et donc que p n'exécute pas la ligne 39.

□

Nous déduisons les deux corollaires suivants qui concernent l'évolution des ensembles $NewHO_p$ au cours d'un macro-round.

Corollaire 2.1.11 *Soit une exécution de la machine* $(\mathcal{SRD}, \mathcal{P}_{sym})$ *et soit* $\rho > 0$ *un macro-round. Pour tout processus* p*, s'il existe un round* r_p *de* ρ *au cours duquel* p *exécute la ligne 39 alors, pour tout round* r *de* ρ*,* $r \geq r_p$*, on a*

$$NewHO_p^{(r)} = NewHO_p^{(r_p)} = D_p^{(r_p)}.$$

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$ et soit $\rho > 0$ un macro-round. Supposons qu'il existe un processus p et un round r_p de ρ tels que p exécute la ligne 39 au round r_p.

Le Lemme 2.1.10 implique que, pour tout round r de ρ, $r < r_p$, p n'exécute pas la ligne 39. Puisque p n'exécute pas la ligne 18, on en conclut que, pour tout round r de ρ, $r_p \leq r$, on a

$$NewHO_p^{(r)} = NewHO_p^{(r_p)} = D_p^{(r_p)}.$$

□

Corollaire 2.1.12 *Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, soit r_0 le premier round de ρ et soit p un processus.*

S'il existe un round r_p de ρ au cours duquel p exécute la ligne 39, alors, pour tout r, $r_0 + 1 \leq r \leq r_p$, on a

$$NewHO_p^{(r-1)} = \emptyset.$$

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$ et soit $\rho > 0$ un macro-round. Supposons qu'il existe un processus p et un round r_p de ρ tels que p exécute la ligne 39 au round r_p.

Supposons qu'il existe un round r, $r_0+1 \leq r < r_p$, tel que $NewHO_p^{(r)} \neq \emptyset$. La manière dont les variables $NewHO_p$ sont mises à jour implique alors qu'il existe un round r' de ρ, $r' \leq r < r_p$, au cours duquel p exécute la ligne 39, ce qui contredit le Lemme 2.1.10. □

Le résultat suivant majore, pour tout processus p, le nombre de rounds d'un même macro-round au cours desquels la valeur de D_p peut être modifiée.

Lemme 2.1.13 *Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, soit r_0 le premier round de ρ, et soit p un processus. Alors :*

(1) Si p exécute la ligne 22 au round r_0 alors p exécute la ligne 32 au plus $n - 2$ fois au cours de ρ.

(2) Si p exécute la ligne 24 au round r_0 alors il existe au plus $n - 2$ rounds r de ρ, $r > r_0 + 1$, au cours desquels p exécute la ligne 32.

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, soit r_0 le premier round de ρ, et soit p un processus.

On commence par démontrer (1). Si p exécute la ligne 22 au round r_0 alors $HO(p, r_0) \neq \{p\}$. Puisque $\mathcal{P}_{sym}$ implique $\mathcal{P}_{ref}$ et donc $p \in HO(p, r_0)$, on en déduit qu'il existe un processus $q \neq p$ tel que $q \in HO(p, r_0)$ et donc $q \in D_p^{(r_1)}$. Ainsi, $|D_p^{(r_1)}| \geq 2$.

Supposons qu'il existe un round r, $r > r_0$, au cours duquel p exécute la ligne 32. Cela implique $D_p^{(r)} \not\subseteq D_p^{(r-1)}$. Or, le Lemme 2.1.8 implique $D_p^{(r-1)} \subseteq D_p^{(r)}$. Par conséquent, on a $D_p^{(r-1)} \subset D_p^{(r)}$, et donc $|D_p^{(r)}| \geq |D_p^{(r-1)}| + 1$. D'autre part, le Lemme 2.1.8 implique que, pour tout $r' > r_0$, on a

$$D_p^{(r'-1)} \subseteq D_p^{(r')}.$$

Comme $|D_p^{(r')}| \leq n$, pour tout r', on en conclut qu'il existe au plus $n - 2$ rounds de ρ au cours desquels p exécute la ligne 32.

Il reste à démontrer (2). Supposons que p exécute la ligne 24 au round r_0. On a donc $k_p^{(r_0)} = 1$ et $N_p^{(r_0)} = 1$. Comme $NewHO_p^{(r_0)} = \emptyset$, p exécute la ligne 30 ou la ligne 32 au round $r_0 + 1$. On distingue deux cas :

- Si p exécute la ligne 30, alors p affecte la valeur 2 à k_p et exécute, par conséquent, la ligne 37. Il affecte alors la valeur $N_p^{(r_0)} - 1 = 0$ à N_p, ce qui implique que p exécute la ligne 39 au round $r_0 + 1$. On a donc $NewHO_p^{(r_0+1)} = D_p^{(r_0)}$. Le Lemme 2.1.9 implique alors $NewHO_p^{(r_0+1)} \neq \emptyset$. Par le Corollaire 2.1.11, on en déduit que, pour tout round r de ρ tel que $r \geq r_0 + 1$, on a $NewHO_p^{(r)} \neq \emptyset$. Cela implique que, pour tout round r de ρ tel que $r > r_0 + 1$, p n'exécute pas la ligne 32. On en conclut donc que p n'exécute pas la ligne 32 au cours de ρ.

- Si p exécute la ligne 32 alors $D_p^{(r_0+1)} \not\subseteq D_p^{(r_0)}$. Le Lemme 2.1.8 implique $D_p^{(r_0)} \subseteq D_p^{(r_0+1)}$. Par conséquent, on a $D_p^{(r_0)} \subset D_p^{(r_0+1)}$, et donc $|D_p^{(r_0+1)}| \geq |D_p^{(r_0)}| + 1$. Le Lemme 2.1.9 implique $|D_p^{(r_0)}| \geq 1$. On en déduit $|D_p^{(r_0+1)}| \geq 2$. L'argument utilisé pour démontrer (1) achève de démontrer qu'il existe au plus $n - 2$ rounds r de ρ tels que $r > r_0 + 1$ au cours desquels p exécute la ligne 32.

□

Corollaire 2.1.14 *Soit une exécution de la machine* $(\mathcal{SRD}, \mathcal{P}_{sym})$ *et soit* $\rho > 0$ *un macro-round. Tout processus* p *exécute la ligne 39 au cours de* ρ.

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round et soit p un processus. On raisonne par contradiction et on suppose que p n'exécute pas la ligne 39 au cours de ρ.

Soit r_0 le premier round de ρ. De manière évidente, p ne peut pas exécuter la ligne 39 au round r_0. Ainsi, pour tout round r de ρ, $r \neq r_0$, la garde de la ligne 27 est vraie. Par conséquent, p n'envoie jamais $\langle null \rangle$ au cours de ρ et, à chaque round r de ρ, $r \neq r_0$, p exécute l'une des lignes 30 ou 32, ainsi que la ligne 33.

Supposons que, pour toute suite $r_1, r_2, \cdots, r_{n+1}$ de $n+1$ rounds consécutifs de ρ, $r_1 \geq r_0 + 1$, il existe un indice i, $i \in \{1, 2, \cdots, n+1\}$, tel que p exécute la ligne 32 au round r_i. Or ρ est composé de $n^2 + n$ rounds, $n \geq 3$. On a

$$n^2 + n - 2 \geq (n+1)(n-1).$$

On en déduit qu'il existe au moins $n - 1$ rounds r de ρ, $r \geq r_0 + 2$, au cours desquels p exécute la ligne 32, ce qui contredit le Lemme 2.1.13. Il existe donc une suite $r_1, r_2, \cdots, r_{n+1}$ de $n + 1$ rounds consécutifs de ρ, $r_1 \geq r_0 + 1$, au cours desquels p exécute la ligne 30.

Soit r_p le premier round de ρ, $r_p \geq r_0 + 1$, tel que p exécute la ligne 30 aux rounds $r_p, r_p + 1, \cdots, r_p + n$. On distingue deux cas :

- $r_p = r_0 + 1$. Si p exécute la ligne 24 au round r_0 on a $k_p^{(r_0)} = 1$. De plus, on a $N_p^{(r_0)} = 1$. Lorsque p exécute la ligne 30 au round $r_0 + 1$, il lui affecte donc la valeur 2. Par conséquent, la garde de la ligne 36 est vraie à ce round, ce qui implique que p exécute la ligne 37 et affecte à N_p la valeur $N_p^{(r_0)} - 1 = 0$. On en déduit que p exécute la ligne 39 au round $r_0 + 1$, une contradiction.

  Le processus p exécute donc la ligne 22 au round r_0, ce qui implique $k_p^{(r_0)} = 0$. Lorsque p exécute la ligne 30 au round $r_0 + 1$, il affecte donc

la valeur 1 à k_p et, par conséquent, affecte la valeur $|Known_p^{(r_p)}|$ à N_p. Comme $\mathcal{P}_{sym}$ implique $\mathcal{P}_{ref}$, on a $p \in HO(p, r_p)$. Comme, d'autre part, p envoie $\langle D_p^{(r_p-1)} \rangle$ à ce round, il s'ensuit que $p \in Known_p^{(r_p)}$. On en déduit $N_p^{(r_p)} > 0$.
Or, on a $N_p^{(r_p)} \leq n$. Par conséquent, pour tout indice i, $i = 1, \cdots, N_p^{(r_p)}$, p exécute la ligne 30 et affecte donc la valeur 2 à k_p au round $r_p + i$. On en déduit que, pour tout indice i, $i = 1, \cdots, N_p^{(r_p)}$, la garde de la ligne 36 est vraie au round $r_p + i$ et donc qu'à ce round p exécute la ligne 37 et n'exécute pas la ligne 35. Par conséquent, pour tout indice i, $i = 1, \cdots, N_p^{(r_p)}$, on a

$$N_p^{(r_p+i)} = N_p^{(r_p+i-1)}.$$

En particulier, on a $N_p^{(r_p+N_p^{(r_p)}-1)} = 1$ et $N_p^{(r_p+N_p^{(r_p)})} = 0$, ce qui implique que p exécute la ligne 39 au round $r_p + N_p^{(r_p)}$, une contradiction.

- $r_p > r_0 + 1$. Par définition de r_p, p exécute nécessairement la ligne 32 au round $r_p - 1$. On a donc $k_p^{(r_p-1)} = 0$. Lorsque p exécute la ligne 30 au round r_p, il affecte donc la valeur 1 à k_p et, par conséquent, affecte la valeur $|Known_p^{(r_p)}|$ à N_p. Comme $\mathcal{P}_{sym}$ implique $\mathcal{P}_{ref}$, on a $p \in HO(p, r_p)$. Comme, d'autre part, p envoie $\langle D_p^{(r_p-1)} \rangle$ à ce round, il s'ensuit que $p \in Known_p^{(r_p)}$. On en déduit $N_p^{(r_p)} > 0$. L'argument utilisé dans le cas précédent démontre alors que p exécute la ligne 39 au round $r_p + N_p^{(r_p)}$, une contradiction.

On en conclut finalement que p exécute nécessairement la ligne 39 au cours de ρ. □

Lemme 2.1.15 *Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, soit r_0 le premier round de ρ et soit p un processus.*

S'il existe un round r_p de ρ, $r_0 + 2 \leq r_p$, au cours duquel p exécute la ligne 39, alors il existe un round r'_p, $r_0 + 1 \leq r'_p < r_p$ tel que :

1. *p exécute la ligne 35 au round r'_p, et*
2. *pour tout r tel que $r'_p < r \leq r_p$, p exécute la ligne 37 au round r.*

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, soit r_0 le premier round de ρ, et soit p un processus. Supposons qu'il existe un round r_p de ρ, $r_0 + 2 \leq r_p$, au cours duquel p exécute la ligne 39. Le Corollaire 2.1.12 implique que, pour tout r, $r_0 < r \leq r_p$, la garde de la ligne 27 est vraie au round r, et donc que p exécute, soit la ligne 30, soit la ligne 32 à ce round.

On démontre tout d'abord (1). On raisonne par contradiction et on suppose que, pour tout r, $r_0 + 1 \leq r < r_p$, p n'exécute pas la ligne 35 au round r. On démontre que, pour tout r, $r_0 + 1 \leq r < r_p$, on a $k_p^{(r)} = 0$. On raisonne par récurrence sur $r - r_0$.

- *Cas de base* $r = r_0 + 1$. Si $k_p^{(r_0)} = 0$, puisque p n'exécute pas la ligne 35 au round $r_0 + 1$, il exécute nécessairement la ligne 32. On a donc bien $k_p^{(r_0+1)} = 0$. Si $k_p^{(r_0)} = 1$, alors, puisque p exécute la ligne 39 au round $r_p \geq r_0 + 2$, le Lemme 2.1.10 implique que p n'exécute pas la ligne 39 au round $r_0 + 1$. On en déduit que p n'exécute pas la ligne 30 à ce round. Il s'ensuit que p exécute la ligne 32 au round $r_0 + 1$ et donc $k_p^{(r_0+1)} = 0$.

- *Étape de récurrence* $r > r_0 + 1$. On suppose que, pour tout r', $r_0 + 1 \leq r' < r$, on a $k_p^{(r')} = 0$. En particulier on a $k_p^{(r-1)} = 0$. Si p exécute la ligne 30 au round r, il affecte à k_p la valeur 1. La garde de la ligne 34 est donc vraie, ce qui implique que p exécute la ligne 35 au round r, une contradiction. Par conséquent, p exécute la ligne 32 au round r et donc $k_p^{(r)} = 0$.

On en conclut $k_p^{(r_p-1)} = 0$. Au round r_p, soit p exécute la ligne 30 et affecte à k_p la valeur 1, soit il exécute la ligne 32 et affecte à k_p la valeur 0. Dans les deux cas, la garde de la ligne 36 est fausse au round r_p, ce qui contredit le fait que p exécute la ligne 39 à ce round.

Il reste à démontrer (2). Soit r'_p, $r_0 + 1 \leq r'_p < r_p$, le dernier round de ρ au cours duquel p exécute la ligne 35. Supposons qu'il existe un round r, $r'_p < r \leq r_p$ au cours duquel p exécute la ligne 32. Puisque, par hypothèse, p exécute la ligne 39 au round r_p, on a $r < r_p$. Le même argument que celui qui est utilisé dans (1), combiné avec la définition de r'_p, implique que, pour tout round r', $r \leq r' < r_p$, on a $k_p^{(r')} = 0$. En particulier, $k_p^{(r_p-1)} = 0$. On

en conclut que p affecte à k_p, au round r_p, soit la valeur 0 soit la valeur 1, et donc que la garde de la ligne 36 est fausse à ce round, ce qui contredit le fait que p exécute la ligne 39 au round r_p. On en conclut que p exécute la ligne 30 à chaque round r, $r'_p < r \leq r_p$.

On démontre alors que, pour tout r, $r'_p < r \leq r_p$, p exécute la ligne 37 au round r. On raisonne par récurrence sur $r - r'_p$.

- Cas de base $r = r'_p + 1$. Par définition de r'_p, on a $k_p^{(r'_p)} = 1$. Puisque p exécute la ligne 30 au round $r'_p + 1$, il affecte à k_p la valeur 2. La garde de la ligne 36 est donc vraie au round $r'_p + 1$, ce qui implique que p exécute la ligne 37 à ce round.

- Étape de récurrence $r > r'_p + 1$. On suppose que, pour tout r', $r'_p + 1 < r' < r$, on a $k_p^{(r')} = 1$. En particulier, on a $k_p^{(r-1)} = 1$. Puisque p exécute la ligne 30 au round r, il affecte à k_p la valeur 2. La garde de la ligne 36 est donc vraie au round r, ce qui implique que p exécute la ligne 37 à ce round.

□

Corollaire 2.1.16 *Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, et soit r_0 le premier round de ρ. Supposons qu'il existe un round r, $r \geq r_0 + 2$, et un processus p qui exécute la ligne 39 au round r.*

Soit r'_p, $r_0 + 1 \leq r'_p < r$, le dernier round tel que p exécute la ligne 35 au round r'_p, et pour tout r tel que $r'_p < r \leq r_p$, p exécute la ligne 37 au round r. On a :

1. *Pour tout r', $r'_p \leq r' \leq r_p$, on a $D_p^{(r')} = D_p^{(r'_p - 1)} = NewHO_p^{(\rho)}$, et*
2. *$r_p = r'_p + N_p^{(r'_p)}$.*

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, et soit r_0 le premier round de ρ. Supposons qu'il existe un round r_p, $r_p \geq r_0 + 2$, et un processus p qui exécute la ligne 39 au round r_p.

Soit r'_p, $r_0 + 1 \leq r'_p < r_p$, le dernier round tel que p exécute la ligne 35 au round r'_p, et pour tout r tel que $r'_p < r \leq r_p$, p exécute la ligne 37 au round r.

L'existence de r'_p est assurée par le Lemme 2.1.15. Le Corollaire 2.1.12 assure que, pour tout r, $r_0 < r \leq r_p$, la garde de la ligne 27 est vraie au round r et donc que p exécute soit la ligne 30 soit la ligne 32 à ce round.

Nous commençons par démontrer (1). Pour cela, on démontre que, pour tout r', $r'_p \leq r' \leq r_p$, on a $D_p^{(r')} = D_p^{(r'_p-1)}$. On raisonne par récurrence sur r'.

- Cas de base $r' = r'_p$. Si p exécute la ligne 32 au round r'_p, alors il affecte la valeur 0 à k_p, et donc la garde de la ligne 34 n'est pas satisfaite, ce qui contredit le fait que p exécute la ligne 35 au round r'_p. Par conséquent, p exécute à ce round la ligne 30. Il s'ensuit $D_p^{(r'_p)} = D_p^{(r'_p-1)}$.

- Étape de récurrence $r' > r'_p$. On suppose $D_p^{(r'-1)} = D_p^{(r'_p-1)}$. Puisque p exécute la ligne 37 au round r', p n'exécute pas la ligne 32. Par conséquent, p exécute la ligne 30. On en déduit $D_p^{(r')} = D_p^{(r'-1)}$, et donc $D_p^{(r')} = D_p^{(r'_p-1)}$.

On a donc, pour tout r', $r'_p \leq r' \leq r_p$,

$$D_p^{(r')} = D_p^{(r'_p-1)} = D_p^{(r_p)}.$$

Le Corollaire 2.1.11 implique $D_p^{(r_p)} = NewHO_p^{(\rho)}$. On en conclut que, pour tout r', $r'_p \leq r' \leq r_p$, on a

$$D_p^{(r')} = D_p^{(r'_p-1)} = NewHO_p^{(\rho)}.$$

Il reste à démontrer (2). Puisque p exécute la ligne 35 au round r'_p, il affecte à N_p la valeur $|Known_p^{(r'_p)}|$. Sous $\mathcal{P}_{sym}$, on a nécessairement $p \in HO(p, r'_p)$. Comme $NewHO_p^{(r'_p-1)} = \emptyset$, p envoie $\langle D_p^{(r'_p-1)} \rangle$ au round r'_p. Il s'ensuit $p \in Known_p^{(r'_p)}$ et donc $|Known_p^{(r'_p)}| > 0$. Le Lemme 2.1.15 implique que, pour tout r', $r'_p < r' \leq r_p$, p exécute la ligne 37 au round r'. On en déduit que, pour tout r', $r'_p < r' \leq r_p$, on a $N_p^{(r')} = N_p^{(r'-1)} - 1$. Par conséquent, on a

$$N_p\big(r'_p + N_p(r'_p) - 1\big) = 1 \quad \wedge \quad N_p\big(r'_p + N_p(r'_p)\big) = 0,$$

ce qui implique que p exécute la ligne 39 au round $r'_p + N_p(r'_p)$. Le Lemme 2.1.10 implique finalement $r_p = r'_p + N_p(r'_p)$. □

Lemme 2.1.17 *Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, et soit r_0 le premier round de ρ. Supposons qu'il existe un round r, $r \geq r_0 + 2$, et un processus p qui exécute la ligne 39 au round r.*

Soit r'_p, $r_0 + 1 \leq r'_p < r$, le dernier round tel que p exécute la ligne 35 au round r'_p, et pour tout r tel que $r'_p < r \leq r_p$, p exécute la ligne 37 au round r.

Il existe au plus $N_p^{(r'_p)} - 1$ processus q distincts de p qui exécutent la ligne 39 au plus tôt au round r'_p et tels que

$$NewHO_p^{(\rho)} \not\subseteq NewHO_q^{(\rho)}.$$

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, et soit r_0 le premier round de ρ. Supposons qu'il existe un round r, $r \geq r_0 + 2$, et un processus p qui exécute la ligne 39 au round r.

Soit r'_p, $r_0 + 1 \leq r'_p < r$, le dernier round tel que p exécute la ligne 35 au round r'_p, et pour tout r tel que $r'_p < r \leq r_p$, p exécute la ligne 37 au round r. L'existence du round r'_p est assurée par le Lemme 2.1.15.

Soit q un processus, distinct de p, qui exécute la ligne 39 au plus tôt au round r'_p et tel que

$$NewHO_p^{(\rho)} \not\subseteq NewHO_q^{(\rho)}.$$

Le Corollaire 2.1.12 implique que p et q envoient $D_p^{(r'_p-1)}$ et $D_q^{(r'_p-1)}$, respectivement, à tous au round r'_p. Sous $\mathcal{P}_{sym}$, on a

$$p \in HO(q, r'_p) \quad \vee \quad q \in HO(p, r'_p),$$

et donc

$$p \in Known_q^{(r'_p)} \quad \vee \quad q \in Known_p^{(r'_p)}.$$

Si $p \in Known_q^{(r'_p)}$ alors $D_p^{(r'_p-1)} \subseteq D_q^{(r'_p)}$. Le Corollaire 2.1.16 implique $D_p^{(r'_p-1)} = NewHO_p^{(\rho)}$. On en déduit $NewHO_p^{(\rho)} \subseteq D_q^{(r'_p)}$. Or le Lemme 2.1.8 et le Corollaire 2.1.11 impliquent $D_q^{(r'_p)} \subseteq NewHO_q^{(\rho)}$. On en déduit donc

$$NewHO_p^{(\rho)} \subseteq NewHO_q^{(\rho)},$$

une contradiction. On en conclut $p \notin Known_q^{(r'_p)}$ et donc $q \in Known_p^{(r'_p)}$.

Or, par définition de r'_p, on a $|Known_p^{(r'_p)}| = N_p^{(r'_p)}$. D'autre part, puisque $\mathcal{P}_{sym}$ implique $\mathcal{P}_{ref}$, on a $p \in HO(p, r'_p)$ et donc $p \in Known_p^{(r'_p)}$. On en conclut finalement qu'il existe au plus $N_p^{(r'_p)} - 1$ tels processus q. □

Lemme 2.1.18 *Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, soit r_0 le premier round de ρ, et soit p un processus.*

Si p exécute la ligne 39 au round r_0+1, alors, pour tout processus $q \in \Pi$, on a

$$NewHO_p^{(\rho)} \subseteq NewHO_q^{(\rho)}.$$

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, et soit r_0 le premier round de ρ. Supposons qu'il existe un processus p qui exécute la ligne 39 au round $r_0 + 1$.

Si p exécute la ligne 22 au round r_0, on a $k_p^{(r_0)} = 0$. Par conséquent, au round $r_0 + 1$, p affecte à k_p soit la valeur 0, soit la valeur 1. Dans les deux cas, la garde de la ligne 36 est fausse, ce qui contredit le fait que p exécute la ligne 39 au round r_0+1. On en déduit que p exécute la ligne 24 au round r_0 et donc que l'on a $D_p^{(r_0)} = \{p\}$. Comme, par hypothèse, p exécute la ligne 39 au round r_0+1, on a $D_p^{(r_0+1)} = D_p^{(r_0)}$ et donc $D_p^{(r_0+1)} = \{p\}$. Le Corollaire 2.1.11 implique alors

$$NewHO_p^{(\rho)} = \{p\}.$$

Soit q un processus distinct de p. Puisque $D_p^{(r_0)} = \{p\}$, on en déduit que $q \notin HO(p, r_0)$ ce qui, sous $\mathcal{P}_{sym}$, implique $p \in HO(q, r_0)$. On en déduit que $p \in D_q^{(r_0)}$. Soit r_q l'unique round de ρ au cours duquel q exécute la ligne 39. L'existence de r_q est assurée par le Lemme 2.1.10 et le Corollaire 2.1.14. Le Lemme 2.1.8 implique $D_q^{(r_0)} \subseteq D_q^{(r_q)}$. D'autre part, le Corollaire 2.1.11 implique

$$NewHO_q^{(\rho)} = D_q^{(r_q)}.$$

On en déduit $p \in NewHO_q^{(\rho)}$ et donc $NewHO_p^{(\rho)} \subseteq NewHO_q^{(\rho)}$. □

Nous déduisons la proposition suivante, qui assure qu'à la fin de chaque macro-round, les ensembles d'écoute simulés sont totalement ordonnés par l'inclusion.

Proposition 2.1.19 *Soit une exécution de la machine* $(\mathcal{SRD}, \mathcal{P}_{sym})$*, et soit* $\rho > 0$ *un macro-round. Pour tous processus* p *et* q*, on a*

$$NewHO_q^{(\rho)} \subseteq NewHO_p^{(\rho)} \qquad \vee \qquad NewHO_p^{(\rho)} \subseteq NewHO_q^{(\rho)}.$$

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, et soit r_0 le premier round de ρ. Soit $p \in \Pi$ (resp. $q \in \Pi$), et soit r_p (resp. r_q) l'unique round de ρ au cours duquel p (resp. q) exécute la ligne 39. L'existence d'un tel round r_p (resp. r_q) est assurée par le Lemme 2.1.10 et le Corollaire 2.1.14.

Si $p = q$, le résultat est immédiat.

Si $p \neq q$, on peut, sans perte de généralité, supposer $r_p \leq r_q$. D'autre part, si $r_p = r_0 + 1$, alors le Lemme 2.1.18 assure

$$NewHO_p^{(\rho)} \subseteq NewHO_q^{(\rho)}.$$

Supposons $r_p \geq r_0 + 2$. Soit r'_p, $r_0 + 1 \leq r'_p < r$, le dernier round tel que p exécute la ligne 35 au round r'_p, et pour tout r tel que $r'_p < r \leq r_p$, p exécute la ligne 37 au round r. L'existence de r'_p est assurée par le Lemme 2.1.15. On raisonne par contradiction et on suppose

$$NewHO_q^{(\rho)} \not\subseteq NewHO_p^{(\rho)} \qquad \wedge \qquad NewHO_p^{(\rho)} \not\subseteq NewHO_q^{(\rho)}.$$

- Si $r'_p = r_p - 1$, le Corollaire 2.1.16 implique $N_p^{(r'_p)} = 1$. Le Lemme 2.1.17 implique alors qu'il n'existe pas de processus q' distinct de p qui exécute la ligne 39 au plus tôt au round r_p et tel que $NewHO_p^{(\rho)} \not\subseteq NewHO_{q'}^{(\rho)}$, une contradiction puisque, par hypothèse, q est un tel processus. On en déduit $r'_p < r_p - 1$.

- Supposons $r'_p = r_p - 2$. Puisque $NewHO_q^{(\rho)} \not\subseteq NewHO_p^{(\rho)}$, il existe $q_0 \in \Pi$ tel que

  $$q_0 \in NewHO_q^{(\rho)} \qquad \wedge \qquad q_0 \notin NewHO_p^{(\rho)}.$$

  La définition de r_p et le Corollaire 2.1.12 impliquent, d'une part, que p et q envoient $D_p^{(r_p-1)}$ et $D_q^{(r_p-1)}$, respectivement, à tous les autres au round r_p, et, d'autre part, que p et q exécutent tous deux les lignes 28 et 33 à ce round.

Si $q \notin HO(p, r_p)$ alors, sous $\mathcal{P}_{sym}$, on a $p \in HO(q, r_p)$, et donc

$$D_p^{(r_p-1)} \subseteq D_q^{(r_p)}.$$

Le Lemme 2.1.8 implique alors $D_p^{(r_p-1)} \subseteq D_q^{(r_q)}$. Or, par le Corollaire 2.1.16, on a $NewHO_p^{(\rho)} = D_p^{(r_p-1)}$ et $NewHO_q^{(\rho)} = D_q^{(r_q)}$. On en déduit

$$NewHO_p^{(\rho)} \subseteq NewHO_q^{(\rho)},$$

une contradiction.

Par conséquent, on a $q \in HO(p, r_p)$, et donc $q \in Known_p^{(r_p)}$. Cela implique $D_q^{(r_p-1)} \subseteq D_p^{(r_p)}$. Puisque $q_0 \notin NewHO_p^{(\rho)} = D_p^{(r_p)}$, on en déduit $q_0 \notin D_q^{(r_p-1)}$. Comme $q_0 \in NewHO_q^{(\rho)}$, la combinaison du Lemme 2.1.8 et du Corollaire 2.1.11 implique qu'il existe un round r_1, $r_p \leq r_1 \leq r_q$, tel que $q_0 \in D_q^{(r_1)} \setminus D_q^{(r_1-1)}$.

* Supposons $r_1 = r_p$. La façon dont les variables D_q sont mises à jour implique alors qu'il existe un processus q_1 tel que
  1. $q_0 \in D_{q_1}^{(r_p-1)}$,
  2. q_1 exécute la ligne 39 au plus tôt au round r_p, et
  3. $q_1 \in HO(q, r_p)$.

  Puisque $q_0 \in D_{q_1}^{(r_p-1)}$ et $q_0 \notin D_q^{(r_p-1)}$, on a $q_1 \neq q$. De plus, le Lemme 2.1.8 et le Corollaire 2.1.11 impliquent $q_0 \in NewHO_{q_1}^{(\rho)}$. Par conséquent, $q_1 \neq p$. Ainsi, il existe deux processus q et q_1, $q \neq q_1$, distincts de p qui exécutent la ligne 39 au plus tôt au round r_p et tels que

  $$NewHO_p^{(\rho)} \not\subseteq NewHO_q^{(\rho)} \quad \wedge \quad NewHO_p^{(\rho)} \not\subseteq NewHO_{q_1}^{(\rho)}.$$

  Or le Corollaire 2.1.16 et le Lemme 2.1.17 assurent qu'il existe au plus $N_p^{(r'_p)} - 1 = r_p - r'_p = 1$ tel processus, une contradiction. On en conclut $r_1 > r_p$.

* Supposons $r_1 = r_p + 1$. La façon dont les variables D_q sont mises à jour implique alors qu'il existe un processus q_1 tel que

1. $q_0 \in D_{q_1}^{(r_p)}$
2. q_1 exécute la ligne 39 au plus tôt au round $r_p + 1$
3. $q_1 \in HO(q, r_p + 1)$.

Supposons $q_0 \in D_{q_1}^{(r_p-1)}$. Le Corollaire 2.1.12 implique que q_1 envoie $D_{q_1}^{(r_p-1)}$ à p au round r_p et que p exécute les lignes 28 et 33 à ce round. On en déduit que si $q_1 \in HO(p, r_p)$ alors $q_1 \in Known_p^{(r_p)}$ et donc $D_{q_1}^{(r_p-1)} \subseteq D_p^{(r_p)}$. On obtient $q_0 \in D_p^{(r_p)}$. Le Corollaire 2.1.11 implique alors $q_0 \in NewHO_p^{(\rho)}$, une contradiction.
Par conséquent, $q_1 \notin HO(p, r_p)$ ce qui, sous $\mathcal{P}_{sym}$, implique $p \in HO(q_1, r_p)$. Le Corollaire 2.1.12 implique, d'une part, que p envoie $D_p^{(r_p-1)} = NewHO_p^{(\rho)}$ au round r_p et donc que $p \in Known_q^{(r_p)}$ et, d'autre part, que q_1 exécute les lignes 28 et 33 à ce round. Par conséquent, on a $NewHO_p^{(\rho)} \subseteq D_{q_1}^{(r_p)}$. Le Corollaire 2.1.12 implique que q_1 envoie $D_{q_1}^{(r_p)}$ à q au round $r_p + 1$ et que q exécute la ligne 39 au round $r_q \geq r_1 = r_p + 1$. Le Corollaire 2.1.12 implique donc que q exécute les lignes 28 et 33 au round $r_p + 1$. Par conséquent, $D_{q_1}^{(r_p)} \subseteq D_q^{(r_p+1)}$. Par le Corollaire 2.1.11, on en déduit $NewHO_p^{(\rho)} \subseteq NewHO_q^{(\rho)}$, une contradiction. On en conclut donc $q_0 \notin D_{q_1}^{(r_p-1)}$. On est donc ramené au cas précédent. L'argument utilisé alors montre que $r_1 > r_p + 1$.

* En itérant le procédé, on montre que, pour tout entier i, $i \geq 0$, on a $r_1 \geq r_p + i$. Or, $r_1 \leq r_q$ et $r_q - r_p < +\infty$, une contradiction. On en conclut $r'_p < r_p - 2$.

- Supposons $r'_p = r_p - 3$. Puisque $NewHO_q^{(\rho)} \not\subseteq NewHO_p^{(\rho)}$, il existe $q_0 \in \Pi$ tel que

$$q_0 \in NewHO_q^{(\rho)} \quad \wedge \quad q_0 \notin NewHO_p^{(\rho)}.$$

Comme précédemment, on montre qu'il existe un round r_1, $r_p \leq r_1 \leq r_q$, tel que $q_0 \in D_q^{(r_1)} \setminus D_q^{(r_1-1)}$. Par le même argument que dans le cas $r'_p = r_p - 2$, on montre que, pour tout entier i, $i \geq 0$, on a $r_1 \geq r_p + i$, une contradiction puisque $r_1 \geq r_q$ et $r_q - r_p < +\infty$. On en conclut donc $r'_p < r_p - 2$.

- En itérant la démarche, on démontre que pour tout entier j, $j \geq 1$, on a $r'_p < r_p - j$, une contradiction.

On en conclut finalement

$$NewHO_q^{(\rho)} \subseteq NewHO_p^{(\rho)} \qquad \vee \qquad NewHO_p^{(\rho)} \subseteq NewHO_q^{(\rho)}.$$

□

Nous démontrons à présent qu'à la fin de chaque macro-round, tout processus p appartient à $NewHO_p$.

Proposition 2.1.20 *Soit une exécution de la machine* $(\mathcal{SRD}, \mathcal{P}_{sym})$*. Pour tout macro-round* ρ*, et pour tout processus* p*, on a*

$$p \in NewHO_p^{(\rho)}.$$

Démonstration: Soit une exécution de la machine $(ST\mathcal{RD}, \mathcal{P}_{sym})$, soit $\rho > 0$ un macro-round, et soit p un processus. Le Corollaire 2.1.14 implique qu'il existe un round r_p de ρ au cours duquel p exécute la ligne 39. Le Corollaire 2.1.11 assure alors $NewHO_p^{(\rho)} = D_p^{(r_p)}$. Par le Lemme 2.1.9, on a $p \in D_p^{(r_p)}$. On en conclut donc

$$p \in NewHO_p^{(\rho)}.$$

□

En combinant les Propositions 2.1.19 et 2.1.20, nous déduisons le théorème suivant :

Théorème 2.1.21 *L'algorithme* $\mathcal{SRD}$ *est une translation en* $n^2 + n$ *rounds de* $\mathcal{P}_{sym}$ *en* $\mathcal{P}_{\mathcal{RD}}$*, et donc*

$$\mathcal{P}_{sym} \succeq_{n^2+n} \mathcal{P}_{\mathcal{RD}}.$$

Démonstration: Soit une exécution de la machine $(\mathcal{SRD}, \mathcal{P}_{sym})$ et soit $\rho > 0$ un macro-round.

Les Propositions 2.1.19 et 2.1.20 impliquent que la collection

$$\left(NewHO_p^{(\rho)}\right)_{p \in \Pi, \rho > 0}$$

induite par l'exécution satisfait $\mathcal{P}_{\mathcal{RD}}$. La condition E2 est donc vérifiée.

La condition E1, quant à elle, est directement impliquée par le code de $\mathcal{SRD}$ (lignes 20, 28, 33 et 39) et par le fait que, sous $\mathcal{P}_{sym}$, pour tout $r > 0$ et pour tout $p \in \Pi$, on a $p \in HO(p, r)$. □

On déduit alors le théorème suivant :

Théorème 2.1.22 *Les prédicats $\mathcal{P}_{sym}$ et $\mathcal{P}_{\mathcal{RD}}$ sont équivalents, i.e.,*

$$\mathcal{P}_{sym} \simeq \mathcal{P}_{\mathcal{RD}}.$$

Démonstration: Le Théorème 2.1.7 implique

$$\mathcal{P}_{\mathcal{RD}} \succeq_1 \mathcal{P}_{sym}$$

et le Théorème 2.1.21 implique

$$\mathcal{P}_{sym} \succeq_{n^2+n} \mathcal{P}_{\mathcal{RD}}.$$

□

2.1.2 Autres objets partagés

Nous nous intéressons dans cette section à d'autres objets partagés qu'il est possible de rencontrer dans la littérature classique. Dans [26], Kruskal *et al.* introduisent une classe générale d'objets partagés, les objets *Read-Modify-Write* (objets RMW). On rencontre aussi, parmi tant d'autres, des objets appelés *files*, dont une description est donnée dans [25].

Objets Read-Modify-Write

Un *objet Read-Modify-Write* est une paire $(X, \mathcal{O})$, où X est une structure de données partagée, ou variable partagée, et $\mathcal{O}$ est une fonction appliquée à des couples $(x; y)$, où x et y appartiennent à l'ensemble des valeurs pouvant être écrites dans X. Un processus qui accède à un objet $(X, \mathcal{O})$ effectue de manière atomique les deux opérations suivantes : 1°) il lit la valeur x inscrite dans X en exécutant l'opération $Read(X)$, puis 2°) écrit dans X la valeur $\mathcal{O}(x, v_p)$, où v_p est une variable locale à p. Les objets RMW sont *atomiques*,

en ce sens que les accès à un objet peuvent se chevaucher mais s'exécutent comme s'ils se produisaient de façon instantanée.

Considérons, par exemple, un registre SWMR atomique que l'on note R. Soit p le seul processus autorisé à écrire dans R. L'ecriture $Write(R, v)$ d'une valeur v par p dans R peut être vue comme l'accès de p à l'objet RMW $(R, \mathbb{I}_v)$, où $\mathbb{I}_v$ est l'application constante égale à v. De manière similaire, la lecture $Read(R)$ du registre R par un processus q peut être vue comme l'accès de q à l'objet RMW (R, Id), où Id est l'application identité (i.e., $\forall x \;\; Id(x, -) = x$).

Nous allons, dans la suite, décrire un certain nombre d'objets RMW et donner les prédicats de communication que nous pensons correspondre aux systèmes disposant de ces objets dans le contexte des pannes bénignes.

Objets *Test&Set*, *Fetch&Add* et *Swap*

La fonction $Test\&Set$, donnée comme Fonction 2.1, prend en argument un couple $(x; true)$, $x \in \{\perp; true\}$ et renvoie la valeur $true$.

Fonction 2.1 $Test\&Set(x, true), \quad x \in \{\perp; true\}$

Pour tout $x \in \{\perp; true\}, \quad Test\&Set(x, true) = true$

La fonction $Fetch\&Add$, donnée comme Fonction 2.2, prend un argument un couple $(x; y) \in (V \cup \{\perp\}) \times V$, où l'ensemble V est arbitraire.

Fonction 2.2 $Fetch\&Add(x, y), \quad (x; y) \in (V \cup \{\perp\}) \times V$

Pour tout $y \in V, \quad Fetch\&Add(\perp, y) = y.$

Pour tous $x, y \in V, \quad Fetch\&Add(x, y) = x + y.$

La fonction $Swap$, donnée comme Fonction 2.3, prend un argument un couple $(x; y) \in (V \cup \{\perp\}) \times V$, où l'ensemble V est arbitraire.

Fonction 2.3 $Swap(x, y), \quad (x; y) \in (V \cup \{\perp\}) \times V$

Pour tout $(x; y) \in (V \cup \{\perp\}) \times V, \quad Swap(x, y) = y.$

Une des propriétés fondamentales de chacune de ces fonctions est que la valeur inscrite dans X n'est plus jamais égale à sa valeur initiale dès lors qu'un des processus a accédé à $(X, \mathcal{O})$. De cette manière, le premier processus qui accède à l'objet est en mesure de savoir qu'il est le premier. Les autres, quant à eux, ont uniquement la possibilité de savoir qu'un autre processus a accédé à l'objet avant eux.

Nous prétendons que, dans tout dans tout système sujet à des pannes bénignes tel que les processus communiquent *via* une infinité de registres SWMR et ont accès à une infinité d'objets $(X, \mathcal{O})$, où $\mathcal{O}$ est l'une des trois fonctions précédentes, il est possible de garantir le prédicat de communication suivant, que nous notons $\mathcal{P}_2^{\mathcal{CN}}$ (pour Consensus Number) :

$$\mathcal{P}_2^{\mathcal{CN}} \ :: \ \forall r > 0, \ \exists q \in \Pi, \ \begin{cases} HO(q, r) = \{q\} \\ \forall p \in \Pi \setminus \{q\}, \quad q \in HO(p, r) \end{cases}$$

Pour le démontrer nous présentons l'algorithme $\mathcal{A}_2^{\mathcal{CN}}$, donné comme Algorithme 2.5. Nous décomposons les exécutions de cet algorithme en rounds. À chaque round $r > 0$, chaque processus p écrit tout d'abord dans son registre R_p^r, puis accède à un objet $(X^r, \mathcal{O})$. Si la valeur qu'il lit dans X^r est différente de la valeur initiale de X^r, alors il parcourt l'ensemble des registres SWMR et met à jour une variable HO_p en lui affectant l'ensemble des processus q tels que p a lu une valeur differente de $\perp$ dans R_q^r. Sinon, il lit son propre registre et met à jour HO_p en lui affectant la valeur $\{p\}$.

Proposition 2.1.23 *Soit $\mathcal{O}$ l'une des trois fonctions $Test\&Set$, $Fetch\&Add$ ou $Swap$. Tout système S sujet à des pannes bénignes dans lequel les processus communiquent via une infinité de registres SWMR et ont accès à une infinité d'objets $(X, \mathcal{O})$ peut garantir le prédicat $\mathcal{P}_2^{\mathcal{CN}}$.*

Démonstration:

Soit donc $\mathcal{O}$ l'une de ces trois fonctions, et soit S un système de n processus, $n \geq 2$, sujet à des pannes bénignes, tel que, pour tout entier $r > 0$, chaque processus p dispose d'un registre SWMR noté R_p^r initialisé avec la

Algorithme 2.5 Algorithme $\mathcal{A}_2^{\mathcal{CN}}$

1: **Initialisation :**
2: $v_p = id_p$ $\{id_p$ est l'identifiant du processus $p\}$
3: $r_p \in \mathbb{N}$; initialement 1
4: $HO_p \subseteq \Pi$

5: **Do forever**

6: $HO_p := \emptyset$
7: $Write(R_p^{r_p}, v_p)$

8: Accéder à $(X_{r_p}, \mathcal{O})$
9: **if** la valeur retournée par $Read(X_p^{r_p})$ est différente de $\perp$ **then**
10: $Read(R_q^{r_p})$ pour tout q de Π
11: $HO_p := \{q :$ la valeur retournée par $Read(R_q^{r_p})$ est différente de $\perp\}$
12: **else**
13: $Read(R_p^{r_p})$
14: $HO_p := \{p\}$
15: $r_p := r_p + 1$

valeur $\perp$ et peut accéder à un objet $(X^r, \mathcal{O})$, initialisé avec la valeur $\perp$.

Considérons une exécution quelconque de $\mathcal{A}_2^{\mathcal{CN}}$ dans S. Soit $r > 0$ et soit p un processus qui met à jour sa variable HO_p au round r. Le code implique alors que p a accédé à X^r. Puisque X^r est atomique, on peut distinguer le seul processus ayant lu $\perp$ dans X^r. On le note q.

Si q met à jour sa variable HO_q au round r, alors, puisqu'il lit $\perp$ dans X^r, il exécute nécessairement la ligne 14. Ainsi, on a $HO_q, = \{q\}$. Supposons qu'il existe un processus p distinct de q qui met à jour sa variable HO_p au round r. Par définition de X^r, le processus p exécute nécessairement la ligne 11 de son code. Puisque p accède à X^r après q, il commence à lire les registres après que q a écrit son identifiant dans R_q^r. Le code (ligne 11) implique alors $q \in HO_p$. □

Objet *Compare&Swap*

La fonction *Compare&Swap* est donnée comme Fonction 2.4. De manière informelle, un processus qui accède à un objet $(X, Compare\&Swap)$ teste la valeur inscrite dans X et ne la modifie que si cette dernière est égale à $\perp$, la valeur initiale de l'objet. D'autre part, si un processus modifie la valeur inscrite dans X, il peut y inscrire une valeur de son choix, différente de $\perp$.

Fonction 2.4 $Compare\&Swap(x, y), \quad (x; y) \in (V \cup \{\perp\}) \times V,$

Pour tout $y \in V$, $Compare\&Swap(\perp, y) = y$
Pour tous $x, y \in V$ $Compare\&Swap(x, y) = x$

Nous prétendons que tout système sujet à des pannes bénignes, dans lequel les processus communiquent *via* une infinité de registres SWMR et peuvent accéder à une infinité d'objets $(X, Compare\&Swap)$, peut garantir le prédicat de communication suivant que nous appelons $\mathcal{P}_n^{\mathcal{CN}}$:

$$\mathcal{P}_n^{\mathcal{CN}} \ :: \ \forall r > 0, \ \exists q \in \Pi \ : \ \forall p \in \Pi, \quad HO(p, r) = \{q\}$$

Pour le démontrer nous présentons l'algorithme $C\&S$, donné comme Algorithme 2.6. Nous décomposons les exécutions de cet algorithme en rounds. À chaque round $r > 0$, chaque processus p écrit tout d'abord dans son registre R_p^r, puis accède à un objet $(X^r, Compare\&Swap)$. Si la valeur qu'il lit dans X^r est égale à $\perp$, il lit son propre registre et met à jour une variable HO_p en lui affectant la valeur $\{p\}$. Sinon, il lit l'identifiant d'un processus $q \neq p$. Il lit alors le registre R_q^r et met à jour sa variable HO_p en lui affectant la valeur $\{q\}$.

Algorithme 2.6 Algorithme $C\&S$

1: **Initialisation :**
2: $v_p = id_p$ $\{id_p$ est l'identifiant du processus $p\}$
3: $r_p \in \mathbb{N}$; initialement 1
4: $HO_p \subseteq \Pi$

5: **Do forever**

6: $HO_p := \emptyset$
7: $Write(R_p^{r_p}, v_p)$

8: Accéder à $(X^{r_p}, Compare\&Swap)$ avec la valeur v_p
9: **if** la valeur retournée par $Read(X_p^{r_p})$ est égale à $\perp$ **then**
10: $Read(R_p^{r_p})$
11: $HO_p := \{p\}$
12: **else**
13: $Read(R_q^{r_p})$ où q est tel que $Compare\&Swap(-, v_p) = v_q$
14: $HO_p := \{q\}$

15: $r_p := r_p + 1$

Proposition 2.1.24 *Tout système S, sujet à des pannes bénignes, dans lequel les processus communiquent* via *une infinité de registres SWMR et peuvent accéder à une infinité d'objets* $(X, Compare\&Swap)$*, peut garantir le prédicat* $\mathcal{P}_n^{\mathcal{CN}}$.

Démonstration: Soit S un système, sujet à des pannes bénignes, dans lequel, pour tout entier $r > 0$, chaque processus dispose d'un registre SWMR que l'on note R_p^r, initialisé avec $\perp$, et peut accéder à un objet $(X^r, Compare\&Swap)$, où X^r est initialisé avec la valeur $\perp$.

Considérons une exécution quelconque de $C\&S$. Soit $r > 0$. Supposons qu'il existe un processus p qui met à jour sa variable HO_p au round r. Alors p a nécessairement accédé à l'objet $(X^r, Compare\&Swap)$. Puisque $(X^r, Compare\&Swap)$ est atomique, on peut distinguer le seul processus qui a lu $\perp$ dans X^r. On le note q. Lorsque q accède à l'objet, il y inscrit son identifiant.

Si $q = p$, les lignes 10 et 11 impliquent $HO_q = \{q\}$.

Sinon, lorsque p accède à l'objet, il y lit l'identifiant de q. Par les lignes 13 et 14, on en déduit $HO_p = \{q\}$. □

File $FIFO$

Une *file* $FIFO$ est une liste (on la note F [pour file]), sur laquelle peuvent être effectuées deux opérations : (1) $enq(F, v)$ place l'élément v en queue de liste, et (2) $deq(F)$ retourne et supprime l'élément placé en tête de liste, ou un message d'erreur si F est vide. On considère des files $FIFO$ atomiques, en ce sens que les opérations peuvent se chevaucher mais s'exécutent comme si elles se produisaient de façon instantanée.

En règle générale, un processus commence par placer un élément dans la file puis récupère l'élément placé en tête de la file. On ne suppose pas qu'il effectue ces deux opérations de manière atomique. Dans un système sujet à des crashs, par exemple, un processus peut donc placer un élément dans la file mais tomber en panne avant d'avoir lu l'élément de tête.

Nous prétendons que tout système sujet à des pannes bénignes dans lequel les processus communiquent *via* une infinité de registres SWMR et ont

accès à une infinité de files $FIFO$ atomiques peut garantir le prédicat de communication $\mathcal{P}_2^{\mathcal{CN}}$ introduit précédemment :

$$\mathcal{P}_2^{\mathcal{CN}} \ :: \ \forall r > 0, \ \exists q \in \Pi, \ \begin{cases} HO(q,r) = \{q\} \\ \forall p \in \Pi \setminus \{q\}, \quad q \in HO(p,r) \end{cases}$$

Pour le démontrer, nous considérons l'algorithme FQ, donné comme Algorithme 2.7, censé être exécuté dans un système S sujet à des pannes bénignes dans lequel, pour tout entier $r > 0$, chaque processus p dispose d'un registre SWMR, noté R_p^r, initialisé avec la valeur $\perp$, et peut accéder à une file FIFO, notée F^r, initialisée avec la valeur $\perp$. Nous décomposons les exécutions de cet algorithme en rounds. À chaque round $r > 0$, chaque processus p écrit son identifiant dans R_p^r, exécute $enq(F^r, id_p)$, puis exécute $deq(F^r)$. Si l'élément retourné par $deq(F^r)$ est égal à $\perp$, alors p lit son propre registre R_p^r et met à jour une variable HO_p en lui affectant la valeur $\{p\}$. Sinon, il parcourt l'ensemble des registres R_q^r et met à jour HO_p en lui affectant l'ensemble des processus q tels que p a lu une valeur différente de $\perp$ dans R_q^r.

Algorithme 2.7 Algorithme FQ

1: **Initialisation :**
2: $v_p = id_p$ $\{id_p$ est l'identifiant du processus $p\}$
3: $r_p \in \mathbb{N}$; initialement 1
4: $HO_p \subseteq \Pi$

5: **Do forever**

6: $HO_p := \emptyset$
7: $Write(R_p^{r_p}, v_p)$

8: $enq(F^{r_p}, v_p)$
9: **if** la valeur retournée par $deq(F^{r_p})$ est différente de $\perp$ **then**
10: $Read(R_q^{r_p})$ pour tout q de Π
11: $HO_p := \{q :$ la valeur retournée par $Read(R_q^{r_p})$ est différente de $\perp\}$
12: **else**
13: $Read(R_p^{r_p})$
14: $HO_p := \{p\}$
15: $r_p := r_p + 1$

Proposition 2.1.25 *Tout système S, sujet à des pannes bénignes, dans lequel les processus communiquent* via *une infinité de registres SWMR et ont accès à une infinité de files $FIFO$ atomiques peut garantir le prédicat de communication $\mathcal{P}_2^{\mathcal{CN}}$.*

Démonstration: Soit S un système sujet à des pannes bénignes dans lequel, pour tout entier $r > 0$, chaque processus p dispose d'un registre R_p^r initialisé avec la valeur $\perp$ et a accès à une file $FIFO$, que l'on note F^r, initialisée avec la valeur $\perp$.

Considérons une exécution de FQ dans S et soit $r > 0$. Supposons qu'il existe un processus p qui met à jour sa variable HO_p au round r. Le code implique alors que p a effectué l'opération $deq(F^r)$. Puisque F^r est atomique, on peut distinguer le seul processus qui exécute $deq(F^r)$ et tel que la valeur retournée alors soit égale à $\perp$; on le note q.

Si q met à jour sa variable HO_q au round r, alors les lignes 13 et 14 impliquent $HO_q, = \{q\}$. Supposons qu'il existe un processus p distinct de q qui met à jour sa variable HO_p au round r et donc effectue $deq(F^r)$. Par définition de F^r, la valeur retournée par $deq(F^r)$ est différente de $\perp$. Par conséquent, p exécute la ligne 11. D'autre part, puisque p effectue $deq(F^r)$ après que q a écrit dans R_q^r, il lit nécessairement l'identifiant de q dans R_q^r. Par la ligne 11, on a donc $q \in HO_p$. $\square$

File $FIFO$ augmentée

Une *file $FIFO$ augmentée* F est une file $FIFO$ telle que l'opération $deq(F)$ retourne l'élément placé en tête de la file mais ne le supprime pas.

Nous prétendons que tout système sujet à des pannes bénignes dans lequel les processus communiquent *via* une infinité de registres SWMR et ont accès à une infinité de files $FIFO$ augmentées peut garantir le prédicat de communication $\mathcal{P}_n^{\mathcal{CN}}$ introduit précédemment :

$$\mathcal{P}_n^{\mathcal{CN}} \ :: \ \forall r > 0, \ \exists q \in \Pi \ : \ \forall p \in \Pi, \quad HO(p, r) = \{q\}$$

Nous considérons l'algorithme FQA, donné comme Algorithme 2.8, censé être exécuté dans un système S sujet à des pannes bénignes dans lequel, pour tout entier $r > 0$, chaque processus p dispose d'un registre SWMR noté R_p^r, initialisé avec la valeur $\perp$, et peut accéder à une file FIFO, notée F^r, initialisée vide. Nous décomposons les exécutions de cet algorithme en rounds. À chaque round $r > 0$, chaque processus p écrit son identifiant dans R_p^r, exécute $enq(F^r, id_p)$, puis exécute $deq(F^r)$. Le processus p lit ensuite le

registre R_q^r tel que id_q est la valeur retournée par $deq(F^r)$ et met à jour une variable HO_p en lui affectant la valeur $\{q\}$.

Algorithme 2.8 Algorithme FQA

1: **Initialisation :**
2: $v_p = id_p$ $\{id_p$ est l'identifiant du processus $p\}$
3: $HO_p \subseteq \Pi$
4: $r_p \in \mathbb{N}$; initialement 1

5: **Do forever**

6: $HO_p := \emptyset$
7: $Write(R_p^{r_p}, v_p)$

8: $enq(F^{r_p}, v_p)$
9: $x :=$ la valeur retournée par $deq(F^{r_p})$
10: $Read(R_x^{r_p})$
11: $HO_p := \{q : x = id_q\}$
12: $r_p := r_p + 1$

Proposition 2.1.26 *Tout système sujet à des pannes bénignes, dans lequel les processus communiquent* via *une infinité de registres SWMR et ont accès à une infinité de files FIFO augmentées, peut garantir le prédicat de communication* $\mathcal{P}_n^{\mathcal{CN}}$.

Démonstration:

Soit S un système sujet à des pannes bénignes dans lequel, pour tout entier $r > 0$, chaque processus p dispose d'un registre R_p^r initialisé avec la valeur $\perp$ et a accès à une file $FIFO$ augmentée, que l'on note F^r, initialisée vide.

Considérons une exécution de FQA dans S et soit $r > 0$. Supposons qu'il existe un processus p qui met à jour sa variable HO_p au round r. Le code implique alors que p a exécuté $enq(F^r, id_p)$. Puisque F^r est atomique et qu'aucun élément n'est jamais supprimé de F^r, on peut distinguer le processus dont l'identifiant est en tête de F^r ; on le note q.

Soit p un processus qui met à jour sa variable HO_p au round r. Le code assure que p exécute $deq(F^r)$. La valeur retournée alors est égale à id_q, par définition de F^r. Par la ligne 11 on en conclut $HO_p = \{q\}$. □

2.2 Systèmes à échanges de messages et prédicats de communication

Comme nous l'avons précisé dans l'introduction, la seconde grande famille de systèmes distribués est celle des systèmes *à échanges de messages*. Dans ceux-ci, contrairement aux systèmes à mémoire partagée où les processus communiquent en accédant à des objets partagés, les communications se font *via* des *canaux de communication*. Il existe un grand nombre de paramètres qui déterminent un système particulier comme, par exemple, les vitesses relatives des processus du système, le degré de synchronisme des communications, le type de pannes considéré : les canaux de communication peuvent ne pas être fiables et perdre, dupliquer ou corrompre des messages ; les processus peuvent être sujets à des pannes bénignes (omission d'envoi ou de réception, crash) ou byzantines (comportement arbitraire).

Nous nous intéressons, dans cette section, à la description dans le modèle HO de certains de ces systèmes, sujets à des pannes bénignes aussi bien que par valeurs. Nous donnons, pour chaque type de systèmes considéré, un prédicat de communication que nous pensons correspondre.

2.2.1 Pannes bénignes

Gafni dans [22] et Charron-Bost et Schiper dans [17] ont passé en revue un grand nombre de systèmes à échanges de messages sujets à des pannes bénignes et ont, pour chacun d'eux, présenté un prédicat de communication qui lui correspond.

Considérons un système synchrone composé de n processus, dont les liens sont fiables et tel que, dans toute exécution, au plus f, $f < n$, processus sont fautifs par omission d'envoi. Puisqu'au plus f processus sont fautifs, au moins $n - f$ processus sont corrects, et seront donc entendus par tous à chaque round. Le prédicat correspondant à ce type de système est $\mathcal{P}_K^f$, qui garantit que le noyau global de toute exécution est de cardinal au moins égal à $n - f$:

$$\mathcal{P}_K^f \ :: \ |K| \geq n - f$$

On peut aussi considérer un système asynchrone dont les liens sont fiables et tel que, dans toute exécution, au plus f processus sont fautifs par crash.

Dans un tel système, au moins $n-f$ processus sont corrects ; chaque processus peut donc attendre de recevoir $n-f$ messages à chaque round sans rester bloqué. Le prédicat qui lui correspond est le prédicat suivant, appelé $\mathcal{P}^f_{HO}$:

$$\mathcal{P}^f_{HO} \ : \ \forall r > 0, \forall p \in \Pi, \quad |HO(p,r)| \geq n - f$$

Le résultat positif de Fischer, Lynch et Paterson ([21]) montre que, dans un système asynchrone dont les liens sont fiables tel que, dans toute exécution, une minorité stricte ($f < \frac{n}{2}$) de processus sont fautifs par crash initial, on peut garantir le prédicat suivant, noté $\mathcal{P}^f_{unif}$:

$$\mathcal{P}^f_{unif} \ : \ \begin{array}{c} \exists \Pi_0 \subseteq \Pi \text{ t.q. } |\Pi_0| \geq n-f, \ \forall r > 0, \ \forall p \in \Pi \quad : \\ HO(p,r) = \Pi_0 \end{array}$$

Dans [19], Dwork *et al.* ont défini un modèle en rounds pour l'étude des systèmes partiellement synchrones sujets à des pannes crashs, c'est-à-dire, de manière informelle, des systèmes qui, à partir d'un moment, se comportent comme des systèmes synchrones dont les liens sont fiables. Le prédicat qui correspond à un système de ce type tel que, dans toute exécution, au plus f processus sont fautifs par crash est le suivant, que nous notons $\mathcal{P}^f_{\diamond sp_unif}$:

$$\mathcal{P}^f_{\diamond sp_unif} \ : \ \begin{array}{c} \exists \Pi_0 \subseteq \Pi \text{ t.q. } |\Pi_0| \geq n-f, \ \exists r_0 > 0, \ \forall r > r_0, \ \forall p \in \Pi \quad : \\ HO(p,r) = \Pi_0 \end{array}$$

Dans [42], Santoro et Widmayer ont défini un modèle en rounds pour l'étude des systèmes synchrones sujets à des erreurs de transmission, entièrement couvert par le modèle HO. Le prédicat de communication qui correspond au cas particulier dans lequel, à chaque round, au plus f erreurs de transmission par omission interviennent est le prédicat suivant, que nous notons $\mathcal{P}^f_{\mathcal{O}}$:

$$\mathcal{P}^f_{\mathcal{O}} \ :: \ \ \forall r > 0 \ : \ \sum_{p \in \Pi} |HO(p,r)| \geq n^2 - f$$

Si les f omissions forment un bloc, c'est-à-dire si les f omissions sont localisées sur un même processus, on considère le prédicat suivant, que nous appelons $\mathcal{P}^{f,bloc}_{\mathcal{O}}$:

$$\mathcal{P}_{\mathcal{O}}^{f,bloc} :: \forall r > 0, \begin{cases} |K(r)| \geq n-1 \\ \\ \exists \Pi_r \subseteq \Pi \text{ t.q. } |\Pi_r| \geq n-f \quad \wedge \quad \forall p \in \Pi_r,\ HO(p,r) = \Pi \end{cases}$$

Systèmes de Dolev *et al.* [18].

Dolev, Dwork et Stockmeyer ont, dans [18], défini cinq paramètres de synchronisme qui caractérisent, selon eux, un système sujet à des pannes crash. La plupart sont classiques (vitesses relatives des processus[3], délais d'acheminements des messages[4]) et ont fait l'objet d'études poussées ([19, 15]). L'un d'entre eux, une propriété sur l'ordre de réception des messages, est toutefois très peu étudié et ne se retrouve, à notre connaissance, que dans [18]. Chacun de ces paramètres peut prendre deux valeurs, "favorable" ou "défavorable". En faisant varier les valeurs de ces paramètres, on obtient donc trente-deux types différents de systèmes, chacun d'entre eux étant entièrement déterminé par une instanciation des cinq paramètres.

De manière informelle, un système distribué est composé de n, ($n \geq 2$), processus qui communiquent en s'envoyant des messages *via* des *canaux de communication*. Les processus sont modélisés par des automates à espace d'états infinis qui effectuent les actions suivantes : (1) recevoir un ensemble possiblement vide de messages, (2) effectuer une transition d'état basée sur les messages reçus, ou sur le fait qu'aucun message n'a été reçu, et sur son état courant, et (3) envoyer un message, qui dépend de l'état courant du processus, à un ou à plusieurs processus. Selon les hypothèses faites sur le système, un *pas de calcul* désigne l'exécution atomique de la séquence de ces trois actions, ou bien l'exécution atomique des deux premières ou des deux dernières. On impose que deux processus ne peuvent pas prendre de pas de calcul simultanément ; en d'autres termes, les exécutions sont supposées être *séquentielles*.

Nous décrivons à présent les valeurs prises par chacun des paramètres. Les lettres U et F indiquent respectivement les caractères défavorable et favorable.

[3]On parlera de synchronisme des processus.
[4]Synchronisme des communications.

• Les processus peuvent être :

- U. *asynchrones :* chaque processus peut attendre un temps arbitraire, bien que fini, entre deux pas de calcul successifs. Dans toute exécution infinie, un processus qui ne prend qu'un nombre fini de pas de calcul est déclaré *fautif*.

- F. *synchrones :* Il existe une constante $\Phi \geq 1$ telle que dans tout intervalle de temps dans lequel un processus prend $\Phi + 1$ pas de calcul, tout processus non-fautif prend au moins un pas de calcul.

• Les communications peuvent être :

- U. *asynchrones :* Il n'existe pas de borne sur les délais d'acheminement des messages. Tout processus qui exécute une infinité de fois *Receive* reçoit tous les messages qui lui ont été envoyés. On ne considère donc pas les pertes de messages.

- F. *synchrones :* Il existe une constante $\Delta \geq 1$ telle que chaque message est reçu au plus tard Δ unités de temps après son émission si le récepteur n'est pas fautif. Si $\Delta = 1$, on dira que les communications sont *immédiates*.

• L'ordre de réception des messages, ou *Message-Order* dans la suite, peut être :

- U. *asynchrone :* Aucune hypothèse n'est faite sur l'ordre de réception des messages.

- F. *synchrone :* Si deux processus p et p' envoient respectivement un message m et un message m' à un processus q et si l'envoi de m a lieu avant celui de m', alors, si q reçoît m', q a reçu m avant m'.

• Le mécanisme de transmission peut être :

- U. *point-à-point :* En un pas de calcul atomique, tout processus peut envoyer un message à au plus un processus.

– F. *diffusion :* En un pas de calcul atomique, tout processus peut envoyer des messages à tous les autres.

• La Réception/Envoi peut être :

– U. *séparée :* En un pas de calcul atomique, un processus ne peut pas recevoir et envoyer.

– F. *atomique :* La réception et l'envoi font partie du même pas de calcul atomique.

Dolev *et al.* ont étudié la résolubilité du problème du Consensus dans chacun des trente-deux type de systèmes obtenus en faisant varier les valeurs des paramètres et ont déterminé, pour chaque type de systèmes, le nombre maximum de processus pouvant être fautifs dans une exécution d'un algorithme résolvant le Consensus. Ils ont, de plus, identifié quatre cas dans lesquels un algorithme tolérant jusqu'à n crashs existe mais tel que tout changement de la valeur d'un paramètre de favorable à défavorable rend le problème du Consensus non-résoluble en présence de f processus fautifs par crash, où f est égal à 1 ou 2 selon les cas. Ces quatre types de systèmes sont les suivants :

1. processus et communications synchrones ;
2. processus synchrones, Message-Order synchrone ;
3. Diffusion, Message-Order synchrone ;
4. communications synchrones, Diffusion, Réception/Envoi atomique.

Dans [17], Charron-Bost et Schiper indiquent que le prédicat de communication qui correspond aux systèmes du type (1) tels que, dans toute exécution, au plus f processus sont fautifs par crash, est la conjonction des prédicats $\mathcal{P}_K^f$ et $\mathcal{P}_{reg}$, avec :

$$\mathcal{P}_K^f \ :: \quad |K| \geq n - f$$

$$\mathcal{P}_{reg} \ :: \quad \forall r > 0,\ \forall p \in \Pi \ : \ HO(p, r+1) \subseteq K(r)$$

et que le prédicat de communication qui correspond aux systèmes du type (4) tels que, dans toute exécution, au plus f processus sont fautifs par crash,

est le prédicat $\mathcal{P}^f_{sp_unif}$, conjonction des prédicats $\mathcal{P}^f_{HO}$ et $\mathcal{P}_{sp_unif}$, où :

$$\mathcal{P}^f_{HO} \ :: \ \forall r > 0, \ \forall p \in \Pi \quad : \quad |HO(p,r)| \geq n - f$$

$$\mathcal{P}_{sp_unif} \ :: \ \forall r > 0, \ \forall p, q \in \Pi \quad : \quad HO(p,r) = HO(q,r)$$

Nous nous proposons, dans cette section, de décrire formellement le modèle de Dolev *et al.* dans un premier temps, puis de présenter les prédicats qui correspondent, selon nous, aux systèmes des types (2) et (3) sujets à des pannes crash.

Modèle de Dolev *et al.* [18]

Dans [18], un système distribué est défini de la manière suivante. On suppose que l'on dispose d'un ensemble Π de cardinal n, ($n \geq 2$), et d'un ensemble infini M de messages. À chaque élément p de Π est associé un *processus* formé des composants suivants : (1) un ensemble infini d'états $states_p$, (2) un sous-ensemble $init_p$ d'états initiaux, (3) un buffer B_p destiné à contenir les messages envoyés à p qu'il n'a pas encore reçus sur lequel peuvent être exécutées deux opérations :

- $Send(p, m)$ place le message m dans B_p, et
- $Receive(p)$, que seul p peut exécuter, qui retourne un ensemble μ possiblement vide de messages présents dans B_p et supprime μ de B_p,

(4) une fonction d'envoi de messages S_p et (5) une fonction de transition d'états T_p. Les propriétés des opérations *Receive*, ainsi que les définitions des fonctions de transition et d'envoi, dépendent des valeurs des paramètres.

Si Message-Order est *synchrone*, le contenu b_p du buffer B_p est une liste finie. L'opération $Send(p, m)$ ajoute le message m à la fin de b_p. L'opération $Receive(p)$ retourne et supprime un préfixe, possiblement vide, de b_p.

* Si la Réception/Envoi est *séparée*, l'ensemble $states_p$ est divisé en deux sous-ensembles disjoints $states^r_p$ (les états de réception) et $states^s_p$ (les états d'envoi). Les fonctions T_p et S_p sont alors définies de la manière

suivante :

$$T_p \ : \ \left| \begin{array}{l} states_p^r \times M^* \longrightarrow states_p^s \\ \\ states_p^s \longrightarrow states_p^r \end{array} \right.$$

$$S_p \ : \ states_p^s \times \Pi \longrightarrow M \cup \{\emptyset\}$$

où M^* est l'ensemble des listes finies d'éléments de M. Ces définitions imposent, d'une part, que les états d'envoi et de transition alternent et, d'autre part, qu'un processus ne peut pas recevoir de message s'il est dans un état d'envoi ou envoyer un message s'il est dans un état de réception.
Un *pas de calcul* du processus p se trouvant dans l'état s_p est défini :

1. si $s_p \in states_p^r$, par l'exécution atomique de $Receive(p)$ et de la transition d'états de s_p à $T_p(s_p, \mu)$, où μ est la liste, possiblement vide, de messages retournée par $Receive(p)$,

2. si $s_p \in states_p^s$, par l'exécution atomique de $Send(q, S_p(s_p, q))$ pour un certain processus q, si les transmission sont point-à-point, et de la la transition d'états de s_p à $T_p(s_p)$ ou alors, dans le cas de la Diffusion, par l'exécution atomique de $Send(q, S_p(s_p, q))$ pour tous $q \in \Pi$ et de la transition d'états de s_p à $T_p(s_p)$.

$*$ Si la Réception/Envoi est *atomique*, les fonctions T_p et S_p sont définies de la manière suivante :

$$T_p \ : \ states_p \times M^* \longrightarrow states_p$$

$$S_p \ : \ states_p^s \times \Pi \longrightarrow M \cup \{\emptyset\}$$

Dans ce cas, un processus peut recevoir et envoyer des messages quel que soit l'état dans lequel il se trouve.
Un pas de calcul de p se trouvant dans l'état s_p est l'exécution atomique de $Receive(p)$, de la transition d'états de s_p à $T_p(s_p, \mu)$, où μ est la liste, possiblement vide, de messages retournée par $Receive(p)$ et de l'exécution de $Send(q, S_p(s_p, q))$ pour un certain processus q si les transmissions sont point-à-point, ou alors, pour tous $q \in \Pi$ dans le cas

de la Diffusion.

Si Message-Order est *asynchrone*, le contenu b_p de B_p est un ensemble fini non-ordonné, l'opération $Send(p, m)$ ajoute le message m à b_p, et l'opération $Receive(p)$ retourne et supprime un sous-ensemble, possiblement vide, de b_p. Les définitions des fonctions T_p et S_p ne diffèrent du cas où Message-Order est synchrone que par le fait que M^* désigne l'ensemble des ensembles finis d'éléments de M. La définition d'un pas de calcul est la même.

Une *configuration* C est une collection $(s_p; b_p)_{p \in \Pi}$. Une *configuration initiale* I est une configuration telle que, pour tout $\in \Pi$, on a $s_p \in init_p$ et $b_p = \emptyset$. Si un processus p prend un pas de calcul σ_p dans une configuration C, on note $\sigma_p(C)$ la configuration résultante.

Une *exécution* d'un système est entièrement déterminée par une configuration initiale I et par une suite infinie de pas de calcul $\sigma = \sigma_1\sigma_2\sigma_3, \cdots$ telle que σ_1 est effectué dans I, σ_2 est effectué dans $\sigma_1(I)$, σ_3 est effectué dans $\sigma_2(\sigma_1(I))$, etc. Pour tout indice i, $i \geq 1$, on note $[\sigma_1 \cdots \sigma_i](I)$ la configuration atteinte par l'exécution de la suite de pas de calcul $\sigma_1 \cdots \sigma_i$.

Un processus est *fautif* dans une exécution s'il ne prend qu'un nombre fini de pas de calcul.

Les processus sont Φ-*synchrones*, $\Phi \geq 1$, dans une exécution σ si, pour toute suite τ de pas de calcul consécutifs de σ, si un processus prend $\Phi + 1$ pas de calcul dans τ et si un processus p ne prend pas de pas de calcul dans τ, alors p ne prend aucun pas de calcul après τ. Un tel processus p est nécessairement fautif dans σ.

Les communications sont Δ-*synchrones* , $\Delta \geq 1$, dans une exécution σ si, pour tout indice j, si σ_j est la réception d'un ensemble de message μ par un processus p, si un message m a été envoyé à p lors d'un pas de calcul σ_i où $i \leq j - \Delta$ et si aucun des événements σ_k, $i < k < j$, n'est la réception de m par p, alors m appartient à μ.

Message-Order synchrone et Diffusion

On considère un système dans lequel les processus et le communications sont asynchrones, la Réception/Envoi est séparée, Message-Order est syn-

Algorithme 2.9 Algorithme MB : code du processus p

1: **Initialisation :**
2: $HO_p \subseteq \Pi$
3: $r_p \in \mathbb{N}$; initialement 1

4: **Do forever :**

5: Diffuser $\langle p; r_p \rangle$ à tous

6: Exécuter $Receive(p)$ tant que $|\{\langle -; r_p \rangle \text{ reçus}\}| < n - f$
7: $HO_p := \{q : \langle q; r_p \rangle$ est un des $n - f$ premiers messages $\langle -; r_p \rangle$ reçus$\}$
8: $r_p := r_p + 1$

chrone et le mécanisme de transmission est la Diffusion. Nous prétendons que le prédicat de communication correspondant à un système de ce type tel que, dans toute exécution, au plus f processus peuvent être fautifs par crash est le prédicat $\mathcal{P}^f_{sp_unif}$ qui, comme nous l'avons vu, correspond aux systèmes dans lesquels les communications sont synchrones, la Réception/Envoi est atomique, le mécanisme de transmission est la Diffusion et tels qu'au plus f processus sont fautifs par crash dans chaque exécution [17] :

$$\mathcal{P}^f_{sp_unif} :: \left| \begin{array}{rcl} \forall r > 0,\ \forall p \in \Pi & : & |HO(p,r)| \geq n - f \\ & & \wedge \\ \forall r > 0,\ \forall p, q \in \Pi & : & HO(p,r) = HO(q,r) \end{array} \right.$$

Pour nous en convaincre, nous considérons l'algorithme MB, donné comme Algorithme 2.9. On décompose les exécutions en "rounds" définis de la manière suivante : chaque processus met à jour une variable r_p à valeurs entières positives, ainsi qu'une variable HO_p. Initialement, toutes les variables r_p ont la valeur 1. Pour chaque valeur prise par r_p, chaque processus p (1) diffuse son identifiant, étiqueté avec r_p, (2) attend de recevoir au moins $n - f$ messages étiquetés avec r_p, (3) met à jour HO_p en lui affectant l'ensemble des $n - f$ premiers identifiants étiquetés avec r_p reçus et (4) incrémente finalement r_p de 1. On utilisera dans la suite le mot round pour désigner l'exécution des opérations (1)-(4) et on notera $HO_p^{(r)}$ la valeur de HO_p à la fin du round r.

On a supposé que la Réception/Envoi est séparée et donc que les états de réception et d'envoi alternent. Puisque, pour tout entier $r > 0$, un processus p qui diffuse $\langle p, r \rangle$ ne diffuse aucun autre message avant d'incrémenter r, on omet de préciser dans le code les diffusions du message vide.

Proposition 2.2.1 *Pour toute exécution de MB dans laquelle au plus f processus sont fautifs par crash, la collection $(HO_p^{(r)})_{r>0,p\in\Pi}$ induite satisfait $\mathcal{P}^f_{sp_unif}$.*

Démonstration: Considérons une exécution de MB dans laquelle au plus f processus sont fautifs par crash, et soit $r > 0$ un entier quelconque. Puisqu'au plus f processus sont fautifs par crash, la règle de réception de la ligne 6 est non-bloquante.

Supposons qu'il existe un processus p qui met à jour sa variable HO_p au round r. On montre tout d'abord que $|HO_p^{(r)}| = n - f$.

La règle de la ligne 6 étant non-bloquante, le code implique que p reçoit au round r au moins $n - f$ message étiquetés avec r. Puisque chaque processus diffuse au plus un message $\langle -; r\rangle$, les messages $\langle -; r\rangle$ reçus par p sont tous distincts. Par la ligne 7, on en conclut $|HO_p^{(r)}| = n - f$.

Supposons, à présent, qu'il existe un processus $q \neq p$ qui met à jour sa variable HO_q. Par hypothèse, deux processus ne peuvent pas effectuer de pas de calcul simultanément. On peut donc renommer $p_1, p_2, \ldots$ les processus en fonction de l'ordre dans lequel ils diffusent leur message $\langle -; r\rangle$ au round r. Puisque p et q sont tous deux destinataires des messages envoyés par $p_1, p_2, \ldots, p_{n-f}$, que le système vérifie la propriété de Message-Order synchrone et que, pour tout indice i, $1 \leq i \leq n - f$, p_i diffuse son message après p_{i-1}, les processus p et q reçoivent au round r les mêmes $n - f$ premiers messages. La ligne 7 implique alors $HO_p^{(r)} = HO_q^{(r)}$.
□

Processus synchrones et Message-Order synchrone

On s'intéresse ici au prédicat de communication correspondant aux systèmes sujets à des pannes crash dans lesquels les processus sont Φ-synchrones ($\Phi \geq 1$), Message-order est synchrone, les communications sont asynchrones, la Réception/Envoi est séparée et les transmissions se font point-à-point. Nous prétendons que le prédicat correspondant aux systèmes de ce type tels que, dans toute exécution, au plus f processus sont fautifs par crash, est le prédicat $\mathcal{P}^f_K \wedge \mathcal{P}_{reg}$ qui, comme nous l'avons précisé, apparaît dans [17] comme étant le prédicat correspondant aux systèmes dans lesquels les processus et les

communications sont synchrones, la Réception/Envoi est séparée, Message-Order est asynchrone et les transmissions se font point-à-point, et dans toute exécution duquel au plus f processus peuvent être fautifs par crash :

$$\mathcal{P}_K^f \ :: \ |K| \geq n - f$$

$$\mathcal{P}_{reg} \ :: \ \forall r > 0, \ \forall p \in \Pi \quad : \quad HO(p, r+1) \subseteq K(r)$$

Algorithme 2.10 Algorithme PM : code du processus p_j

1: **Initialisation :**
2: $Known_{p_i} \subseteq \Pi \times \mathbb{N}$; initialement $\emptyset$
3: $TempHO_{p_i} \subseteq \Pi$; initialement $\emptyset$
4: $HO_{p_i} \subseteq \Pi$; initialement $\emptyset$
5: $r_{p_i} \in \mathbb{N}$; initialement 1
6: $Alive_{p_i} \in (\mathbb{N} \cup \{\perp\})^{\Pi}$; initialement $(0, 0, \cdots, 0)$

7: $TempHO_{p_i} := \{p_k \ : \ (p_k, r_{p_i}) \in Known_{p_i}\}$
8: $Known_{p_i} := Known_{p_i} \setminus \{(-, r) \ : \ r = r_{p_i}\}$

9: **Pour tout $j = 1, 2, \cdots n$**

10: Exécuter $Send(p_j, \langle p_i; r_{p_i} \rangle)$
11: Exécuter $Receive(p_i)$

12: **Pour tout message m reçu, exécuter dans l'ordre de réception :**

13: **if** $m = \langle p_k, r' \rangle$, $r' > r_{p_i}$ **then**
14: $Alive_{p_i}[p_k] := 0$
15: $Known_{p_i} := Known_{p_i} \cup \{(p_k, r')\}$

16: **if** $m = \langle p_k, r_{p_i} \rangle$ **then**
17: $Alive_{p_i}[p_k] := 0$
18: $TempHO_{p_i} := TempHO_{p_i} \cup \{p_k\}$

19: **if** $m = \langle p_k, alive \rangle$ **then**
20: $Alive_{p_i}[p_k] := 0$

21: **if** $m = \langle p_i, alive \rangle$ **then**
22: Pour tout $k = 1, 2, \cdots, n$
23: **if** $0 \leq Alive_{p_i}[p_k] \leq \Phi$ **then**
24: $Alive_{p_i}[p_k] := Alive_{p_i}[p_k] + 1$
25: **else**
26: $Alive_{p_i}[p_k] := \perp$

27: **Tant qu'il existe $l \in \{1, 2, \cdots, n\}$, t.q. $Alive_{p_i}[p_l] \neq \perp$ et $p_l \notin TempHO_{p_i}$:**

28: **Pour tout $j = 1, 2, \cdots, n$:**

29: Exécuter $Send(p_j, \langle p_i; alive \rangle)$
30: Exécuter $Receive(p_i)$

31: **Pour tout message m reçu, exécuter dans l'ordre de réception :**

32: **if** $m = \langle p_k, r' \rangle$, $r' > r_{p_i}$ **then**
33: $Alive_{p_i}[p_k] := 0$
34: $Known_{p_i} := Known_{p_i} \cup \{(p_k, r')\}$

35: **if** $m = \langle p_k, r_{p_i} \rangle$ **then**
36: $Alive_{p_i}[p_k] := 0$
37: $TempHO_{p_i} := TempHO_{p_i} \cup \{p_k\}$

38: **if** $m = \langle p_k, alive \rangle$ **then**
39: $Alive_{p_i}[p_k] := 0$

40: **if** $m = \langle p_i, alive \rangle$ **then**
41: Pour tout $k = 1, 2, \cdots, n$
42: **if** $0 \leq Alive_{p_i}[p_k] \leq \Phi$ **then**
43: $Alive_{p_i}[p_k] := Alive_{p_i}[p_k] + 1$
44: **else**
45: $Alive_{p_i}[p_k] := \perp$

46: $HO_{p_i} := TempHO_{p_i}$
47: $r_{p_i} := r_{p_i} + 1$

Nous présentons l'algorithme PM, donné comme Algorithme 2.10, censé être exécuté dans un système de ce type. Comme pour l'algorithme MB précédent, nous décomposons les exécutions en rounds. À chaque round $r > 0$, chaque processus p_i met à jour une variable r_{p_i} qui représente le numéro du round auquel il se trouve, une variable HO_{p_i} qui représente l'ensemble d'écoute de p_i au round r, une variable $TempHO_{p_i}$ qui contient les identifiants des processus que p_i entend au cours du round r et une variable $Known_{p_i}$ dans laquelle il stocke, à chaque round, les messages qu'il reçoit et qui sont étiquetés avec un numéro de round supérieur à r. Tout au long de l'exécution, p_i met à jour un tableau $Alive_{p_i}$ à n composantes, une pour chaque processus, destinée à lui indiquer les processus ayant arrêté de fonctionner.

À chaque round, chaque processus p_i place dans $TempHO_{p_i}$ les identifiants des processus dont un message étiqueté avec son numéro de round courant se trouve dans $Known_{p_i}$ et supprime de $Known_{p_i}$ les messages en question. Il envoie ensuite, à tous les autres, son identifiant étiqueté avec son numéro de round courant et attend de recevoir un message, étiqueté avec le numéro de son round courant, de chacun des processus dont il n'a pas encore déterminé qu'il a arrêté de fonctionner. D'autre part, il envoie de façon répétée, à tous les autres, des messages leur indiquant qu'il est toujours actif. Il met finalement à jour sa variable HO_{p_i} en lui affectant l'ensemble des identifiants des processus dont il a reçu un message étiqueté avec son numéro de round courant et incrémente r_{p_i} de 1.

Lemme 2.2.2 *Pour toute exécution de PM, pour tout processus p_i fautif, tout processus p_j correct affecte finalement la valeur $\perp$ à $Alive_{p_j}[p_i]$.*

Démonstration: Soit $\sigma = \sigma_1\sigma_2\cdots$ une exécution de PM. Soit p_i un processus fautif dans σ. Il existe un pas de calcul σ_i de σ tel que p_i ne prend aucun pas de calcul après σ_i. Soit p_j correct dans σ. Le code implique que p_j exécute une infinité de fois $Send(p_j, \langle p_j, alive\rangle)$ après σ_i. Puisqu'il est correct et que p_i ne prend aucun pas de calcul après σ_i, on en déduit que p_j reçoit une infinité de fois $\langle p_j, alive\rangle$ sans recevoir de message de p_i. Le code implique que p_j affecte finalement la valeur $\perp$ à $Alive_{p_j}[p_i]$. $\square$

Lemme 2.2.3 *Dans toute exécution de PM, aucun processus correct ne reste bloqué dans la boucle initiée à la ligne 27.*

Démonstration: Soit $\sigma = \sigma_1\sigma_2\cdots$ une exécution de PM. Soit $r > 0$ un entier. Supposons qu'il existe un processus p_i correct dans σ qui au round r

entre dans la boucle initiée à la ligne 27. On démontre, par récurrence sur $r > 0$, que p sort nécessairement de la boucle et exécute la ligne 47.

- *Cas de base* $r = 1$. Tout processus p_j correct exécute nécessairement $Send(p_i, \langle p_j, 1\rangle)$. Puisque p_i est correct, il existe un pas de calcul au cours duquel p_i reçoit $\langle p_j, 1\rangle$ et donc place p_j dans $TempHO_{p_i}$. De plus, le Lemme 2.2.2 implique que, pour tout processus p_j fautif, p_i affecte finalement la valeur $\perp$ à $Alive_{p_i}[p_j]$. On en conclut que p_i exécute finalement la ligne 47 au round 1.

- *Étape de récurrence* $r > 1$. Supposons que tout processus correct p_j exécute la ligne 47 au round $r - 1$. Puisque p_j est correct dans σ, il exécute nécessairement $Send(p_i, \langle p_j, r\rangle)$. Comme p_i est correct dans σ, il existe un pas de calcul au cours duquel p_i reçoit le message $Send(p_i, \langle p_j, r\rangle)$. S'il le reçoit au round $r' < r$, les lignes 15 et 34 assurent $(p_j, r_{p_j}) \in Known_{p_i}^{(r)}$. La ligne 8 impose alors p_i place p_j dans $TempHO_{p_i}$ au début du round r.
  Si p_i reçoit le message $\langle p_j, r\rangle$ au round r, les lignes 18 et 37 impliquent que p_i place p_j dans $TempHO_{p_i}$
  Enfin, le Lemme 2.2.2 implique que, pour tout processus p_k fautif, p_i affecte finalement la valeur $\perp$ à $Alive_{p_i}[p_k]$. On en conclut que p_i exécute finalement la ligne 47 au round r.

□

Lemme 2.2.4 *Pour toute exécution de PM, pour tout entier $r > 0$ et pour tous processus p_i et p_j, si p_i exécute la ligne 46 au round r et si p_j exécute $Send(p_i, \langle p_j, r\rangle)$, alors $p_j \in HO_{p_i}^{(r)}$.*

Démonstration: Soit $\sigma = \sigma_1\sigma_2 \cdots \sigma_m \cdots$ une exécution de PM et soit $r > 0$ un entier. Supposons qu'il existe deux processus p_i et p_j tels que p_i exécute la ligne 46 au round r, p_j exécute $Send(p_i, \langle p_j, r\rangle)$ au cours d'un pas de calcul σ_j de σ.

Puisque p_i exécute la ligne 47 au round r, il sort de la boucle initiée à la ligne 27. Si $p_j \notin HO_{p_i}^{(r)}$, alors nécessairement $p_j \notin TempHO_{p_i}^{(r)}$. On en déduit donc qu'il existe un pas de calcul σ_i au cours duquel p_i affecte la valeur $\perp$ à $Alive_{p_i}[p_j]$. Par conséquent, p_i a reçu $\Phi + 2$ messages $\langle p_i, alive\rangle$ consécutifs

sans recevoir aucun message de p_j. Puisque Message-Order est synchrone, on en déduit qu'il existe $\Phi + 2$ pas de calcul $\sigma i_1, \sigma_{i_2}, \cdots, \sigma_{i_{\phi+2}}$ de σ pris par p_i tels que :

1. Pour tout indice $k = 1, 2, \cdots, \Phi + 2$, p_i exécute $Send(p_i, \langle p_i, alive \rangle)$ au cours de σ_{i_k}

2. Pour tout indice $k = 2, \cdots, \Phi + 2$, σ_{i_k} est le premier pas de calcul postérieur à $\sigma_{i_{k-1}}$ au cours duquel p_i exécute $Send(p_i, \langle p_i, alive \rangle)$.

3. $\sigma_{i_{\Phi+2}}$ est antérieur à σ_j.

La manière dont fonctionne l'algorithme assure que, pour tout indice $k = 1, 2, \cdots, \Phi + 2$, σ_{i_k} est au moins le n-ième pas de calcul postérieur à $\sigma_{i_{k-1}}$ au cours duquel p_i exécute $Send(-, -)$. On en déduit que la sous-suite de σ

$$\sigma_{i_1} \cdots \sigma_{i_2} \cdots \sigma_{i_{\phi+2}}$$

contient au moins $n(\Phi + 1) + 1$ pas de calcul au cours desquels p_i exécute $Send(-, -)$ et aucun au cours duquel p_j exécute $Send(p_i, -)$.

Comme la Réception/Envoi est séparée, entre deux tels pas de calcul, p_i prend nécessairement un pas de calcul au cours duquel il n'exécute pas $Send(-, -)$. On obtient donc que la suite

$$\sigma_{i_1} \cdots \sigma_{i_2} \cdots \sigma_{i_{\phi+2}}$$

contient au moins $2n(\Phi+1)+1$ pas de calcul pris par p_i. Comme les processus sont Φ-synchrones et que p_j exécute $Send(p_i, \langle p_j, r \rangle)$, on en déduit que

$$\sigma_{i_1} \cdots \sigma_{i_2} \cdots \sigma_{i_{\phi+2}}$$

contient au moins $2n$ pas de calcul pris par p_j, et donc au moins n au cours desquels il exécute $Send(-, -)$. La manière dont fonctionne l'algorithme impose alors que p_j envoie des messages à tous les autres toujours dans le même ordre. Ainsi, parmi ces $2n$ pas de calcul, et donc entre σ_{i_1} et $\sigma_{i_{\Phi+2}}$, il en existe un au cours duquel p_j exécute $Send(p_i, -)$, une contradiction. $\square$

Proposition 2.2.5 *Pour toute exécution de PM dans un système où les processus sont Φ-synchrones, Message-Order est synchrone, les communications sont asynchrones, la Réception/Envoi est séparée, les transmissions se font point-à-point et tel qu'au plus f processus peuvent être fautifs par crash, la collection $(HO_p^{(r)})_{r>0,p\in\Pi}$ satisfait $\mathcal{P}_K^f \wedge \mathcal{P}_{reg}$.*

Démonstration:

Soit σ une exécution de PM dans laquelle au plus f processus sont fautifs par crash. Soit $r > 0$ un entier, et soit p_i un processus qui exécute la ligne 46 au round r. Un tel processus existe puisque, par le Lemme 2.2.3, aucun processus correct ne reste bloqué dans σ.

On démontre tout d'abord que la collection $(HO_p^{(r)})_{r>0,p\in\Pi}$ satisfait $\mathcal{P}_K^f$. Soit $r > 0$ un entier. Supposons qu'il existe un processus p_i qui exécute la ligne 46 au round r. Soit p_j un processus correct dans σ. Le Lemme 2.2.3 implique que p_j exécute $Send(p_i, \langle p_j, r\rangle)$. Le Lemme 2.2.4 implique alors $p_j \in HO_{p_i}^{(r)}$.

Ainsi, à chaque round r, tous les ensembles $HO_{p_i}^{(r)}$ contiennent au moins tous les identifiants des processus corrects dans σ. Puisque, par hypothèse, au plus f processus peuvent être fautifs dans σ, on a bien le résultat voulu.

On démontre à présent que la collection $(HO_p^{(r)})_{r>0,p\in\Pi}$ satisfait $\mathcal{P}_{reg}$. Soit $r > 0$ un entier. Supposons qu'il existe deux processus p_i et p_j tels que p_i exécute la ligne 46 au round $r+1$ et $p_j \in HO_{p_i}^{(r+1)}$. Cela implique que p_j a exécuté $Send(p_i, \langle p_j, r+1\rangle)$ et donc que p_j a exécuté $Send(p_k, \langle p_j, r\rangle)$, pour tout $k = 1, 2, \cdots, n$. Le Lemme 2.2.4 assure alors que, pour tout processus p_k qui a exécuté la ligne 46 au round r, on a $p_j \in HO_{p_k}^{(r)}$.

On a montré que si p_j appartient à un ensemble HO_{p_i} à la fin du round $r+1$, alors il appartient à tous les ensembles HO_{p_i} à la fin du round r.

On en conclut finalement que la collection $(HO_p^{(r)})_{r>0,p\in\Pi}$ satisfait bien $\mathcal{P}_K^f \wedge \mathcal{P}_{reg}$.

□

Récapitulatif

SYSTÈME	PRÉDICAT DE COMMUNICATION	
Synchrone, liens fiables au plus f processus fautifs par omission d'envoi	$\mathcal{P}_K^f$	Charron-Bost *et al.* [17] Gafni [22]
Asynchrone, liens fiables, au plus f processus fautifs par crash	$\mathcal{P}_{HO}^f$	Charron-Bost *et al.* [17] Gafni [22]
Asynchrone, liens fiables, au plus $f < \frac{n}{2}$ processus fautifs par crash initial	$\mathcal{P}_{unif}^f$	Charron-Bost *et al.* [17]
Partiellement synchrone [19] liens finalement fiables, au plus f processus fautifs par crash	$\mathcal{P}_{\diamond sp_unif}^f$	Charron-Bost *et al.* [17]
Synchrone, au plus f erreurs de transmission par omission [42]	$\mathcal{P}_{\mathcal{O}}^f$	Charron-Bost *et al.* [17]
Synchrone, un bloc d'au plus f erreurs de transmission par omission [42]	$\mathcal{P}_{\mathcal{O}}^{f,\,bloc}$	Charron-Bost *et al.* [17]
MO synchrone, Diffusion, liens fiables, au plus f processus fautifs par crash [18]	$\mathcal{P}_{sp_unif}^f = \mathcal{P}_{HO}^f \wedge \mathcal{P}_{sp_unif}$	Section 2.2.1
Communications synchrones, Diffusion, R/E atomique, liens fiables, au plus f processus fautifs par crash [18]	$\mathcal{P}_{sp_unif}^f = \mathcal{P}_{HO}^f \wedge \mathcal{P}_{sp_unif}$	Charron-Bost *et al.* [17]
Processus et communications synchrones, liens fiables, au plus f processus fautifs par crash [18]	$\mathcal{P}_K^f \wedge \mathcal{P}_{reg}$	Charron-Bost *et al.* [17]
Processus synchrones, MO synchrone, liens fiables, au plus f processus fautifs par crash [18]	$\mathcal{P}_K^f \wedge \mathcal{P}_{reg}$	Section 2.2.1

TAB. 2.1 – Systèmes à échanges de messages et prédicats de communication correspondants : cas bénin

$\mathcal{P}_K^f$	$\lvert K\rvert \geq n-f$
$\mathcal{P}_{reg}$	$\forall r>0,\ \forall p\in\Pi\ :\ HO(p,r+1)\subseteq K(r)$
$\mathcal{P}_{HO}^f$	$\forall r>0,\ \forall p\in\Pi\ :\ \lvert HO(p,r)\rvert \geq n-f$
$\mathcal{P}_{unif}^f$	$\exists\Pi_0\subseteq\Pi$ t.q. $\lvert\Pi_0\rvert\geq n-f,\ \forall r>0,\ \forall p\in\Pi\ :\ HO(p,r)=\Pi_0$
$\mathcal{P}_{\diamond\, sp_unif}^f$	$\exists\Pi_0\subseteq\Pi$ t.q. $\lvert\Pi_0\rvert\geq n-f,\ \exists r_0>0,\ \forall r>r_0,\ \forall p\in\Pi\ :\ HO(p,r)=\Pi_0$
$\mathcal{P}_{\mathcal{O}}^f$	$\forall r>0\ :\ \sum_{p\in\Pi}\lvert HO(p,r)\rvert\geq n^2-f$
$\mathcal{P}_{\mathcal{O}}^{f,\,bloc}$	$\forall r>0,\ \left\lvert \begin{array}{l} \lvert K(r)\rvert\geq n-1 \\ \exists\Pi_r\subseteq\Pi \text{ t.q. } \lvert\Pi_r\rvert\geq n-f\ \wedge\ \forall p\in\Pi_r,\ HO(p,r)=\Pi \end{array}\right.$
$\mathcal{P}_{sp_unif}$	$\forall r>0,\ \forall p,q\in\Pi\ :\ HO(p,r)=HO(q,r)$

Il est intéressant de remarquer, ici, que dans le cas où l'on considère qu'au plus f processus peuvent être fautifs par crash, d'une part le prédicat $\mathcal{P}_K^f \wedge \mathcal{P}_{reg}$ correspond aux systèmes dans lesquel les processus sont synchrones et Message-Order est synchrone et aux systèmes dans lesquels les processus et les communications sont synchrones et, d'autre part, que $\mathcal{P}_{sp_unif}^f$ correspond aux systèmes dans lesquels les communications sont synchrones, le mécanisme de transmission est la Diffusion.

2.2.2 Pannes par valeurs

Nous avons décrit dans ce chapitre un grand nombre de systèmes sujets à des pannes bénignes. Dans [38, 30], Pease, Shostak et Lamport ont introduit pour la première fois un nouveau type de pannes, communément appelées *pannes byzantines* en raison de l'énoncé imagé du Problème des Généraux Byzantins étudié par eux dans ces travaux. Un processus est dit *byzantin* s'il peut se comporter de manière arbitraire.

Comme c'est le cas pour les pannes bénignes, on peut distinguer deux types de pannes byzantines : des déviations par rapport à la fonction de transition d'état, ou des déviations par rapport à la fonction d'envoi de messages. Alors qu'il est possible de couvrir toutes les pannes bénignes par des prédicats de communication, les déviations arbitraires par rapport à la fonction de transition dans le cas byzantin, ou *corruptions d'état*, ne peuvent pas être exprimées comme des propriétés sur les communications. Nous nous attachons, dans la suite, à présenter des prédicats de communication correspondant à des systèmes dans lesquels seules des déviations par rapport aux fonctions d'envoi de messages peuvent intervenir.

Systèmes synchrones. Pannes statiques. [30]

Lamport, Shostak et Pease considèrent dans [30] le modèle élémentaire en rounds pour les systèmes synchrones afin d'étudier des systèmes synchrones sujets à des pannes byzantines. Ils supposent qu'à chaque round, chaque processus envoie un message à chacun des autres et que si, au cours d'un round, un processus p ne reçoit pas de message provenant d'un processus q, alors p se comporte comme s'il avait reçu de q une valeur arbitraire fixée. Les erreurs de transmission bénignes sont donc couvertes ici par des erreurs de transmission par valeurs. On peut, dans le modèle HO, adopter la même approche dans le cas des pannes par valeurs en supposant $HO(p,r) = \Pi$, pour tout $r > 0$ et pour tout processus $p \in \Pi$, et en considérant des prédicats sur les collections

$$(SHO(p,r))_{r>0,p\in\Pi}$$

et non sur les collections

$$(HO(p,r); SHO(p,r))_{r>0,p\in\Pi}\,.$$

Considérons un système synchrone comme celui décrit dans [30] dans lequel au plus f processus sont fautifs. À chaque round, chacun des $n - f$

processus corrects est correctement entendu par tous les autres, ce à quoi correspond le prédicat de communication suivant, que nous notons $\mathcal{P}_{SK}^{f}$:

$$\mathcal{P}_{SK}^{f} \ :: \ |SK| \geq n - f$$

Systèmes synchrones. Pannes dynamiques. [42]

Comme nous l'avons précisé, Santoro et Widmayer ([42]) ont défini un modèle en rounds pour l'étude des systèmes distribués, entièrement couvert par le modèle HO. Nous avons, dans la sous-section 2.2.1, présenté deux prédicats de communication qui correspondent à deux systèmes sujets à des pannes bénignes considérés dans [42]. Nous présentons ici des prédicats de communication qui correspondent à des systèmes considérés dans [42] dans le cas particulier des erreurs de transmission par valeurs.

Considérons tout d'abord un système synchrone dans lequel les transmissions sont *continues*, en ce sens qu'à chaque round, chaque processus envoie un message à chacun des autres, les erreurs de transmission sont des corruptions - le message reçu est différent de celui qui a été envoyé - et, à chaque round $r > 0$, on dénombre au plus f, $f \leq n^2$, erreurs de transmission.

Dans un tel système, la continuité des transmissions, combinée avec le fait que le système est synchrone, implique qu'à chaque round $r > 0$, n^2 transmissions sont effectuées. D'autre part, puisqu'au plus f de ces n^2 transmissions effectuées au round r sont incorrectes, le nombre total de transmissions correctes est supérieur à $n^2 - f$, ce qui peut s'écrire

$$\sum_{p \in \Pi} |SHO(p, r)| \geq n^2 - f.$$

Ainsi,

Proposition 2.2.6 *Le prédicat de communication qui correspond aux systèmes synchrones dans lesquels les transmissions sont continues et tels qu'à chaque round, au plus f transmissions sont incorrectes et sont des corruptions, est le prédicat de communication suivant, que nous notons $\mathcal{P}_{\mathcal{C}}^{f}$:*

$$\mathcal{P}_{\mathcal{C}}^{f} \ : \ \forall r > 0, \ \sum_{p \in \Pi} |SHO(p, r)| \geq n^2 - f$$

Dans leur étude, Santoro et Widmayer considèrent aussi des systèmes particuliers dans lesquels, à chaque round, les transmissions incorrectes forment un *bloc*, en ce sens que toutes sont initiées par un même processus, c'est-à-dire des systèmes tels que

$$\forall r > 0, \quad |AS(r)| \leq 1.$$

Considérons un système semblable au précédent mais tel qu'à chaque round, les transmissions incorrectes forment un bloc. Puisque les transmissions sont continues et qu'à chaque round toutes les transmissions incorrectes sont initiées par un unique processus p, tous les autres processus distincts de p sont correctement entendus par tous à ce round. En d'autres termes, on a

$$\forall r > 0, \quad |SK(r)| \geq n - 1.$$

De plus, puisqu'à chaque round r, au plus f des transmissions initiées par p sont incorrectes, au moins $n - f$ processus entendent correctement p à ce round. Ainsi, il existe $\Pi_r \subseteq \Pi$ tel que $|\Pi_r| \geq n - f$ et, pour tout processus q de Π_r, on a $SHO(q, r) = \Pi$.

En résumé,

Proposition 2.2.7 *Le prédicat de communication qui correspond aux systèmes synchrones dans lesquels les transmissions sont continues et tels qu'à chaque round, au plus f transmissions sont incorrectes, sont des corruptions et forment un bloc, est le prédicat de communication suivant, que nous notons $\mathcal{P}_{\mathcal{C}}^{f,bloc}$:*

$$\mathcal{P}_{\mathcal{C}}^{f,bloc} \ : \ \forall r > 0, \left| \begin{array}{c} \exists \Pi_r \subseteq \Pi : \ |\Pi_r| \geq n - f \ \wedge \ \forall p \in \Pi_r, \ SHO(p, r) = \Pi \\ \wedge \\ |SK(r)| \geq n - 1 \end{array} \right.$$

Nous terminons notre étude des systèmes synchrones en considérant un système dans lequel les transmissions sont continues et tel qu'à chaque round, au plus t transmissions sont des corruptions, et au plus f transmissions sont des omissions.

Si à chaque round, les corruptions peuvent être initiées par plus d'un processus, elles ne peuvent l'être que par au plus t distincts. En d'autres termes, pour tout round $r > 0$, la réunion de tous les ensembles $AHO(p, r)$ d'écoute altérés est de cardinal au plus t, ou encore, $|AS(r)| \leq t$. D'autre part, puisqu'au plus f transmissions sont des omissions, on a $\sum_{p \in \Pi} |HO(p, r)| \geq n^2 - f$.

On peut donc énoncer

Proposition 2.2.8 *Le prédicat de communication qui correspond aux systèmes synchrones dans lesquels les transmissions sont continues et tels qu'à chaque round, au plus t transmissions sont des corruptions et au plus f transmissions sont des omissions, est le prédicat de communication suivant, que nous notons $\mathcal{P}_{\mathcal{OC}}^{f,t}$:*

$$\mathcal{P}_{\mathcal{OC}}^{f,t} \ : \ \forall r > 0, \left| \begin{array}{c} \sum_{p \in \Pi} |HO(p, r)| \geq n^2 - f \\ \wedge \\ |AS(r)| \leq t \end{array} \right.$$

Systèmes partiellement synchrones. Dwork et al. ([19]).

Nous avons présenté dans la sous-section précédente le prédicat de communication correspondant aux systèmes partiellement synchrones sujets à des pannes crash décrit dans [19]. Dans ce même travail, Dwork *et al.* décrivent deux types de systèmes partiellement synchrones sujets à des pannes byzantines. Pour chacun des deux systèmes, Dwork *et al.* supposent que, dans toute exécution, au plus f processus peuvent être byzantins et, de plus, qu'il existe un round appelé GST à partir duquel le système se comporte comme s'il était synchrone, en ce sens que tout message envoyé au cours d'un round $r \geq GST$ par un processus correct est reçu au round r. Dans le premier type de systèmes, les messages peuvent être *authentifiés*, en ce sens que chaque processus peut signer un message qu'il envoie et que les signatures ne peuvent pas être falsifiées ou imitées. Dans le deuxième type de systèmes, aucune hypothèse d'authentification n'est faite. Cependant, Dwork *et al.* montre qu'il est possible d'implémenter une primitive de communication qui simule les propriétés vérifiées par le mécanisme d'authentification.

Nous restreignons ici le type de pannes considérées en nous limitant aux déviations par rapport aux fonctions d'envoi. Nous imposons, de plus, que

tous les processus envoient un message à tous les autres à chaque round. L'hypothèse selon laquelle au plus f processus sont fautifs implique, d'une part, que la partie altérée AS de toute exécution est de cardinal au plus égal à f, ce à quoi correspond le prédicat suivant, que nous avons présenté dans le Chapitre 1 :

$$\mathcal{P}_f^{perm} \ :: \quad |AS| \leq f$$

D'autre part, puisqu'au plus f processus sont fautifs dans chaque exécution, tous les processus corrects sont entendus par tous à partir du round r_0 considéré précédemment, ce à quoi correspond le prédicat suivant, que nous notons $\mathcal{P}_{\diamond SK}^f$:

$$\mathcal{P}_{\diamond SK}^f \ :: \quad \exists r_0 > 0 \ : \ \left| \bigcap_{r \geq r_0} SK(r) \right| \geq n - f$$

Ces trois observations nous conduisent à énoncer le résultat suivant :

Proposition 2.2.9 *Le prédicat de communication qui, dans le cas particulier des erreurs de transmission par valeurs, correspond aux deux types de systèmes partiellement synchrones décrits dans [19], dans lesquels on suppose que chaque processus envoie un message à chacun des autres à chaque round et tels que, dans toute exécution, au plus f processus sont fautifs est le prédicat suivant :*

$$\mathcal{P}_f^{perm} \ \wedge \ \mathcal{P}_{\diamond SK}^f$$

Systèmes asynchrones

Nous donnons ici deux prédicats correspondant à des systèmes asynchrones sujets à des pannes de transmission par valeurs.

On considère tout d'abord un système asynchrone dont les exécutions évoluent en rounds sujet à des pannes dynamiques et transitoires comme ceux qui sont décrits dans [13]. À chaque round, chaque processus reçoit au plus f messages différents de ceux qui auraient dû lui être envoyés. Le prédicat correspondant est le suivant, que nous notons $\mathcal{P}_f$:

$$\mathcal{P}_f \ :: \quad \forall r > 0, \ \forall p \in \Pi \ : \ |AHO(p, r)| \leq f$$

Dans le cas où les pannes sont statiques, le prédicat correspondant est $\mathcal{P}_f^{perm}$, que nous avons évoqué dans le cas des systèmes partiellement synchrones :

$$\mathcal{P}_f^{perm} :: \ |AS| \leq f$$

Notons que, pour f fixé, $\mathcal{P}_f^{perm}$ implique $\mathcal{P}_f$.

Récapitulatif

SYSTÈME	PRÉDICAT DE COMMUNICATION
Synchrone, au plus f processus fautifs [30]	$\mathcal{P}_{SK}^f$
Synchrone, transmissions continues, au plus f corruptions [42]	$\mathcal{P}_{\mathcal{C}}^f$
Synchrone, transmissions continues, un bloc d'au plus f corruptions [42]	$\mathcal{P}_{\mathcal{C}}^{f,\,bloc}$
Synchrone, transmissions continues, au plus t corruptions et f omissions par round [42]	$\mathcal{P}_{\mathcal{OC}}^{f,\,t}$
Partiellement synchrone, au plus f processus fautifs [19]	$\mathcal{P}_f^{perm} \wedge \mathcal{P}_{\diamond SK}^f$
Asynchrone, pannes dynamiques et transitoires [13]	$\mathcal{P}_f$
Asynchrone, pannes statiques et transitoires [13]	$\mathcal{P}_f^{perm}$

TAB. 2.2 – Systèmes à échanges de messages et prédicats de communication correspondants : pannes par valeurs.

2.3 Prédicats pour la résolution de Consensus dans le cas des pannes bénignes

Nous avons, dans le présent chapitre, étudié un grand nombre de types de systèmes et avons déterminé les prédicats de communication qui, selon nous, capturent le mieux les propriétés de vivacité et de sûreté des communications

que chacun d'eux peut garantir. Une autre question consiste à déterminer quelles sont les propriétés qui doivent être vérifiées par un système pour qu'un problème donné soit résoluble.

Nous considérons ici le problème du Consensus, que nous avons présenté dans le Chapitre 1, dans lequel les processus doivent se mettre d'accord sur une valeur unique. Nous rappelons tout d'abord une caractérisation des machines HO qui résolvent Consensus dans le cas des pannes bénignes, établie par Charron-Bost et Schiper dans [17]. Dans un second temps, nous mettons en perspective ce résultat avec les différents prédicats de communication que nous avons introduits dans ce chapitre.

2.3.1 Caractérisation des machines HO qui résolvent Consensus

Dans [17], Charron-Bost et Schiper se sont attachés à déterminer les prédicats des machines HO qui résolvent Consensus. En se restreignant aux prédicats à partir desquels il est possible de simuler des rounds dont le noyau est non-vide, ils sont arrivés à la conclusion que la résolution de Consensus nécessitait "un accord implicite permanent sur les ensembles d'écoute".
On rappelle les prédicats de communication

$$\mathcal{P}_{nekrounds} \ :: \ \forall r > 0, \ K(r) \neq \emptyset,$$

et

$$\mathcal{P}_{sp_unif} \ :: \ \forall r > 0, \ \forall p, q \in \Pi, \quad HO(p, r) = HO(q, r).$$

Soit $\mathcal{P}^{n-1}_{sp_unif}$ le prédicat qui garantit qu'à chaque round, les ensembles d'écoute sont non-vides et tous égaux, c'est-à-dire le prédicat $\mathcal{P}_{sp_unif} \wedge \mathcal{P}^{n-1}_{HO}$, avec

$$\mathcal{P}^{n-1}_{HO} \ :: \ \forall r > 0, \ \forall p \in \Pi, \ |HO(p, r)| \geq 1.$$

Le théorème de caractérisation de Charron-Bost et Schiper s'énonce alors ainsi :

Théorème 2.3.1 ([17])
Soit $\mathcal{P}$ un prédicat de communication tel que $\mathcal{P} \succeq \mathcal{P}_{nekrounds}$. Les assertions suivantes sont équivalentes :

1. *Il existe un algorithme $\mathcal{A}$ tel que la machine $(\mathcal{A}, \mathcal{P})$ résout Consensus ;*

2. $\mathcal{P} \succeq \mathcal{P}^{n-1}_{sp_unif}$.

Dans les deux sections suivantes, nous revenons sur les différents systèmes sujets à des pannes bénignes étudiés dans ce chapitre et utilisons le théorème précédent pour, d'une part, déduire des relations entre certains des prédicats que nous avons introduits et, d'autre part, retrouver un certain nombre de résultats, établis dans les modèles classiques, relatifs à la résolubilité de Consensus.

2.3.2 Consensus et systèmes à mémoire partagée

Les premiers objets partagés que nous avons présentés sont les registres SWMR et Atomic-Snapshot dont nous avons montré qu'ils correspondaient respectivement aux prédicats $\mathcal{P}_{sym}$ et $\mathcal{P}^*_{\mathcal{RD}} = \mathcal{P}_{\mathcal{RD}} \wedge \mathcal{P}_{ref}$, où :

$$\mathcal{P}_{sym} \ :: \ \forall r > 0, \ \forall p, q \in \Pi \ \ : \ \ p \in HO(q,r) \ \vee \ q \in HO(p,r)$$

$$\mathcal{P}_{ref} \ :: \ \ \forall r > 0, \ \forall p \in \Pi \ \ : \ \ p \in HO(p,r)$$

$$\mathcal{P}_{\mathcal{RD}} \ :: \ \ \forall r > 0, \ \forall p, q \in \Pi \ : \ HO(p,r) \subseteq HO(q,r) \ \vee \ HO(q,r) \subseteq HO(p,r)$$

Dans [25], Herlihy montre qu'il est impossible de résoudre Consensus dans un système où les processus ne disposent que de ces registres. Nous avons montré dans la Section 2.1 que $\mathcal{P}_{sym}$ et $\mathcal{P}^*_{\mathcal{RD}}$ étaient équivalents, et tous deux strictement plus forts que $\mathcal{P}_{nekrounds}$. Le résultat d'impossibilité d'Herlihy, combiné avec le Théorème 2.3.1, nous permet donc d'établir le résultat suivant :

Théorème 2.3.2 *il est impossible de translater les prédicats* $\mathcal{P}_{sym}$ *et* $\mathcal{P}^*_{\mathcal{RD}}$ *en* $\mathcal{P}^{n-1}_{sp_unif}$. *En d'autres termes :*

$$\mathcal{P}_{sym} \not\succeq \mathcal{P}^{n-1}_{sp_unif} \quad et \quad \mathcal{P}^*_{\mathcal{RD}} \not\succeq \mathcal{P}^{n-1}_{sp_unif}.$$

Nous nous sommes ensuite intéressé à quatre objets, *Test&Set*, *Swap*, *Fetch&Add* et *Compare&Swap*, de la classe des objets dits Read-Modify-Write. Nous avons présenté le prédicat $\mathcal{P}_2^{\mathcal{CN}}$ et avons montré qu'il peut être garanti par tout système sujet à des pannes crash dans lequel les processus communiquent *via* une infinité de registres SWMR et ont accès à une infinité d'objets partagés de l'un des trois premiers types (Proposition 2.1.23), où

Algorithme 2.11 Algorithme $\mathcal{A}$, code du processus p_i, $i = 0, 1$.

```
1:  Initialisation :
2:    NewHO_{p_i} ⊆ Π ; initialement ∅

3:  Round r :

4:    S_p^r :
5:      Envoyer ⟨p_i⟩ à tous

6:    T_p^r :
7:      if HO(p_i, r) = {p_i; p_j} then
8:        NewHO_{p_i} := {p_j}
9:      else
10:         NewHO_{p_i} := HO(p_i, r)
```

$$\mathcal{P}_2^{\mathcal{CN}} \ :: \ \forall r > 0, \ \exists q \in \Pi, \ \begin{cases} HO(q, r) = \{q\} \\ \forall p \in \Pi \setminus \{q\}, \quad q \in HO(p, r) \end{cases}$$

Considérons un système composé d'un ensemble Π de deux processus p_0 et p_1. Nous allons démontrer que, dans ce cas, $\mathcal{P}_2^{\mathcal{CN}}$ se translate en $\mathcal{P}_{sp_unif}^{n-1}$, i.e., $\mathcal{P}_2^{\mathcal{CN}} \succeq \mathcal{P}_{sp_unif}^{n-1}$.

Nous présentons l'algorithme $\mathcal{A}$ sur Π, donné comme Algorithme 2.11. À chaque round, chaque processus p_i met à jour une variable $NewHO_{p_i}$ qui représente son ensemble d'écoute simulé.

Proposition 2.3.3 *L'algorithme $\mathcal{A}$ est une translation en 1 round de $\mathcal{P}_2^{\mathcal{CN}}$ en $\mathcal{P}_{sp_unif}^1$, et donc*

$$\mathcal{P}_2^{\mathcal{CN}} \succeq_1 \mathcal{P}_{sp_unif}^1.$$

Démonstration: Considérons une exécution e de la machine $(\mathcal{A}, \mathcal{P}_2^{\mathcal{CN}})$ pour $\Pi = \{p_0; p_1\}$.

Le code (lignes 8 et 10) implique qu'à chaque round r et pour chaque processus p_i, i$= 0, 1$, on a $NewHO_{p_i}^{(r)} \subseteq HO(p_i, r)$. Par conséquent, la condition E1 de la définition d'une translation est bien satisfaite dans e.

Montrons à présent que la condition E2 est, elle aussi, satisfaite dans e, c'est-à-dire que la collection $(NewHO_{p_i})_{r>0, i=1,2}$ induite par e satisfait $\mathcal{P}_{sp_unif}^1$. Soit $r > 0$ un round. Puisque $\mathcal{P}_2^{\mathcal{CN}}$ est satisfait dans e, il existe un

indice i, $i \in \{0,1\}$, tel que $HO(p_i, r) = \{p_i\}$ et $p_i \in HO(p_{i+1(mod2)}, r)$. Le processus p_i exécute donc la ligne 10, et ainsi

$$NewHO_{p_i}^{(r)} = HO(p_i, r) = p_i.$$

Si $|HO(p_{i+1(mod2)}, r)| = 2$, le processus $p_{i+1(mod2)}$ exécute la ligne 8, sinon il exécute la ligne 10. Dans les deux cas, on a

$$NewHO_{p_{i+1(mod2)}}^{(r)} = \{p_i\}.$$

On en conclut donc

$$NewHO_{p_i}^{(r)} = NewHO_{p_{i+1(mod2)}}^{(r)} = \{p_i\}.$$

La collection $(NewHO_{p_i})_{r>0,i=1,2}$ satisfait donc bien $\mathcal{P}^1_{sp_unif}$, ce qui achève de démontrer que $\mathcal{A}$ est une translation en 1 round de $\mathcal{P}_2^{\mathcal{CN}}$ en $\mathcal{P}^1_{sp_unif}$. $\square$

On a démontré $\mathcal{P}_2^{\mathcal{CN}} \succeq \mathcal{P}^1_{sp_unif}$. Le Théorème 2.3.1 assure alors qu'il existe un algorithme $\mathcal{B}$ pour Π tel que la machine $(\mathcal{B}, \mathcal{P}_2^{\mathcal{CN}})$ résout Consensus. On retrouve ainsi le résultat d'Herlihy [25] :

Théorème 2.3.4 ([25]) *Le problème du Consensus est résoluble dans un système composé de deux processus, dont l'un peut être fautif par crash, qui communiquent via des registres SWMR et qui ont accès à un objet partagé muni de l'une des trois fonctions Test&Set, Fetch&Add ou Swap.*

Nous avons ensuite montré que tout système, composé de n processus, sujet à des pannes crashs, dans lequel les processus communiquent *via* une infinité de registres SWMR et ont accès à une infinité d'objets partagés munis de la fonction *Compare&Swap*, peut garantir le prédicat $\mathcal{P}_n^{\mathcal{CN}}$ suivant :

$$\mathcal{P}_n^{\mathcal{CN}} \ :: \ \forall r > 0, \ \exists q \in \Pi \ : \ \forall p \in \Pi, \quad HO(p, r) = \{q\}.$$

Il est clair que $\mathcal{P}_n^{\mathcal{CN}} \Rightarrow \mathcal{P}^{n-1}_{sp_unif}$. Le Théorème 2.3.1 implique alors un autre résultat établi par Herlihy [25] :

Théorème 2.3.5 ([25]) *Le problème du Consensus est résoluble dans un système sujet à des pannes crash, composé de n processus qui communiquent* via *des registres SWMR et ont accès à un objet partagé muni de la fonction Compare&Swap.*

Nous avons terminé notre exploration partielle des objets partagés avec le cas des files FIFO et FIFO augmentées.

Nous avons montré que tout système sujet à des pannes crash dans lequel les processus ont accès à une infinité de files FIFO (resp. FIFO augmentée) peut garantir le prédicat $\mathcal{P}_2^{\mathcal{CN}}$ (resp. $\mathcal{P}_n^{\mathcal{CN}}$). Ainsi, les deux théorèmes précédents restent valables respectivement pour l'un et l'autre de ces systèmes.

Théorème 2.3.6 ([25]) *Le problème du Consensus est résoluble dans un système composé de deux processus, dont l'un peut être fautif par crash, qui communiquent via des registres SWMR et qui ont accès à une file FIFO.*

De plus, si le système est doté d'une file FIFO augmentée, Consensus est résoluble sans restriction sur la taille du système.

2.3.3 Consensus et systèmes à échanges de messages

Comme nous l'avons précisé dans la Section 2.2.1, Dolev *et al.* ont défini cinq paramètres de synchronisme qui déterminent, selon eux, un type particulier de systèmes distribués. Chaque paramètre pouvant prendre deux valeurs, favorable ou défavorable, ils ont donc décrit trente-deux différents types de systèmes. Dans chacun d'entre eux, Dolev *et al.* ont étudié la résolubilité du problème du Consensus et ont déterminé le nombre maximum de processus pouvant être fautifs dans une exécution d'un algorithme résolvant le Consensus. Ils ont, de plus, exhibé quatre types "minimaux" de systèmes, en ce sens que le problème du Consensus est résoluble dans chacun d'eux en présence d'un nombre arbitraire de processus fautifs, mais que tout changement de la valeur d'un paramètre de favorable à défavorable rend le Consensus non-résoluble en présence d'une ou deux pannes, selon les cas. Nous nous sommes attachés, dans la Section 2.2.1, à présenter les prédicats de communication qui correspondent à ces quatre types de systèmes. La Table 2.3 récapitule les résultats obtenus alors.

Comme on le remarque aisément, le prédicat $\mathcal{P}_{sp_unif}^{f}$, qui correspond aux deux premiers types de systèmes considérés, implique directement $\mathcal{P}_{sp_unif}^{n-1}$ si $f \leq n-1$. Par le Théorème 2.3.1, on retrouve le résultat de [18] qui assure que le Consensus est résoluble dans ces deux types de systèmes en présence d'un nombre arbitraire de processus fautifs.

SYSTÈME	PRÉDICAT	
MO synchrone, Diffusion, liens fiables au plus f processus fautifs par crash	$\mathcal{P}^f_{sp_unif}$	Section 2.2.1
Communications synchrones, Diffusion, R/E atomique, liens fiables, au plus f processus fautifs par crash	$\mathcal{P}^f_{sp_unif}$	Charron-Bost *et al.* [17]
Processus et communications synchrones, liens fiables au plus f processus fautifs par crash	$\mathcal{P}^f_K \wedge \mathcal{P}_{reg}$	Charron-Bost *et al.* [17]
Processus synchrones, Message-Order synchrone, liens fiables au plus f processus fautifs par crash	$\mathcal{P}^f_K \wedge \mathcal{P}_{reg}$	Section 2.2.1

TAB. 2.3 – Systèmes "minimaux" de Dolev *et al.* [18] et prédicats de communication correspondants.

D'autre part, Charron-Bost et Schiper [17] ont présenté une translation, donnée alors comme "*Translation for space uniformity*", en $f+1$ rounds de $\mathcal{P}^f_K$ en $\mathcal{P}^f_K \wedge \mathcal{P}_{sp_unif}$. Ainsi,

$$\mathcal{P}^f_K \succeq \mathcal{P}^f_K \wedge \mathcal{P}_{sp_unif},$$

et donc

$$\mathcal{P}^f_K \wedge \mathcal{P}_{reg} \succeq \mathcal{P}^f_K \wedge \mathcal{P}_{sp_unif}.$$

Comme, de plus,

$$\mathcal{P}^f_K \wedge \mathcal{P}_{sp_unif} \quad \Rightarrow \quad \mathcal{P}^f_{sp_unif},$$

on obtient $\mathcal{P}^f_K \wedge \mathcal{P}_{reg} \succeq \mathcal{P}^f_{sp_unif}$. En supposant $f \leq n-1$, on en conclut $\mathcal{P}^f_K \wedge \mathcal{P}_{reg} \succeq \mathcal{P}^{n-1}_{sp_unif}$. Par le Théorème 2.3.1, puisque $\mathcal{P}^f_K \wedge \mathcal{P}_{reg}$ est le prédicat correspondant aux deux derniers types de systèmes, on en déduit que le Consensus est résoluble dans chacun d'eux en présence d'un nombre arbitraire de processus fautifs. On retrouve ainsi un des résultats de [18].

Chapitre 3

Solutions algorithmiques pour Consensus

Nous nous intéressons, dans le présent chapitre, à la résolubilité du Consensus par des machines HO dans un contexte d'erreurs de transmission par valeurs. Contrairement au cas des pannes bénignes où tout message reçu est identique à celui qui a été envoyé, l'information transmise peut, ici, être altérée. Un processus q est susceptible de recevoir de la part d'un processus p un message différent de celui que p aurait dû lui envoyer. La clause d'Intégrité considérée jusqu'à présent, qui requiert que toute valeur de décision soit la valeur initiale d'un processus, apparaît dès lors beaucoup trop forte. Cette aggravation du type de pannes considéré a conduit les chercheurs à étudier une version affaiblie du Consensus. Les clauses d'Accord, d'Irrévocabilité et de Terminaison restent inchangées, tandis que la clause d'Intégrité est modifiée de la manière suivante :

Intégrité Si toutes les valeurs initiales sont égales à une même valeur v, alors v est la seule valeur de décision possible.

Dans [17], Charron-Bost et Schiper présentent de nombreuses machines HO qui résolvent le Consensus dans le cas bénin, s'inspirant d'algorithmes connus, coordonnés ou non, destinés à être exécutés dans des systèmes asynchrones ou partiellement synchrones, et décrivant de nouvelles solutions. Dans ce chapitre, nous transformons les algorithmes présentés dans [17] et déterminons des prédicats de communication tels que les machines HO obtenues résolvent le Consensus en présence d'erreurs de transmission par valeurs.

3.1 *UniformVoting*, $\mathcal{U}_{T,M,E}$ et $\mathcal{UV}_{vf}$: Consensus avec des communications vivaces.

La plupart des algorithmes classiques de Consensus dans le cas bénin sont censés être exécutés dans des systèmes où une stricte minorité des processus sont susceptibles d'être fautifs. Cette hypothèse implique que l'on considère des exécutions présentant une certaine vivacité des communications, puisque l'on peut garantir dans de tels systèmes que tous les ensembles d'écoute soient strictement majoritaires. Par exemple, la correction de l'algorithme *UniformVoting* de [17] repose sur cette hypothèse de vivacité des communications.

3.1.1 Algorithme *UniformVoting*

L'une des premières approches utilisées pour contourner le résultat de Fischer *et al.*[21] dans le cas bénin a été d'avoir recours à des algorithmes probabilistes [40], [32]. On simule un comportement non-déterministe des processus en leur donnant accès à une source d'information aléatoire à valeurs dans un ensemble Ω fini muni d'une mesure de probabilité P et en étendant la définition d'une fonction de transition de manière qu'elle prenne en argument supplémentaire un élément de Ω. Etant donné un algorithme $\mathcal{A}$ qui autorise les processus à consulter cette source aléatoire, on munit l'ensemble $\mathcal{E}_{\mathcal{A}}$ de ses exécutions d'une autre mesure de probabilité $P_{\mathcal{A}}$, fonction de P.

Dès lors, on ne requièrt plus que chaque exécution de $\mathcal{A}$ satisfasse la clause de Terminaison, mais que l'ensemble $\mathcal{T}_{\mathcal{A}}$ des exécutions qui la satisfont soit tel que

$$P_{\mathcal{A}}(\mathcal{T}_{\mathcal{A}}) = 1.$$

A notre connaissance, le premier algorithme probabiliste de Consensus est dû à Ben-Or [6] et a été décrit pour des systèmes sujets à un nombre f strictement minoritaire de processus fautifs.

Les exécutions sont décomposées en rounds, chacun d'entre eux consistant en deux étapes. Lors de la première, chaque processus p envoie à tous les autres sa préférence, initialement égale à sa propre valeur initiale, et attend de recevoir $n - f$ messages. Si p reçoit une majorité stricte de messages

contenant une même valeur v , alors il effectue un vote pour v. Dans le cas contraire, il vote pour “ ? ”.

Lors de la seconde étape, chaque processus p envoie son vote à tous les autres. Si p reçoit au moins un message contenant une valeur $v \neq ?$, alors il adopte v comme sa nouvelle préférence. Si, de plus, p reçoit plus que f tels messages[1], alors il décide immédiatement v. Enfin, si p ne reçoit que des messages contenant ?, il tire à pile ou face et adopte le résultat comme nouvelle préférence.

L'algorithme *UniformVoting*, donné comme Algorithme 3.1, peut être vu comme la version HO et déterministe de l'algorithme de Ben-Or. Ses exécutions sont décomposées en phases de deux rounds, chacune correspondant à un round de l'algorithme de Ben-Or.

Algorithme 3.1 L'algorithme *UniformVoting*

Initialisation :
  $x_p \in V$; initialement $x_p = v_p$ $\{v_p$ est la valeur initiale de $p\}$
  $vote_p \in V$; initialement $vote_p = ?$

Round $2\phi - 1$:
  $\boldsymbol{S_p^r}$ **:**
    Envoyer $\langle x_p \rangle$ à tous

  $\boldsymbol{T_p^r}$ **:**
    $x_p :=$ la plus petite valeur reçue
    if toutes les valeurs reçues sont égales à une valeur v **then**
      $vote_p := v$

Round 2ϕ :
  $\boldsymbol{S_p^r}$ **:**
    Envoyer $\langle x_p; vote_p \rangle$ à tous

  $\boldsymbol{T_p^r}$ **:**
    if au moins un message $\langle *; v \rangle$ est reçu **then**
      $x_p := v$
    else
      $x_p :=$ la plus petite valeur w telle que $\langle w; ? \rangle$ a été recu
    if tous les messages reçus sont égaux à $\langle *; v \rangle$ avec $v \neq ?$ **then**
      DECIDE(v)
    $vote_p := ?$

Charron-Bost et Schiper ont montré que si, à chaque round, les ensembles d'écoute sont deux à deux d'intersection non-vide, alors les clauses d'Intégrité et d'Accord sont satisfaites. Notons que, sous les hypothèses de Ben-Or,

[1]On remarque que si p reçoit deux messages contenant respectivement v et v' toutes deux différentes de ?, alors la règle de vote assure que v et v' sont égales.

les ensembles d'écoute de tous les processus sont strictement majoritaires à chaque étape et sont donc deux à deux d'intersection non-vide. Charron-Bost et Schiper ont de plus démontré que le Terminaison, qui était assurée avec probabilité 1 dans [6], nécessite l'existence d'un unique round uniforme, ce à quoi correspond, par exemple, le prédicat suivant :

$$\forall r > 0,\ \forall p, q \in \Pi\ :\quad HO(p,r) \cap HO(q,r) \neq \emptyset$$
$$\wedge$$
$$\exists r_0 > 0,\ \exists \Pi_0 \neq \emptyset,\ \forall p \in \Pi\ :\quad HO(p,r_0) = \Pi_0$$

3.1.2 Algorithme $\mathcal{U}_{T,M,E}$

Nous présentons ici notre premier algorithme, nommé $\mathcal{U}_{T,M,E}$ et donné comme Algorithme 3.2. La démarche que nous adoptons pour décrire cet algorithme consiste à paramétrer les différents seuils de réception que l'on rencontre dans *UniformVoting*.

Algorithme 3.2 L'algorithme $\mathcal{U}_{T,M,E}$

```
1:  Initialisation :
2:    x_p ∈ V ; initialement x_p = v_p        {v_p est la valeur initiale de p}
3:    vote_p ∈ V ∪ {?} ; initialement vote_p = ?
4:    v_0 ∈ V

5:  Round 2φ − 1 :
6:    S_p^r :
7:      Envoyer ⟨x_p⟩ à tous

8:    T_p^r :
9:      x_p := la plus petite des valeurs les plus fréquemment reçues

10:     if > T valeurs reçues sont égales à une valeur v then
11:       vote_p := v

12: Round 2φ :
13:   S_p^r :
14:     Envoyer ⟨vote_p⟩ à tous

15:   T_p^r :
16:     if ≥ M messages ⟨v⟩ sont reçus then
17:       x_p := v
18:     if > E messages reçus sont égaux à ⟨v⟩ avec v ≠ ? then
19:       DECIDE(v)
20:     vote_p :=?
```

Le code de $\mathcal{U}_{T,M,E}$ est censé s'exécuter sous les prédicats $\mathcal{P}_\alpha$ et $\mathcal{P}_{HO}^A$, donnés en Figure 3.1. Le premier assure qu'à chaque round les ensembles

$$\mathcal{P}_\alpha \ :: \ \forall r > 0, \ \forall p \in \Pi \ : \ |AHO(p,r)| \leq \alpha \qquad (\alpha \leq n)$$

$$\mathcal{P}_{HO}^A \ :: \ \forall r > 0, \ \forall p \in \Pi \ : \ |HO(p,r)| \geq A$$

FIG. 3.1 – Prédicats $\mathcal{P}_\alpha$ et $\mathcal{P}_{HO}^A$

d'écoute altérés sont de cardinal au plus α, où α est un entier plus petit que n. Le second, quant à lui, garantit qu'à chaque round les ensembles d'écoute sont de cardinal au moins égal à A.

Preuve de correction

Avant de commencer notre preuve de correction, nous rappelons quelques notations. Pour toute variable x_p, locale au processus p, on note $x_p^{(r)}$ la valeur de x_p à la fin du round r et $x_p^{(0)}$ sa valeur initiale. Pour toute valeur v de V, pour tout processus p de Π, et pour tout round $r > 0$, nous définissons les ensembles $Q_p^r(v)$ et $R_p^r(v)$ comme suit :

$$R_p^r(v) := \{q \in \Pi \ : \ \vec{\mu}_p^r[q] = v\}$$
$$Q_p^r(v) := \{q \in \Pi \ : \ S_q^r(s_q, p) = v\}.$$

où s_q désigne l'état du processus q au début du round r. L'ensemble $R_p^r(v)$ (resp. $Q_p^r(v)$) représente l'ensemble des processus desquels p reçoit la valeur v (resp. qui devraient envoyer la valeur v à p) au round r. Puisque dans l'algorithme $\mathcal{U}_{T,M,E}$, à chaque round, chaque processus envoie le même message à tous les autres, les ensembles $Q_p^r(v)$ ne dépendent pas de p ; on peut donc les noter $Q^r(v)$ sans ambiguïté.

Nos deux premiers lemmes sont des résultats techniques élémentaires.

Lemme 3.1.1 *Pour tout processus p, pour toute valeur v, et pour tout round $r > 0$, on a :*

$$|R_p^r(v)| \leq |Q^r(v)| + |AHO(p,r)|$$

Démonstration: Supposons qu'un processus p reçoive un message contenant une valeur v et provenant d'un processus q à un round $r > 0$. On distingue deux cas :

– soit $S_q^r(s_q, p) = v$ et donc, par définition, $q \in Q^r(v)$,

– soit le message de q à destination de p a été corrompu et donc $q \in AHO(p, r)$.

Par conséquent

$$R_p^r(v) \subseteq Q^r(v) \cup AHO(p, r),$$

ce qui implique

$$|R_p^r(v)| \leq |Q^r(v)| + |AHO(p, r)|.$$

□

Lemme 3.1.2 *Si* Π_1 *et* Π_2 *sont deux sous-ensembles d'un ensemble* Π *de cardinal* n*, alors*

$$|\Pi_1| + |\Pi_2| \geq n + (\alpha + 1) \quad \Rightarrow \quad |\Pi_1 \cap \Pi_2| \geq \alpha + 1$$

Démonstration: Puisque Π_1 et Π_2 sont deux sous-ensembles de Π et que $|\Pi| = n$, on a

$$|\Pi_1 \cup \Pi_2| \leq n.$$

Comme, d'autre part

$$|\Pi_1 \cap \Pi_2| = |\Pi_1| + |\Pi_2| - |\Pi_1 \cup \Pi_2|,$$

on en déduit

$$|\Pi_1 \cap \Pi_2| \geq |\Pi_1| + |\Pi_2| - n.$$

Si on suppose

$$|\Pi_1| + |\Pi_2| \geq n + (\alpha + 1),$$

alors il s'ensuit

$$|\Pi_1 \cap \Pi_2| \geq \alpha + 1.$$

□

Nous donnons, à présent, une condition suffisante sur le paramètre E pour que la règle de décision soit déterministe ou, en d'autres termes, pour assurer qu'à chaque round, un processus puisse décider au plus une valeur.

Lemme 3.1.3 *Si $E \geq \frac{n}{2}$ alors, dans toute exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^A_{HO})$, au plus une valeur par processus et par round peut être décidée.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^A_{HO})$. Supposons, par contradiction, qu'il existe un processus p et un round $r > 0$ tels que la garde de la ligne 18 soit vraie pour deux valeurs distinctes v et v' de V. Le code implique alors que $|R^r_p(v)| > E$ et $|R^r_p(v')| > E$. Puisque les valeurs v et v' sont distinctes et qu'à chaque round, tous les processus reçoivent au plus un message provenant de chacun des autres, les ensembles $R^r_p(v)$ et $R^r_p(v')$ sont disjoints. Par conséquent

$$|R^r_p(v) \cup R^r_p(v')| = |R^r_p(v)| + |R^r_p(v')|.$$

Si $E \geq \frac{n}{2}$, alors on a $|R^r_p(v) \cup R^r_p(v')| > n$, ce qui contredit le fait que $R^r_p(v)$ et $R^r_p(v')$ sont des sous-ensembles de Π. □

Notre lemme suivant démontre que si $T \geq \frac{n}{2} + \alpha$, alors l'argument de base qui garantit que, dans toute exécution de *UniformVoting*, à chaque round, au plus une valeur différente de ? est votée par les processus est toujours valable.

Lemme 3.1.4 *Si $T \geq \frac{n}{2} + \alpha$ alors, dans toute exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^A_{HO})$, si un processus p vote pour une valeur $v \neq ?$ à un round $r > 0$, alors tous les processus votent pour v ou ? à ce round.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^A_{HO})$. Supposons, par contradiction, qu'à un round r de ρ deux processus p et q votent pour v et v' respectivement, avec $v \neq v'$.
Le code implique que

$$|R^r_p(v)| > T \qquad \text{et} \qquad |R^r_q(v')| > T.$$

Sous le prédicat $\mathcal{P}_\alpha$, le Lemme 3.1.1 assure alors que

$$|Q^r(v)| > T - \alpha \qquad \text{et} \qquad |Q^r(v)| > T - \alpha.$$

Puisqu'on a supposé que v et v' étaient distinctes, on en déduit que $Q^r(v)$ et $Q^r(v)$ sont des ensembles disjoints, et donc

$$|Q^r(v) \cup Q^r(v')| = |Q^r(v)| + |Q^r(v')|.$$

Si $T \geq \frac{n}{2} + \alpha$, on obtient

$$|Q^r(v) \cup Q^r(v')| > n,$$

une contradiction puisque $Q^r(v)$ et $Q^r(v')$ sont des sous-ensembles de Π. $\square$

Nous démontrons à présent que, sous certaines conditions relatives aux paramètres T, M, E et A, la décision d'un processus à un round r force les autres processus à adopter la valeur décidée.

Lemme 3.1.5 *Si les paramètres T, M, E et A vérifient les inéquations :*

$$T \geq \frac{n}{2} + \alpha \tag{3.1}$$

$$M \geq \alpha + 1 \tag{3.2}$$

$$E + A \geq n + M + 2\alpha, \tag{3.3}$$

alors dans toute exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^A_{HO})$, si un processus p décide une valeur v à un round r, alors tout processus q affecte la valeur v à sa variable x_q à ce round.

Démonstration: Soit ρ une exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^A_{HO})$ telle qu'à un round r, un processus p décide une valeur v. Soit q un processus quelconque de Π.

Le code implique alors $|R^r_p(v)| > E$. Sous $\mathcal{P}_\alpha$, le Lemme 3.1.1 assure $|Q^r(v)| > E - \alpha$. Sous $\mathcal{P}^A_{HO}$, on en déduit

$$|Q^r(v)| + |HO(q,r)| > E - \alpha + A,$$

et donc

$$|Q^r(v)| + |SHO(q,r)| > E + A - 2\alpha.$$

Si les paramètres E, M et A vérifient les inéquations (3.2) et (3.3), on obtient

$$|Q^r(v)| + |SHO(q,r)| \geq n + (\alpha + 1).$$

Le Lemme 3.1.2 assure alors

$$|Q^r(v) \cap SHO(q,r)| \geq \alpha + 1,$$

ce qui, puisque $Q^r(v) \cap SHO(q,r) \subseteq R_q^r(v)$, implique

$$|R_q^r(v)| \geq \alpha + 1. \tag{3.4}$$

D'autre part, si $T \geq \frac{n}{2} + \alpha$ alors, sous $\mathcal{P}_\alpha$, le Lemme 3.1.4 assure que, pour toute valeur $v' \neq v$, l'ensemble $Q^r(v')$ est vide et ainsi, par le Lemme 3.1.1, on a

$$\forall v' \neq v \quad : \quad |R_q^r(v')| \leq \alpha. \tag{3.5}$$

La combinaison de (3.4) et (3.5) implique finalement que q affecte la valeur v à sa variable x_q au round r. □

A partir des lemmes précédents, nous donnons une condition suffisante sur les paramètres T, M, E et A pour que toute exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}_{HO}^A)$ satisfasse la clause d'Accord du Consensus.

Proposition 3.1.6 (Accord) *Si les paramètres T, M, E et A vérifient les inéquations*

$$n > T \geq \frac{n}{2} + \alpha \tag{3.6}$$

$$n \geq M > \alpha \tag{3.7}$$

$$n > E \geq \frac{n}{2} \tag{3.8}$$

$$n \geq A \geq 2\alpha + 1 \tag{3.9}$$

$$E + A \geq n + M + 2\alpha \tag{3.10}$$

alors toute exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}_{HO}^A)$ satisfait la clause d'Accord du Consensus.

Démonstration: Soit ρ une exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}_{HO}^A)$ telle qu'une décision soit prise. Soit ϕ_0 la première phase de ρ au cours de laquelle un processus p décide et soit v la valeur décidée par p au round $2\phi_0$.
Nous montrons tout d'abord, par récurrence sur $\phi > \phi_0$, que

$$\forall \phi > \phi_0, \left| \begin{array}{lll} \forall q \in \Pi & : & x_q^{(2\phi-1)} = x_q^{(2\phi)} = v \\ \forall v' \notin \{v; ?\} & : & Q^{2\phi}(v') = \emptyset \end{array} \right.$$

- *Cas de base* $\phi = \phi_0 + 1$ *:* Soit q un processus quelconque.

  Si T, M, E et A vérifient les inéquations (3.6), (3.7) et (3.10), alors, puisque p décide v au round $2(\phi - 1)$, le Lemme 3.1.5 assure que q affecte la valeur v à sa variable x_q au round $2(\phi - 1)$.

  Par conséquent, on a $Q^{2\phi-1}(v) = \Pi$ et, pour toute valeur $v' \neq v$, $Q^{2\phi-1}(v') = \emptyset$.
  Sous $\mathcal{P}_\alpha$ on a donc, pour toute valeur $v' \neq v$, $|R_q^{2\phi-1}(v')| \leq \alpha$.
  Si A vérifie (3.9), alors on a $|R_q^{2\phi-1}(v)| \geq \alpha + 1$. Le code implique alors que q affecte la valeur v à sa variable x_q au round $2\phi - 1$. Ainsi

  $$\forall q \in \Pi \qquad x_q^{(2\phi-1)} = v.$$

  De plus, si $T \geq \frac{n}{2} + \alpha$, le code implique que $vote_q^{(2\phi-1)} \in \{v; ?\}$. Par conséquent

  $$\forall v' \notin \{v; ?\} \qquad Q^{2\phi}(v') = \emptyset.$$

  Puisque, pour toute valeur $v' \notin \{v; ?\}$, $Q^{2\phi}(v') = \emptyset$, on a sous $\mathcal{P}_\alpha$, $|R_q^{2\phi}(v')| \leq \alpha$. Si $M > \alpha$, on en déduit que q ne peut pas affecter une autre valeur que v à sa variable x_q au round 2ϕ. Comme on a montré $x_q^{(2\phi-1)} = v$, on en conclut $x_q^{(2\phi)} = v$. Ainsi

  $$\forall q \in \Pi \qquad x_q^{(2\phi)} = v,$$

  ce qui achève de démontrer le cas de base.

- *Étape de récurrence* $\phi > \phi_0 + 1$: Supposons que, pour tout processus q, $x_q^{(2(\phi-1))} = v$. Le raisonnement est identique à celui qui est utilisé pour démontrer le cas de base.

Soit à présent q un processus distinct de p. Supposons que q décide une valeur v' au round 2ϕ. Par définition de ϕ_0, on a $\phi \geq \phi_0$.

$*$ Si $\phi = \phi_0$ alors le code implique que

$$|R_p^{2\phi}(v)| > E \qquad \text{et} \qquad |R_q^{2\phi}(v')| > E.$$

Le Lemme 3.1.1 assure alors que, sous $\mathcal{P}_\alpha$, on a

$$|Q^{2\phi}(v)| > E - \alpha \qquad \text{et} \qquad |Q^{2\phi}(v')| > E - \alpha.$$

Si T et E vérifient les inéquations (3.6) et (3.8), on a

$$|Q^{2\phi}(v)| > 0 \qquad \text{et} \qquad |Q^{2\phi}(v')| > 0.$$

Le Lemme 3.1.4 assure alors que $v' = v$.

* Supposons $\phi > \phi_0$. Le code implique que $|R_q^{2\phi}(v')| > E$. Le Lemme 3.1.1 assure alors que, sous $\mathcal{P}_\alpha$, on a $|Q^{2\phi}(v')| > E - \alpha$.
  Si T et E vérifient les inéquations (3.6) et (3.8), on a $|Q^{2\phi}(v')| > 0$. Or on a montré que, pour toute phase $\phi > \phi_0$, pour toute valeur $v' \notin \{v; ?\}$, $Q^{2\phi}(v') = \emptyset$. On en déduit donc que $v' = v$, ce qui achève de démontrer que ρ satisfait bien la clause d'Accord du Consensus.

□

De la même manière, nous établissons une conditions suffisante sur les paramètres pour que chaque exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}_{HO}^A)$ satisfasse la clause d'Intégrité de Consensus.

Proposition 3.1.7 (Intégrité) *Si les paramètres T, M, E et A vérifient les inéquations*

$$n > T \geq \frac{n}{2} + \alpha \tag{3.11}$$

$$n \geq M > \alpha \tag{3.12}$$

$$n > E \geq \frac{n}{2} \tag{3.13}$$

$$n \geq A \geq 2\alpha + 1 \tag{3.14}$$

alors, dans toute exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}_{HO}^A)$ telle que les valeurs initales sont toutes égales à une valeur v, la seule valeur de décision possible est v.

Démonstration: Soit ρ une exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}_{HO}^A)$ telle que toutes les valeurs initiales sont égales à une même valeur v. Nous montrons tout d'abord, par récurrence sur $\phi > 0$, que

$$\forall \phi > 0, \left| \begin{array}{lcl} \forall q \in \Pi & : & x_q^{(2\phi-1)} = x_q^{(2\phi)} = v \\ \forall v' \notin \{v; ?\} & : & Q^{2\phi}(v') = \emptyset \end{array} \right.$$

- *Cas de base* $\phi = 1$ *:* Puisque toutes les valeurs initiales sont égales à v dans ρ, tous les processus envoient v au round 1.
  Sous $\mathcal{P}_\alpha$, chaque processus reçoit donc au plus α messages ne contenant pas v à ce round.
  Si $A > 2\alpha$, on en déduit que chaque processus reçoit au moins $\alpha + 1$ messages contenent v.
  Le code implique alors que tout processus q affecte la valeur v à sa variable x_q au round 1.

  D'autre part, si $T \geq \frac{n}{2} + \alpha$, alors chaque processus reçoit au plus T messages ne contenant pas v. Le code implique donc qu'aucun processus ne peut voter pour une valeur $v' \neq ?$ différente de v à ce round.

  On vient de montrer que pour toute valeur $v' \notin \{v; ?\}$, on a $Q^2(v') = \emptyset$.
  Sous $\mathcal{P}_\alpha$, on en déduit que chaque processus reçoit au plus α messages ne contenant pas v au round 2.
  Si $M > \alpha$, le code implique que tout processus q qui met à jour sa variable x_q au round 2 lui affecte nécessairement la valeur v.
  Puisque toutes les variables x_q sont égales à v à la fin du round 1, on en conclut qu'elles sont toutes égales à v à la fin du round 2.

- *Étape de récurrence* $\phi > 1$ *:* Supposons que pour tout q, $x_q^{(2(\phi-1))} = v$.
  Le raisonnement est identique à celui du cas de base.

Supposons à présent qu'il existe un processus p, une phase $\phi > 0$ de ρ et une valeur v' tels que p décide v' au round 2ϕ. Le code implique que $|R_p^{2\phi}(v')| > E$. Sous $\mathcal{P}_\alpha$, le Lemme 3.1.1 assure alors que $|Q^{2\phi}(v')| > E - \alpha$. Si T et E vérifient (3.11) et (3.13), on en déduit que $Q^{2\phi}(v') \neq \emptyset$.

Or on a montré que pour toute phase $\phi > 0$ et pour toute valeur w distincte de v et ?, on a $Q^{2\phi}(w) = \emptyset$. On en conclut donc que $v' = v$. $\square$

Il nous reste à déterminer sous quelles conditions la clause de Terminaison est vérifiée. Nous introduisons le prédicat $\mathcal{P}^{live}_{\mathcal{U}_{T,M,E}}$ donné en Figure 3.2.

La première partie assure l'existence d'une phase telle que les processus reçoivent exactement le même ensemble de messages au deuxième round ce qui, par conséquent, a pour effet d'uniformiser les variables x_q. La seconde assure qu'à la phase suivante, tous les processus votent pour une même va-

$\forall \phi > 0,\ \exists \phi_u \geq \phi,$	$\exists \Pi_u,\ \forall q \in \Pi \quad : \quad HO(q, 2\phi_u - 1) = SHO(q, 2\phi_u - 1) = \Pi_u$ $\forall q \in \Pi \ : \ (\lvert SHO(p, 2\phi_u + 1)\rvert > T) \wedge (\lvert SHO(p, 2(\phi_u + 1))\rvert > E)$

FIG. 3.2 – Prédicat $\mathcal{P}^{live}_{\mathcal{U}_{T,M,E}}$

leur v et finalement décident v. Ainsi, en combinant cette remarque et les propositions 3.1.6 et 3.1.7, nous énonçons le théorème suivant :

Théorème 3.1.8 *Si les paramètres T, M, E et A vérifient les inéquations*

$$n > T \geq \frac{n}{2} + \alpha \tag{3.15}$$

$$n > E \geq \frac{n}{2} \tag{3.16}$$

$$n \geq A \geq 2\alpha + 1 \tag{3.17}$$

$$n \geq M > \alpha \tag{3.18}$$

$$E + A \geq n + M + 2\alpha \tag{3.19}$$

alors la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^A_{HO} \wedge \mathcal{P}^{live}_{\mathcal{U}_{T,M,E}})$ résout le Consensus.

Démonstration: Soit ρ une exécution de la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^A_{HO} \wedge \mathcal{P}^{live}_{\mathcal{U}_{T,M,E}})$.

Si les paramètres vérifient les inéquations de l'énoncé, alors les propositions 3.1.6 et 3.1.7 assurent que ρ satisfait les clauses d'Accord et d'Intégrité du Consensus.

Nous montrons à présent que, sous ces mêmes hypothèses, ρ satisfait aussi la clause de Terminaison. On note, tout d'abord, que l'ensemble des collections $\big(HO(p,r); SHO(p,r)\big)_{p\in\Pi, r>0}$ qui satisfont $\mathcal{P}_\alpha \wedge \mathcal{P}^A_{HO} \wedge \mathcal{P}^{live}_{\mathcal{U}_{T,M,E}}$ est non-vide.

Soit ϕ_u une phase de ρ pour laquelle il existe un ensemble Π_u tel que

$$\forall q \in \Pi \quad : \quad HO(q, 2\phi_u - 1) = SHO(q, 2\phi_u - 1) = \Pi_u.$$

Sous $\mathcal{P}^A_{HO}$, on en déduit que $|\Pi_u| \geq A$. Si A vérifie (3.17), alors $\Pi_u \neq \emptyset$. Par conséquent, le code implique qu'il existe une valeur $v \in V$ telle que

$$\forall q \in \Pi \quad : \quad x_q^{(2\phi_u - 1)} = v.$$

Si T vérifie (3.15), alors le code, combiné au Lemme 3.1.4, assure que

$$\forall q \in \Pi \quad : \quad vote_q^{(2\phi_u-1)} \in \{v; ?\}.$$

Si les paramètres vérifient les inéquations de l'énoncé, le même argument que celui qui est utilisé dans les propositions 3.1.6 et 3.1.7 montre alors que

$$\forall q \in \Pi \quad : \quad x_q^{(2\phi_u)} = v.$$

On a donc nécessairement $Q^{2(\phi_u+1)}(v) = \Pi$. Puisque, pour tout q,

$$|SHO(q, 2\phi_u + 1)| > T,$$

on en déduit que si T vérifie (3.15), alors le code, combiné au Lemme 3.1.4, implique $vote_q^{(2\phi_u+1)} = v$. Ainsi, $Q^{2(\phi_u+1)}(v) = \Pi$.

Comme, pour tout processus q, $|SHO(q, 2(\phi_u + 1))| > E$, on en déduit que si E vérifie (3.16), alors le code, combiné au Lemme 3.1.3, assure que q décide v au round $2(\phi_u + 1)$. □

Quelles valeurs pour les paramètres ?

Nous venons d'exhiber une condition suffisante sur les paramètres T, M, E et A pour que la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}_{HO}^A \wedge \mathcal{P}_{\mathcal{U}_{T,M,E}}^{live})$ résolve le Consensus. Il faut à présent examiner dans quelle mesure, pour un entier donné α, $0 \leq \alpha \leq n$, il est possible d'affecter des valeurs à T, M, E et A qui satisfassent les inéquations (3.15), (3.16), (3.17), (3.18), (3.19).

De manière évidente (3.18), (3.16), et (3.19) impliquent $\alpha < \frac{n}{3}$, tandis que les deux autres impliquent $\alpha < \frac{n}{2}$.

Réciproquement, supposons $\alpha = \frac{n}{3} - \epsilon$, $0 < \epsilon \leq \frac{n}{3}$. Trivialement $E = A = n - \frac{\epsilon}{3}$ et $M = \alpha + \epsilon$ satisfont (3.18), (3.16), (3.17) et (3.19). De plus $T = \frac{n}{2} + \alpha$ satisfait (3.15). Il s'ensuit que si $\alpha < \frac{n}{3}$, il existe une instanciation des paramètres T, M, E et A telle que la machine $(\mathcal{U}_{T,M,E}, \mathcal{P}_\alpha \wedge \mathcal{P}_{HO}^A \wedge \mathcal{P}_{\mathcal{U}_{T,M,E}}^{live})$ résolve le Consensus.

Il est important de remarquer que, dans le cas où $\alpha = \frac{n}{3} - \epsilon$, $E = A = n - \frac{\epsilon}{3}$, $M = \alpha + \epsilon$ et $T = \frac{n}{2} + \alpha$, le prédicat $\mathcal{P}_{HO}^A$ impose que tous les ensembles d'écoute soient égaux à Π à chaque round. En d'autres termes, on ne peut pas assurer la sûreté d'une exécution sans requérir une vivacité extrêmement

$$\mathcal{P}^{vf}_{nosplit} :: \forall p, q \in \Pi,\ \forall r > 0,\ |SHO(p,r) \cap SHO(q,r)| \geq \alpha + 1$$

FIG. 3.3 – Predicate $\mathcal{P}^{vf}_{nosplit}$

forte des communications. Plus précisément, on ne peut pas tolérer un tiers de corruptions, au sens de Santoro et Widmayer [42], par processus et par round et, dans le même temps, tolérer des omissions. En d'autres termes, si on veut pouvoir tolérer des omissions, alors on doit réduire le nombre de corruptions à chaque round.

3.1.3 Algorithme $\mathcal{UV}_{vf}$

Nous présentons à présent l'algorithme $\mathcal{UV}_{vf}$, donné comme Algorithme 3.3, version non-paramétrée de $\mathcal{U}_{T,M,E}$. Tout vote ou décision requiert qu'une unique valeur soit reçue. Nous donnons, de plus, la valeur fixe $\alpha + 1$ au paramètre M. Enfin, nous supposons que le code de $\mathcal{UV}_{vf}$ est exécuté sous le prédicat $\mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit}$, où $\mathcal{P}^{vf}_{nosplit}$ est le prédicat donné en Figure 3.3.

Preuve de correction

Nous reprenons ici les notations utilisées dans la preuve de correction de $\mathcal{U}_{T,M,E}$.

Notre premier lemme montre que deux valeurs distinctes ne peuvent pas être votées au cours d'un même round.

Lemme 3.1.9 *Dans toute exécution de la machine* $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$*, si un processus vote pour une valeur* $v \neq ?$ *à un round* $r > 0$*, alors tous les processus votent pour* v *ou* $?$ *à ce round.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$ telle qu'il existe un processus p qui vote pour une valeur $v \neq ?$ à un round $r > 0$.

Supposons qu'il existe un processus $q \neq p$ et une valeur $v' \neq ?$ tels que q vote pour v' au round r. Le code (ligne 9) implique

$$SHO(p,r) \subseteq Q^r(v) \qquad \text{et} \qquad SHO(q,r) \subseteq Q^r(v').$$

Sous $\mathcal{P}^{vf}_{nosplit}$, on a

$$SHO(p,r) \cap SHO(p,r) \neq \emptyset.$$

Algorithme 3.3 Algorithme $\mathcal{UV}_{vf}$

1: **Initialization :**
2: $x_p \in V$; initialement $x_p = v_p$
3: $vote_p \in V \cup \{?\}$; initialement $vote_p = ?$

4: **Round $r = 2\phi - 1$:**
5: S_p^r :
6: envoyer $\langle x_p \rangle$ à tous

7: T_p^r :

8: $x_p :=$ la plus petite des valeurs les plus fréquemment reçues
9: **if** toutes les valeurs reçues sont égales à v **then**
10: $vote_p := v$

11: **Round $r = 2\phi$:**
12: S_p^r :
13: envoyer $\langle vote_p \rangle$ à tous

14: T_p^r :
15: **if** au moins $\alpha + 1$ messages $\langle v \rangle$ sont reçus avec $v \neq ?$ **then**
16: $x_p := v$

17: **if** toutes les valeurs reçues sont égales à v avec $v \neq ?$ **then**
18: DECIDE(v)

19: $vote_p := ?$

On en déduit donc

$$Q^r(v) \cap Q^r(v) \neq \emptyset.$$

Puisqu'à chaque round, chaque processus envoie le même message à tous les autres, on en conclut finalement $v' = v$. Ainsi

$$\forall q \in \Pi \ : \quad vote_q^{(r)} \in \{v; ?\}.$$

□

Le lemme suivant démontre qu'à chaque phase $\phi > 0$, la règle de mise à jour de la ligne 15 est "déterministe".

Lemme 3.1.10 *Sous $\mathcal{P}_\alpha \wedge \mathcal{P}_{nosplit}^{vf}$, la garde de la ligne 15 est vraie pour au plus une valeur $v \neq ?$ par processus et par phase.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{nosplit}^{vf})$. Supposons, par contradiction, qu'il existe une phase $\phi > 0$ de ρ et un pro-

cessus p pour lesquels la garde de la ligne 15 est vraie pour deux valeurs distinctes v, v', toutes deux différentes de ?, au round 2ϕ.

Le code assure alors que $|R_p^r(v)| \geq \alpha + 1$ et $|R_p^r(v')| \geq \alpha + 1$. Sous $\mathcal{P}_\alpha$, cela implique que $|Q^r(v)| \geq 1$ et $|Q^r(v')| \geq 1$ ce qui, sous $\mathcal{P}_{nosplit}^{vf}$, contredit le Lemme 3.1.9. □

Nous étendons à présent le résultat précédent en montrant qu'à chaque phase ϕ, si un processus p affecte une valeur v à sa variable x_p au round 2ϕ, alors tout processus q qui met à jour sa variable x_q à ce round lui affecte nécessairement la valeur v.

Lemme 3.1.11 *Sous $\mathcal{P}_\alpha \wedge \mathcal{P}_{nosplit}^{vf}$, la garde de la ligne 15 est vraie pour au plus une valeur $v \neq ?$ par phase.*

Démonstration: La preuve est identique à celle du Lemme 3.1.10. □

Le lemme suivant démontre qu'à chaque phase, au plus une valeur peut être décidée.

Lemme 3.1.12 *Dans toute exécution de la machine $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{nosplit}^{vf})$, il existe au plus une valeur de décision possible par phase.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{nosplit}^{vf})$ telle qu'il existe deux processus p et q qui décident respectivement v et v' à un round r. Le code (ligne 17) implique alors

$$SHO(p,r) \subseteq Q^r(v) \qquad \text{et} \qquad SHO(q,r) \subseteq Q^r(v').$$

Sous $\mathcal{P}_{nosplit}^{vf}$, on a

$$SHO(p,r) \cap SHO(p,r) \neq \emptyset,$$

et par conséquent

$$Q^r(v) \cap Q^r(v) \neq \emptyset.$$

Le Lemme 3.1.9 assure finalement $v = v'$. □

Le lemme suivant assure que, dès lors qu'une valeur v a été décidée, d'une part les variables x_p sont nécessairement toutes égales à v et le restent, et que, d'autre part, aucun processus ne peut plus voter pour une valeur $v' \neq ?$ différente de v.

Lemme 3.1.13 *Dans toute exécution de la machine* $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$, *s'il existe un processus* p *qui décide une valeur* v *à un round* r, *alors :*

$$\forall q \in \Pi,\ \forall r' \geq r \quad : \quad x_q^{(r')} = v \quad \wedge \quad vote_q^{(r')} \in \{v; ?\}.$$

Démonstration: Soit ρ une exécution de la machine $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$ dans laquelle une décision est prise. Soit $r > 0$ le premier round de ρ au cours duquel un processus p décide une valeur v. On raisonne par récurrence sur $r' \geq r$.

- *Cas de base :* $r' = r$. Puisque p décide au round r, le code (ligne 17) assure $SHO(p, r) \subseteq Q^r(v)$. Par conséquent, pour tout processus q, on a $SHO(q, r) \cap SHO(p, r) \subseteq Q^r(v)$, et donc

  $$SHO(q, r) \cap SHO(p, r) \cap Q^r(v) = SHO(q, r) \cap SHO(p, r).$$

  Puisque $SHO(q, r) \cap SHO(p, r) \cap Q^r(v) \subseteq R^r_q(v)$, on en déduit que

  $$|R^r_q(v)| \geq |SHO(q, r) \cap SHO(p, r)|.$$

  Sous $\mathcal{P}^{vf}_{nosplit}$, on obtient $|R^r_q(v)| \geq \alpha + 1$. Le code (ligne 15) et le Lemme 3.1.10 impliquent finalement que q affecte la valeur v à sa variable x_q au round r.
  Enfin, le code implique que toutes les variables $vote_q$ sont égales à ? à la fin de chaque phase.

- *Étape de récurrence :* $r' > r$. Soit $r' > r$ et supposons que pour tout processus q on ait $x_q^{(r'-1)} = v$ et $vote_q^{(r'-1)} \in \{v; ?\}$. Soit p' un processus quelconque.

  Si r' est le premier round d'une phase ϕ', l'hypothèse de récurrence implique $Q^{r'}(v) = \Pi$ et donc, pour toute valeur $v' \neq v$, $Q^{r'}(v') = \emptyset$. Par conséquent, sous $\mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit}$, on a $|R^{r'}_{p'}(v)| \geq \alpha + 1$ et $|R^{r'}_{p'}(v')| \leq \alpha$ pour toute valeur $v' \neq v$. Le code (ligne 8) implique alors que p' affecte la valeur v à sa variable $x_{p'}$ au round r'. Enfin, puisque $R^{r'}_{p'}(v) \neq \emptyset$, on a nécessairement $vote_{p'}^{(r')} \in \{v; ?\}$.

  Si r' est le second round d'une phase ϕ', l'hypothèse de récurrence implique $Q^{r'}(v') = \emptyset$, pour toute valeur $v' \notin \{v; ?\}$. Il s'ensuit que,

sous $\mathcal{P}_\alpha$, pour tout processus q et pour toute valeur $v' \notin \{v; ?\}$, on a $|R^{r'}(v')| \leq \alpha$. Le code (ligne 15) assure alors qu'aucun processus q ne peut affecter une valeur différente de v à sa variable x_q au round r'. Enfin, le code implique que toutes les variables $vote_q$ sont égales à ? à la fin de chaque phase.

□

La combinaison des précédents lemmes nous conduit à énoncer la proposition suivante :

Proposition 3.1.14 (Accord) *Toute exécution de la machine* $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$ *satisfait la clause d'Accord du Consensus.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$ dans laquelle une décision est prise. Soit $r > 0$ le premier round de ρ au cours duquel un processus p décide une valeur v.

Supposons qu'il existe un processus q distinct de p qui décide une valeur v' à un round r'. Par définition de r on a $r' \geq r$. Il y a deux cas à considérer :

(1) Si $r' = r$, alors le Lemme 3.1.12 assure $v' = v$.

(2) Si $r' > r$, alors, puisque q décide v' au round r', le code (ligne 17) implique que toutes les valeurs reçues par q au round r' sont égales à v'.
Sous $\mathcal{P}^{vf}_{nosplit}$, on en déduit que $Q^{r'}(v') \neq \emptyset$. Sous $\mathcal{P}_\alpha$, le Lemme 3.1.13 implique finalement $v' = v$. □

Nous montrons à présent que la clause d'Intégrité du Consensus est, elle aussi, satisfaite par toute exécution de $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$.

Proposition 3.1.15 (Intégrité) *Dans toute exécution de la machine* $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$ *telle que toutes les valeurs initiales sont égales à une valeur* v*, la seule valeur de décision possible est* v*.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$ telle que toutes les valeurs initiales sont égales à une valeur v. L'argument utilisé dans la preuve du Lemme 3.1.13 démontre que l'on a

$$\forall r > 0,\ \forall q \in \Pi \quad : \quad x_q^{(r)} = v \quad \wedge \quad vote_q^{(r)} \in \{v; ?\}.$$

$$\mathcal{P}^{live}_{\mathcal{UV}_{vf}} :: \forall \phi > 0,\ \exists \phi' \geq \phi,\ \exists \Pi' \subseteq \Pi \left| \begin{array}{l} \forall q \in \Pi,\ \ HO(q, 2\phi' - 1) = SHO(q, 2\phi' - 1) = \Pi' \\ \forall q \in \Pi,\ \exists \phi_q > \phi'\ \ AS(2\phi_q - 1) = \emptyset \wedge AHO(q, 2\phi_q) = \emptyset \end{array} \right.$$

FIG. 3.4 – Prédicat $\mathcal{P}^{live}_{\mathcal{UV}_{vf}}$

Supposons qu'il existe un processus p qui décide une valeur v' à un round r. Le code (ligne 17) implique alors que toutes les valeurs reçues par p au round r sont égales à v'.
Sous $\mathcal{P}^{vf}_{nosplit}$, on en déduit que $Q^r(v') \neq \emptyset$. Puisqu'on a montré qu'à chaque round $r > 0$, pour tout processus q, on a $vote_q^{(r-1)} \in \{v; ?\}$, on en conclut que nécessairement $v' = v$. □

Les propositions 3.1.14 et 3.1.15 assurent que toute exécution de la machine $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit})$ est sûre. Pour ce qui est de la vivacité, nous introduisons le prédicat de communication $\mathcal{P}^{live}_{\mathcal{UV}_{vf}}$, donné en Figure 3.4.

La première partie force l'uniformisation des variables x_q, tandis que la seconde assure l'existence de phases dans lesquelles le nombre de processus qui votent pour une valeur différente de ? et qui sont correctement entendus au round suivant est suffisant pour forcer la décision d'un processus.

Théorème 3.1.16 *La machine* $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit} \wedge \mathcal{P}^{live}_{\mathcal{UV}_{vf}})$ *résout le Consensus.*

Démonstration: On note, tout d'abord, que, pour tout entier $\alpha < n$, l'ensemble des collections $\big(HO(p,r); (SHO(p,r)\big)_{p \in \Pi, r > 0}$ qui satisfont $\mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit} \wedge \mathcal{P}^{live}_{\mathcal{UV}_{vf}}$ est non-vide. Soit donc ρ une exécution de la machine $(\mathcal{UV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{vf}_{nosplit} \wedge \mathcal{P}^{live}_{\mathcal{UV}_{vf}})$.

Le fait que ρ satisfait bien les clauses d'Accord et d'Intégrité du Consensus est établi par les propositions 3.1.14 et 3.1.15, respectivement.

Il reste à démontrer que ρ satisfait aussi la clause de Terminaison. Soit ϕ' la première phase de ρ telle que

$$\exists \Pi' \subseteq \Pi,\ \forall q \in \Pi : \quad HO(q, 2\phi' - 1) = SHO(q, 2\phi' - 1) = \Pi'.$$

Puisque $\mathcal{P}^{vf}_{nosplit}$ est satisfait dans ρ, on en déduit $\Pi' \neq \emptyset$. Tous les processus reçoivent donc le même ensemble non-vide de messages au round $2\phi' - 1$. Le

code implique alors qu'à ce round, d'une part (ligne 8), tous les processus q affectent une même valeur v à leur variable x_q et, d'autre part (ligne 9), qu'aucun ne peut voter pour une valeur $v' \neq ?$ différente de v. On a ainsi

$$\forall q \in \Pi \quad : \quad x_q^{(2\phi'-1)} = v \quad \wedge \quad vote_q^{(2\phi'-1)} \in \{v; ?\}.$$

Sous $\mathcal{P}_\alpha \wedge \mathcal{P}_{nosplit}^{vf}$, l'argument utilisé dans la preuve du Lemme 3.1.13 démontre que cela est valable pour tout round $r' \geq 2\phi' - 1$.

Soit à présent p un processus quelconque et soit ϕ_p la première phase telle que $\phi_p > \phi'$ et

$$AS(2\phi_p - 1) = \emptyset \quad \wedge \quad AHO(p, 2\phi_p) = \emptyset.$$

Supposons que p n'ait pas encore décidé au début de la phase ϕ_p. Puisque toutes les variables x_q sont égales à v à la fin de la phase $(\phi_p - 1)$ et que $AS(2\phi_p - 1) = \emptyset$, on a nécessairement, pour toute valeur $v' \neq v$ et pour tout processus q, $R_q^{2\phi_p - 1}(v') = \emptyset$. De plus, $\mathcal{P}_{nosplit}^{vf}$ implique que, pour tout processus q, on a $R_q^{2\phi_p - 1}(v) \neq \emptyset$. Par conséquent, le code (ligne 9) assure que tous les processus votent pour v au round $(2\phi_p - 1)$.

Puisque $AHO(p, 2\phi_p) = \emptyset$ et, sous $\mathcal{P}_{nosplit}^{vf}$, $HO(p, 2\phi_p) \neq \emptyset$, on en déduit alors $R_p^{2\phi_p}(v) = HO(p, 2\phi_p)$. Le code (ligne 17) implique finalement que p décide v au round $2\phi_p$. □

$\mathcal{UV}_{vf}$ est plus sûr que $\mathcal{U}_{T,M,E}$

Comme c'était le cas pour $\mathcal{U}_{T,M,E}$, le prédicat garantissant la sûreté de $\mathcal{UV}_{vf}$, c'est-à-dire $\mathcal{P}_\alpha \wedge \mathcal{P}_{nosplit}^{vf}$, impose une certaine vivacité des communications en présence d'erreurs de transmissions par valeurs. En effet, $\mathcal{P}_{nosplit}^{vf}$ force les ensembles d'écoute à être de cardinal strictement supérieur au nombre de corruptions que l'on veut tolérer. On retrouve ainsi le même type de dépendance entre le nombre d'omissions et le nombre de corruptions pouvant intervenir dans un même round. Cette dépendance est cependant moins forte puisque $\mathcal{UV}_{vf}$ tolère $\alpha < \frac{n}{2}$ erreurs de transmissions par valeurs dans les rounds sans omissions, contre $\alpha < \frac{n}{3}$ pour $\mathcal{U}_{T,M,E}$.

3.2 Communications plus sûres

Avec *UniformVoting*, Charron-Bost et Schiper ont donné une solution pour le Consensus qui nécessite une propriété invariante de vivacité des communications. On trouve dans [17] un autre algorithme, appelé *OneThirdRule*, pour lequel ce n'est pas le cas. Plus précisément, toute exécution de cet algorithme satisfait les clauses d'Accord et d'Intégrité sans qu'il soit nécessaire de faire la moindre hypothèse sur la vivacité des communications.

3.2.1 Algorithme *OneThirdRule*

Le schéma algorithmique est le même que celui qui est utilisé dans le premier round de [7], dans [39] et dans l'algorithme *Fast Paxos* [29].

Algorithme 3.4 Algorithme *OneThirdRule*

```
1:  Initialisation :
2:    x_p := v_p

3:  Round r :
4:    S_p^r :
5:      envoyer ⟨x_p⟩ à tous

6:    T_p^r :
7:      if |HO(p,r)| > 2n/3 then
8:        x_p := la plus petite des valeurs les plus fréquemment reçues
9:        if plus de 2n/3 valeurs reçues sont égales à v then
10:         DECIDE(v)
```

Cet algorithme est toujours sûr, même en présence de nombreuses erreurs de transmission. La Terminaison est assurée par l'existence d'un round uniforme de noyau suffisamment grand (de cardinal supérieur à $2n/3$) et postérieurement, pour chaque processus p, d'un round au cours duquel p reçoît plus de $2n/3$ messages, ce qui est capturé par le prédicat suivant :

$$\forall r > 0,\ \exists r_0 \geq r,\ \exists \Pi_0 \text{ t.q. } |\Pi_0| > \frac{2n}{3},\ \forall p \in \Pi \quad : \quad HO(p, r_0) = \Pi_0.$$

D'autre part, Charron-Bost et Schiper font remarquer que, dans certaines exécutions, une décision est possible en un round : si toutes les valeurs initiales sont égales et si le premier round satisfait le prédicat donné ci-dessus, alors tous les processus décident à la fin du premier round. *OneThirdRule* peut donc se révéler relativement efficace.

3.2.2 Algorithme $\mathcal{A}_{T,E}$

Nous reprenons ici la démarche, adoptée pour $\mathcal{U}_{T,M,E}$, consistant à paramétrer les seuils de réception. Nous obtenons l'algorithme $\mathcal{A}_{T,E}$, donné comme Algorithme 3.5, que nous supposons être exécuté sous le prédicat $\mathcal{P}_\alpha$.

Algorithme 3.5 Algorithme $\mathcal{A}_{T,E}$

```
1:  Initialisation :
2:    x_p := v_p

3:  Round r :
4:    S_p^r :
5:      envoyer ⟨x_p⟩ à tous

6:    T_p^r :
7:      if |HO(p,r)| > T then
8:        x_p := la plus petite des valeurs les plus fréquemment reçues
9:        if plus de E valeurs reçues sont égales à v then
10:           DECIDE(v)
```

De manière informelle, dans $\mathcal{A}_{T,E}$, chaque processus p dispose d'une variable locale x_p, initialement égale à sa valeur initiale. A chaque round, chaque processus p diffuse x_p, et la met à jour s'il reçoit plus de T messages. De plus, si un processus p reçoit plus de E messages contenant une même valeur v, alors il décide v.

Preuve de correction

Avant de débuter, nous rappelons l'énoncé du Lemme 3.1.1.

Lemme 3.2.1 *Pour tout processus p, pour toute valeur v, et pour tout round $r > 0$, on a :*

$$|R_p^r(v)| \leq |Q^r(v)| + |AHO(p,r)|$$

Le lemme suivant montre que si on choisit $E \geq \frac{n}{2}$, on rend la règle de décision "déterministe".

Lemme 3.2.2 *Si $E \geq \frac{n}{2}$, alors la garde à la ligne 9 est vraie pour au plus une valeur par processus et par round.*

Démonstration: Supposons, par contradiction, qu'il existe un processus p et un round r tels que la garde à la ligne 9 soit vraie pour deux valeurs distinctes v et v'. Le code implique que $|R_p^r(v)| > E$ et $|R_p^r(v')| > E$.
Puisque v et v' sont distinctes, les ensembles $R_p^r(v)$ et $R_p^r(v')$ sont disjoints, et donc

$$|R_p^r(v) \cup R_p^r(v')| = |R_p^r(v)| + |R_p^r(v')|.$$

Si $E \geq \frac{n}{2}$, on en déduit que $|R_p^r(v) \cup R_p^r(v')| > n$, une contradiction. □

Nous montrons à présent qu'une condition plus forte sur E assure que deux processus ne peuvent pas décider différemment au cours d'un même round.

Lemme 3.2.3 *Si $E \geq \frac{n}{2} + \alpha$ alors, dans toute exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$, il y a au plus une valeur de décision possible par round.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$. Supposons, par contradiction, qu'il existe deux processus distincts p et q qui décident deux valeurs distinctes v et v' au cours d'un round $r > 0$. Le code implique alors $|R_p^r(v)| > E$ et $|R_q^r(v')| > E$. Sous $\mathcal{P}_\alpha$, le Lemme 3.2.1 assure $|Q^r(v)| > E - \alpha$ et $|Q^r(v')| > E - \alpha$.

Puisque tout processus envoie le même message à tous les autres à chaque round, et qu'on a supposé v et v' distinctes, les ensembles $Q^r(v)$ et $Q^r(v')$ sont disjoints. Par conséquent

$$|Q^r(v) \cup Q^r(v')| = |Q^r(v)| + |Q^r(v')|.$$

Si $E \geq \frac{n}{2} + \alpha$, on en déduit $|Q^r(v) \cup Q^r(v')| > n$, une contradiction. □

Le lemme suivant assure que si T est "assez grand", alors la décision d'une valeur v à un round empêche les autres processus d'affecter à leur variable x_p, à ce round, une autre valeur que la valeur de décision.

Lemme 3.2.4 *Si $T \geq 2(n + 2\alpha - E)$, alors, dans toute exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$, s'il existe un processus p qui décide une valeur v à un round r, tout processus q qui met à jour sa variable x_q à ce round lui affecte nécessairement la valeur v.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$. Supposons qu'il existe un processus p qui décide une valeur v à un round $r > 0$. Supposons qu'il existe un processus q tel que $|HO(q,r)| > T$, et qui donc met à jour sa variable x_q au round r. On définit les ensembles $Q^r(\overline{v})$ et $R_q^r(\overline{v})$ de la manière suivante :

$$Q^r(\overline{v}) = \bigcup_{v' \neq v} Q^r(v') \qquad \text{et} \qquad R_q^r(\overline{v}) = \bigcup_{v' \neq v} R_q^r(v').$$

Puisque, à chaque round, tous les processus envoient le même message à tous les autres et reçoivent au plus un message de chacun des autres, on a

$$Q^r(\overline{v}) = \Pi \setminus Q^r(v) \qquad \text{et} \qquad R_q^r(\overline{v}) = HO(q,r) \setminus R_q^r(v).$$

Puisque $R_q^r(v) \subseteq HO(q,r)$, on en déduit

$$|Q^r(\overline{v})| = n - |Q^r(v)| \qquad \text{et} \qquad |R_q^r(\overline{v})| < T - |R_q^r(v)|.$$

Puisque p décide v au round r, le code (ligne 9) implique $|R_p^r(v)| > E$. Sous $\mathcal{P}_\alpha$, le Lemme 3.2.1 assure $|Q^r(v)| > E - \alpha$, ce qui nous permet de déduire $|Q^r(\overline{v})| < n - (E - \alpha)$. Le Lemme 3.2.1 implique alors, sous $\mathcal{P}_\alpha$, $|R_q^r(\overline{v})| < n + 2\alpha - E$.

Il s'ensuit que, si $T \geq 2(n + 2\alpha - E)$, on obtient $|R_q^r(v)| > |R_q^r(\overline{v})|$. La valeur v est par conséquent la plus fréquemment reçue par q au round r. Le code implique finalement que q affecte la valeur v à sa variable x_q à ce round. □

Nous étendons l'énoncé du lemme précédent de telle sorte qu'il soit valable à chaque round après qu'une valeur a été décidée.

Lemme 3.2.5 *Si $T \geq 2(n + 2\alpha - E)$, alors, dans toute exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$, s'il existe un processus p qui décide une valeur v à un round r, tout processus q qui met à jour sa variable x_q à un round $r' \geq r$ lui affecte nécessairement la valeur v.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$. Supposons qu'il existe un processus p qui décide une valeur v à un round $r > 0$.
On démontre tout d'abord, par récurrence sur r', que pour tout round $r' \geq r$, $|Q^{r'}(v)| > E - \alpha$ ou, de manière équivalente,

$$\forall r' \geq r \quad : \quad |\{p' \in \Pi \ : \ x_{p'}^{(r'-1)} = v\}| > E - \alpha.$$

- *Cas de base : $r' = r$.* Puisque p décide v au round r, le code implique $|R_p^r(v)| > E$. Par le Lemme 3.2.1, sous $\mathcal{P}_\alpha$, on obtient $|Q^r(v)| > E - \alpha$.

- *Étape de récurrence : $r' > r$.* Supposons $|Q^{r'-1}(v)| > E - \alpha$. Le même argument que celui utilisé dans le Lemme 3.2.4 démontre que, si $T \geq 2(n + 2\alpha - E)$, alors $|\{p' \in \Pi \ : \ x_{p'}^{(r')} = v\}| > E - \alpha$.

Supposons à présent qu'il existe un processus q et un round $r' \geq r$ tels que $|HO(q, r')| > T$.

Puisque $|Q^{r'}(v)| > E - \alpha$, l'argument du Lemme 3.2.4 s'applique. Le code force donc q à affecter la valeur v à sa variable x_q au round r'. □

La combinaison des lemmes précédents nous conduit à énoncer une condition suffisante sur le paramètres T et E pour que toute exécution de $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$ satisfasse la clause d'Accord du Consensus.

Proposition 3.2.6 (Accord) *Si $E \geq \frac{n}{2} + \alpha$ et $T \geq 2(n + 2\alpha - E)$, alors il y a au plus une valeur de décision dans toute exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$. Supposons qu'une décision soit prise. Soit $r > 0$ le premier round de ρ au cours duquel un processus p décide une valeur. On note v la décision de p. Supposons qu'il existe un processus q distinct de p qui décide une valeur v' à un round r'. Par définition de r, on a $r' \geq r$.

(1) Si $r' = r$, alors, si $E \geq \frac{n}{2} + \alpha$, le Lemme 3.2.2 assure $v' = v$, sous $\mathcal{P}_\alpha$.

(2) Si $r' > r$, alors, sous $\mathcal{P}_\alpha$, le Lemme 3.2.1 implique $|Q^r(v)| > E - \alpha$ et $|Q^{r'}(v')| > E - \alpha$.

Supposons, par contradiction, que v et v' sont distinctes.
Si $T \geq 2(n + 2\alpha - E)$, le Lemme 3.2.5 assure alors que $Q^r(v)$ et $Q^{r'}(v')$ sont des ensembles disjoints, et donc

$$|Q^r(v) \cup Q^{r'}(v')| = |Q^r(v)| + |Q^{r'}(v')|.$$

Par conséquent, on obtient

$$|Q^r(v) \cup Q^{r'}(v')| > 2(E - \alpha)$$

ce qui, si $E \geq \frac{n}{2} + \alpha$, implique $|Q^r(v) \cup Q^{r'}(v')| > n$, une contradiction. $\square$

De manière similaire, nous exhibons une condition suffisante sur T et E pour que toute exécution de $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$ satisfasse la clause d'Intégrité du Consensus.

Proposition 3.2.7 (Intégrité) *Si $E \geq \alpha$ et $T \geq 2\alpha$ alors, dans toute exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$ telle que toute les valeurs initiales sont égales à une même valeur v, la seule valeur de décision possible est v.*

Démonstration: Soit ρ une exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$ dans laquelle toutes les valeurs initiales sont égales à une valeur v. On montre tout d'abord, par récurrence sur r, que pour tout round $r > 0$ on a $Q^r(v) = \Pi$ ou, de manière équivalente,

$$\forall r > 0 \quad : \quad \{q \in \Pi \ : \ x_q^{(r-1)} = v\} = \Pi.$$

- *Cas de base : $r = 1$.* Puisque toutes les valeurs initiales sont égales à v, tous les processus envoient un message contenant v au round 1.

- *Étape de récurrence : $r > 1$.* Supposons que $Q^{r-1}(v) = \Pi$. Soit p un processus qui met à jour sa variable x_p au round $r-1$, c'est-à-dire tel que $|HO(p, r-1) > T$. Puisque tous les processus envoient v au round $r-1$ on en déduit que, sous $\mathcal{P}_\alpha$, le processus p reçoit au plus α messages ne contenant pas v. Par conséquent, si $T \geq 2\alpha$, il reçoit au moins $\alpha + 1$ fois v à ce round. On en déduit que v est la valeur la plus fréquemment reçue par p au round $r-1$. Le code implique que p affecte nécessairement la valeur v à sa variable x_p à ce round.
  Ainsi, au round $r-1$, les variables x_p restent inchangées et donc $Q^r(v) = \Pi$.

Supposons à présent qu'il existe un processus p qui décide une valeur v' à un round $r > 0$.

Le code implique $|R_p^r(v')| > E$. Sous $\mathcal{P}_\alpha$, le Lemme 3.2.1 assure alors $|Q^r(v')| > E - \alpha$.

Si $E \geq \alpha$, on en déduit $Q^r(v') \neq \emptyset$. Comme $Q^r(v) = \Pi$ et comme chaque processus envoie le même message à tous à chaque round, on en conclut finalement $v' = v$. $\square$

$$\forall r > 0,\ \exists r_0 \geq r,\ \exists \Pi^1_{r_0}, \Pi^2_{r_0} \subseteq \Pi : \left| \begin{array}{l} |\Pi^1_{r_0}| > E - \alpha \quad \wedge \quad |\Pi^2_{r_0}| > T \\ \forall p \in \Pi^1_{r_0} :\ HO(p, r_0) = SHO(p, r_0) = \Pi^2_{r_0} \end{array} \right. \tag{3.20}$$

$$\wedge$$

$$\forall r > 0,\ \forall p \in \Pi,\ \exists r_p > r : \quad |HO(p, r_p)| > T \tag{3.21}$$

$$\wedge$$

$$\forall r > 0,\ \forall p \in \Pi,\ \exists r'_p > r : \quad |SHO(p, r'_p)| > E \tag{3.22}$$

FIG. 3.5 – Prédicat $\mathcal{P}^{live}_{\mathcal{A}_{T,E}}$

Nous venons de donner des conditions suffisantes sur les paramètres T et E pour garantir que toute exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$ soit sûre, c'est-à-dire qu'elle satisfasse les clauses d'Accord et d'Intégrité du Consensus. Pour la vivacité, nous introduisons le prédicat $\mathcal{P}^{live}_{\mathcal{A}_{T,E}}$, donné en Figure 3.5.

Les deux premières parties (3.2.2 et 3.21) forcent l'uniformisation des variables x_p, tandis que la dernière (3.22) assure, pour chaque processus, l'existence d'un round au cours duquel il reçoit assez de messages pour décider.

Proposition 3.2.8 (Terminaison) *Si $n > E \geq \frac{n}{2} + \alpha$ et $n > T \geq 2(n + 2\alpha - E)$, alors toute exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{\mathcal{A}_{T,E}})$ satisfait la clause de Terminaison du Consensus.*

Démonstration: On remarque tout d'abord que, si $n > T$ et $n > E$, alors l'ensemble des collections $\big(HO(p,r); SHO(p,r)\big)_{p \in \Pi, r > 0}$ qui satisfont $\mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{\mathcal{A}_{T,E}}$ est non-vide. Il existe donc des exécutions qui satisfont $\mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{\mathcal{A}_{T,E}}$.

Soit ρ une exécution de $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{\mathcal{A}_{T,E}})$, et soit r_0 un round de ρ tel que

$$\exists \Pi^1_{r_0}, \Pi^2_{r_0} \subseteq \Pi : \left| \begin{array}{l} |\Pi^1_{r_0}| > E - \alpha \quad \wedge \quad |\Pi^2_{r_0}| > T \\ \forall p \in \Pi^1_{r_0} :\ HO(p, r_0) = SHO(p, r_0) = \Pi^2_{r_0} \end{array} \right.$$

Tous les processus de $\Pi^1_{r_0}$ reçoivent exactement le même ensemble de messages, de cardinal strictement supérieur à T, au round r_0. Le code (ligne 8) assure qu'ils affectent tous une même valeur v à leur variable x_p à ce round.

Puisque $|\Pi^1_{r_0}| > E - \alpha$, on en déduit donc $|Q^{r_0+1}(v)| > E - \alpha$. Sous $\mathcal{P}_\alpha$, si $T \geq 2(n + 2\alpha - E)$, un argument similaire à celui qui est utilisé dans le Lemme 3.2.5 démontre que tout processus q qui met à jour sa variable x_q au cours d'un round $r > r_0$ lui affecte nécessairement la valeur v.

Or $\mathcal{P}^{live}_{\mathcal{A}_{T,E}}$ implique que, pour tout processus q, il existe un round $r_q > r_0$ tel que $|HO(p, r_q)| > T$, c'est-à-dire un round au cours duquel q met à jour sa variable x_q. Par conséquent, il existe un round r_1 tel que, pour tout processus q, on a $x_q^{(r_1)} = v$.

De plus, si $T \geq 2(n+2\alpha-E)$, un argument similaire à celui qui est utilisé dans le Lemme 3.2.5 démontre que les x_q sont uniformément égales à v par la suite.

Enfin $\mathcal{P}^{live}_{\mathcal{A}_{T,E}}$ implique que, pour tout processus q, il existe un round $r'_q > r_1$ tel que $|SHO(q, r_q)| > E$. Comme $Q^{r_q}(v) = \Pi$, on en déduit que $|R_q^{r_q}(v)| > E$.

Le code (ligne 9) assure alors que q décide v au round r_q. □

En combinant les propositions 3.2.6, 3.2.7 et 3.2.8, nous obtenons le théorème suivant :

Théorème 3.2.9 *Si $n > E \geq \frac{n}{2} + \alpha$ et $n > T \geq 2(n + 2\alpha - E)$, alors la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{\mathcal{A}_{T,E}})$ résout le Consensus.*

Démonstration: Soit ρ une exécution de $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{\mathcal{A}_{T,E}})$. Le fait que ρ satisfait bien les clauses d'Accord et de Terminaison est une conséquence directe des propositions 3.2.6 et 3.2.8.

Pour l'Intégrité, il suffit de vérifier que les conditions $E \geq \alpha$ et $T \geq 2\alpha$ sont assurées par $n > E \geq \frac{n}{2} + \alpha$ et $n > T \geq 2(n + 2\alpha - E)$ respectivement.

De manière évidente, pour tout $\alpha \geq 0$, on a :

$$E \geq \frac{n}{2} + \alpha \quad \Rightarrow \quad E \geq \alpha.$$

De plus

$$n > E \quad \wedge \quad T \geq 2(n + 2\alpha - E) \quad \Rightarrow \quad T \geq 2\alpha.$$

□

Quelles valeurs pour T et E ?

Comme c'était le cas pour $\mathcal{U}_{T,M,E}$, il est impératif de déterminer si, étant donné un entier α, $0 \leq \alpha \leq n$, il est possible d'affecter une valeur à T et E de manière que $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{\mathcal{A}_{T,E}})$ résolve le Consensus. Le Théorème 3.2.9 implique que cela revient à déterminer si le système composé des deux inéquations suivantes admet une solution :

$$n > E \geq \frac{n}{2} + \alpha \tag{3.23}$$

$$n > T \geq 2(n + 2\alpha - E) \tag{3.24}$$

De manière évidente, (3.23) et (3.24) impliquent $\alpha < \frac{n}{4}$. Réciproquement, supposons $0 \leq \alpha < \frac{n}{4}$. On pose $\alpha = \frac{n}{4} - \epsilon$, avec $\frac{n}{4} \geq \epsilon > 0$.

Si on choisit $E = n - \epsilon$, alors on obtient $2(n + 2\alpha - E) = n - 2\epsilon$, et donc $2(n + 2\alpha - E) < n$. Par conséquent, si $\alpha < \frac{n}{4}$, il est possible de trouver des valeurs pour T et E qui sont solutions de (3.23) et (3.24), et donc telles que $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{\mathcal{A}_{T,E}})$ résolve Consensus.

Cela nous conduit naturellement à nous demander quel est le "meilleur choix" de valeurs pour T et E, pour un entier donné $\alpha < \frac{n}{4}$. Au vu de $\mathcal{P}^{live}_{\mathcal{A}_{T,E}}$, la meilleure stratégie consisterait à choisir T et E aussi petits que possible. Cependant, T et E ne sont pas indépendants puisqu'on doit satisfaire la condition $T \geq 2(n+2\alpha-E)$. Les "meilleurs choix" pour T et E apparaissent ainsi paradoxaux.

De manière informelle, on remarque que dans $\mathcal{P}^{live}_{\mathcal{A}_{T,E}}$, le paramètre T joue le rôle de "seuil minimal de vivacité des communications" tandis que E joue celui de "seuil minimal de sûreté des communications" garantissant la correction d'$\mathcal{A}_{T,E}$. Puisque nous ne faisons pas d'hypothèse supplémentaire sur les communications, nous cherchons des valeurs pour T et E de sorte que $T = E$. Cela nous conduit aux deux inéquations suivantes, avec $\alpha = \frac{n}{4} - \epsilon$:

$$n > E \geq \frac{3n}{4} - \epsilon \tag{3.25}$$

$$n > E \geq 3n - 4\epsilon - 2E \tag{3.26}$$

Trivialement, l'inéquation (3.26) peut être remplacée par $E \geq n - \frac{4}{3}\epsilon$. On vérifie alors aisément qie $E = n - \frac{4}{3}\epsilon$ est une solution du système précédent, puisque $\frac{n}{4} \geq \epsilon$. Dans ce cas, on obtient $E = T = \frac{2}{3}(n + 2\alpha)$. En résumé :

Proposition 3.2.10 *Pour tout entier* $0 \leq \alpha < \frac{n}{4}$*, si* $E = \frac{2}{3}(n + 2\alpha)$*, la machine* $(\mathcal{A}_{E,E}, \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{\mathcal{A}_{T,E}})$ *résout le Consensus.*

Il est intéressant de remarquer, d'une part, que dans le cas bénin, c'est-à-dire si $\alpha = 0$, on obtient $E = T = \frac{2}{3}$ et que $\mathcal{A}_{\frac{2}{3},\frac{2}{3}}$ coïncide exactement avec *OneThirdRule*, et d'autre part que le Théorème 3.2.9 montre que les modifications apportées à *OneThirdRule* pour obtenir $\mathcal{A}_{T,E}$ ne nous ont pas fait perdre les propriétés de base de *OneThirdRule*, et ce même en présence de transmissions corrompues. En effet, comme *OneThirdRule*, l'algorithme $\mathcal{A}_{T,E}$ est relativement résilient puisqu'il ne requiert que $\mathcal{P}_\alpha$ pour être sûr. De plus, pour chaque configuration initiale, il existe au moins une exécution de $\mathcal{A}_{T,E}$ qui termine en deux rounds et, dans le cas où toutes les valeurs initiales sont égales, il existe une exécution qui termine en un round. Comme l'est *OneThirdRule*, l'algorithme $\mathcal{A}_{T,E}$ est donc *"rapide"* au sens de [29].

3.3 Algorithmes uniformément coordonnés

Comme le font remarquer Charron-Bost et Schiper, la correction de *UniformVoting* requiert que la collection des ensembles d'écoute satisfasse une propriété invariante ($\mathcal{P}_{nosplit}$), l'existence de mauvaises périodes au cours desquelles cette propriété n'est pas vérifiée pouvant avoir comme conséquence une violation des clauses de sûreté du Consensus.

L'algorithme *OneThirdRule*, quant à lui, est toujours sûr (sous $\mathcal{P}_\alpha$), en ce sens que les clauses d'Accord et d'Intégrité ne peuvent pas être violées, et ce sans qu'aucune hypothèse sur la vivacité des communications soit nécessaire.

Une autre manière de se défaire de cette propriété invariante sur la vivacité des communications consiste à avoir recours à des coordinateurs qui ont en charge de collecter les valeurs des autres processus et d'effectuer un vote pour l'une d'entre elles s'ils ont connaissance du fait qu'un "assez grand nombre" de processus la considèrent comme acceptable. D'autre part, les processus ne considèrent une nouvelle valeur comme acceptable que s'ils ont reçu un vote pour cette valeur de la part du coordinateur. En supposant que cet "assez grand nombre" est en fait une majorité stricte, on peut garantir, dans le cas bénin, que les exécutions sont sûres. Ainsi, aucune vivacité des communications n'est requise pour assurer qu'aucune exécution ne viole les clauses d'Accord ou d'Intégrité.

Charron-Bost et Schiper présentent deux algorithmes coordonnés, adaptations respectives d'un algorithme de Dwork et al. [19] et de l'algorithme *Paxos* de Lamport [28]. Nous nous intéressons tout d'abord au premier. Une section sera, par la suite, consacrée au second.

3.3.1 Algorithme *DLS*

Dans [19], Dwork *et al.* présentent un algorithme dont le but est de garantir les clauses de sûreté du Consensus quelle que soit la qualité des communications, et d'assurer la clause de Terminaison dans le cas où existerait une "bonne période" suffisamment longue. L'Algorithme *DLS*, donné comme Algorithme 3.6, en est la version HO.

Algorithme 3.6 Algorithme $DLS(f < n/2)$

```
1:  Initialisation :
2:    Acceptable_p := {v_p}
3:    Proper_p := {v_p}
4:    vote_p := ⊥
5:    Lock_p := ∅

6:  Round r = 4k − 3 :
7:    S_p^r :
8:      Envoyer ⟨Acceptable_p⟩ à coord(k)

9:    T_p^r :
10:     if p = coord(k) et ≥ n − f messages
        reçus contiennent une même valeur then
11:       vote_p := l'une de ces valeurs

12: Round r = 4k − 2 :
13:   S_p^r :
14:     if p = coord(k) et vote_p ≠ ⊥ then
15:       envoyer ⟨vote_p⟩ à tous

16:   T_p^r :
17:     if un message ⟨v⟩ est reçu de coord(k) then
18:       Lock_p := Lock_p \ {v, −} ;
19:       Lock_p := Lock_p ∪ {(v, k)} ;
20:       Acceptable_p := {v}

21: Round r = 4k − 1 :
22:   S_p^r :
23:     if ∃v s.t. (v , k) ∈ Lock_p then
24:       envoyer ⟨ack⟩ à coord(k)

25:   T_p^r :
26:     if p = coord_k et ≥ f + 1 messages ack sont
        reçus then
27:       DECIDE(vote_p) ;
28:     vote_p := ⊥

29: Round r = 4k − 1 :
30:   S_p^r :
31:     envoyer ⟨Lock_p⟩ à tous

32:   T_p^r :
33:     for all (v, θ) ∈ Lock_p do
34:       if un message contenant (w, θ̄) t.q. w ≠ v et
          θ̄ ≥ θ est reçu then
35:         Lock_p := Lock_p ∪ {(w, θ̄)} \ {(v, θ)} ;
36:         Acceptable_p := Acceptable_p ∪ {w} \ {v}
37:     if Lock_p = ∅ then
38:       Acceptable_p := Proper_p
```

Les exécutions sont décomposées en phases de quatre rounds et incluent la stratégie du coordinateur tournant, que nous avons évoquée dans le Chapitre 1, qui assure que chaque phase est uniformément coordonnée, c'est-à-dire durant laquelle les processus du système ont tous le même coordinateur. Pour tout entier $k > 0$, on note $coord(k)$ le coordinateur de la phase k.

De manière informelle, chaque processus p met à jour une variable $Proper_p$, ainsi qu'une variable $Acceptable_p \subseteq Proper_p$. L'ensemble $Proper_p$ contient l'ensemble des valeurs initiales dont p a connaissance et qui sont donc de potentielles valeurs de décision. Le processus p joint son ensemble $Proper_p$ à chacun des messages qu'il envoie et y inclut tous les ensembles $Proper_q$ qu'il reçoit. Un processus p peut verrouiller (*lock*) une valeur s'il pense "raisonnablement" qu'elle peut être décidée. Moralement, une valeur v est acceptable pour p si elle est propre et si p n'a de verrou sur aucune valeur sauf possiblement v.

Au premier round de chaque phase, chaque processus envoie son ensemble de valeurs acceptables au coordinateur qui, s'il reçoit au moins $n-f$ messages contenant une valeur commune v, vote pour v et envoie son vote à tous au round suivant.

Si un processus p reçoit un vote pour v de la part du coordinateur au deuxième round de la phase, alors p verouille v, enlève tous les verrous antérieurs et envoie un accusé de réception au coordinateur au début du troisième round.

Si le coordinateur reçoit au moins $f+1$ accusés de réception, alors il décide v.

Au dernier round de chaque phase, chaque processus envoie à tous les autres l'ensemble des valeurs sur lesquelles il a un verrou et déverrouille toutes ses valeurs en ne conservant que celle qu'il a verrouillée le plus récemment.

Dwork *et al.* ont démontré qu'aucune exécution de cet algorithme ne viole jamais les clauses d'Accord et d'Intégrité du Consensus et que la Terminaison est assurée s'il existe un round GST tel que tout message envoyé par un processus correct à un round postérieur à GST est reçu à ce même round, ce qui correspond au prédicat de communication suivant :

$$\exists GST > 0,\ \exists \Pi_0\ :\ (|\Pi_0| > n/2)\ \wedge\ (\forall r \geq GST\ :\ \Pi_0 \subseteq K(r)).$$

3.3.2 Algorithme DLS_{vf}

Si on veut transformer *DLS* afin de pouvoir tolérer des erreurs de transmission par valeurs, il est important de noter plusieurs points.

Tout d'abord, la procédure de mise à jour des ensembles $Proper_p$ doit être modifiée si l'on veut qu'ils ne contiennent que des valeurs initiales. En effet, si on autorise des corruptions de messages, la réception d'un ensemble

de valeurs prétendues propres n'implique pas qu'elles le soient réellement. Dwork *et al.* ont donc décrit une nouvelle procédure de mise à jour, que nous donnons comme Algorithme 3.7 pour plus de lisibilité. Il est important ici de comprendre que nous ne décrivons pas dans le style HO un des deux algorithmes décrits par Dwork *et al.* pour résoudre Consensus dans le cas byzantin. Nous nous concentrons sur les transformations à apporter à celui qui est décrit pour le cas bénin pour le rendre tolérant aux erreurs par valeurs.

Algorithme 3.7 Procédure Update($Proper_p$)

1: **if** $|HO(p,r)| \geq 2\alpha + 1$ et $\nexists v \in V$ t.q. v au moins $\alpha + 1$ fois **then**
2: $\quad Proper_p := V$

3: **if** une valeur v est reçue au moins $\alpha + 1$ fois **then**
4: $\quad Proper_p := Proper_p \cup \{v\}$

Le premier test permet d'assurer que si toutes les valeurs initiales sont égales à une même valeur v alors, sous $\mathcal{P}_\alpha$, aucun processus ne peut mettre une valeur différente de v dans son ensemble $Proper_p$.

Le deuxième test force, sous $\mathcal{P}_\alpha$, chaque processus p à n'ajouter à son ensemble $Proper_p$ que des valeurs qui sont dans au moins un autre ensemble $Proper_q$.

Cette procédure permet ainsi de garantir que tous les ensembles $Proper_p$ ne contiennent que des valeurs initiales.

Le deuxième point critique concerne les verrouillages et déverrouillages de valeurs au cours du dernier round d'une phase. En effet, dans le cas bénin, la reception d'un message contenant une valeur verrouillée avec un numéro de phase donné suffit à garantir que le coordinateur de la phase en question a effectivement voté pour cette valeur. Ce n'est plus la cas si on tolère des erreurs de transmission par valeurs.

Nous prenons le parti d'anéantir toute possibilité de verrouillage au cours du dernier round d'une phase, en ne permettant aux processus que de verrouiller une valeur reçue au deuxième round de la part du coordinateur, et dont il estime par conséquent qu'elle a été votée par lui. Le dernier round de chaque phase n'est dès lors plus consacré qu'aux possibles enlèvements de verrous. Un processus qui a un verrou sur une valeur v, avec un numéro de phase associé k, et qui reçoit au moins $\alpha + 1$ valeurs, différentes de v, verrouillées avec un numéro de phase $k' > k$, enlève son verrou sur v. Sous $\mathcal{P}_\alpha$, cela garantit qu'au moins un processus a effectivement verrouillé une

Algorithme 3.8 Procédure Update($Acceptable_p$)

```
1: if il existe v ∈ V et l ≥ 1 s.t. Lock_p = {(v, l)} then
2:    Acceptable_p := {v}
3: else
4:    if Lock_p = ∅ then
5:       Acceptable_p := Proper_p
```

valeur avec un numéro de phase associé strictement plus grand, et donc que le verrou de p sur v n'est plus d'actualité.

Pour plus de lisibilité, la procédure de mise a jour des ensembles $Acceptable_p$ est décrite par l'Algorithme 3.8.

Enfin, le dernier problème réside dans le fait que, dans le cas bénin, la réception d'un message $\langle vote = v \rangle$ du coordinateur par un processus p au deuxième round assure que le coordinateur a effectivement voté pour v au round précédent. Le processus p peut donc verrouiller v sans crainte.

Le schéma coordonné nous empêche ici de requérir de p qu'il reçoive au moins $\alpha + 1$ fois une valeur v pour pouvoir la verrouiller. En effet, p ne peut recevoir un tel message que de la part du coordinateur qui, pour sa part, n'envoie qu'un message par round à chacun des autres processus.

Nous sommes contraints de supposer qu'en plus de $\mathcal{P}_\alpha$, toute exécution de DLS_{vf} satisfait le prédicat de communication $\mathcal{P}_{cc}$, qui garantit que le coordinateur d'une phase k n'appartient à aucun des ensembles d'écoute altérés du round $4k - 2$.

$$\mathcal{P}_{cc} \ :: \qquad \forall k > 0 \ : \qquad coord(k) \notin AS(4k-2)$$

Preuve de correction

Pour commencer, nous démontrons qu'aucun processus ne peut avoir de verrou sur une valeur avec un numéro de phase associé strictement supérieur à celui de la phase courante.

Lemme 3.3.1 *Pour tous entiers $k \geq 1$ et $k' \geq 1$ tels que $k' > k$, pour tout round $r \leq 4k$ et pour tout processus $p \in \Pi$, il n'existe pas de valeur $v \in V$ telle que $(v, k') \in Lock_p$.*

Démonstration: Pour tout entier $k \geq 1$ et pour tout processus p, le code implique que p ne peut verrouiller une valeur v avec numéro de phase associé

Algorithme 3.9 Algorithme DLS_{vf}

```
1:  Initialization :
2:    vote_p ∈ V ∪ {⊥}, initially ⊥
3:    Proper_p ⊆ V, initially Proper_p = {v_p}
4:    Acceptable_p ⊆ Proper_p, initially {v_p}
5:    Lock_p, initially ∅

6:  Round r = 4k − 3 :
7:    S_p^r :
8:      send ⟨ v_p, Proper_p, Acceptable_p ⟩ to all processes

9:    T_p^r :
10:     Update(Proper_p)
11:     if p = coord(k) and ≥ T messages reçus
        contiennent une valeur acceptable commune then
12:       vote_p := select one of these common values

13: Round r = 4k − 2 :
14:   S_p^r :
15:     if p = coord(k) and vote_p ≠ ⊥ then
16:       send ⟨ v_p, Proper_p, vote_p ⟩ to all processes
17:     else
18:       send ⟨ v_p, Proper_p ⟩ to all processes

19:   T_p^r :
20:     Update(Proper_p)
21:     if ⟨−, −, v⟩ is received from coord(k) then
22:       Lock_p := {(v, k)}
23:       Proper_p := Proper_p ∪ {v}
24:       Update(Acceptable_p)

25: Round r = 4k − 1 :
26:   S_p^r :
27:     send ⟨ v_p, Proper_p ⟩ to all processes
28:     if il existe v telle que (v, k) ∈ Lock_p then
29:       send ⟨ack⟩ to coord(k)

30:   T_p^r :
31:     Update(Proper_p)
32:     if p = coord(k) et vote_p ≠ ⊥ et
        au moins E messages ⟨ack⟩ reçus then
33:       Decide(vote_p)

34: Round r = 4k :
35:   S_p^r :
36:     send ⟨ v_p, Proper_p, Lock_p ⟩ to all processes

37:   T_p^r :
38:     Update(Proper_p)
39:     if Lock_p = {(w_p, k_p)} et au moins α+1 messages
        ⟨{(w_q, k_q)}⟩ reçus t.q. pour tout q, k_q ≥ k_p then
40:       Lock_p := ∅
41:       Acceptable_p := Proper_p
```

k que si p reçoit un message $\langle -, -, v\rangle$ de la part de $coord(k)$ au round $4k-2$. Comme aucun processus ne dévie de sa fonction de transition, on en déduit qu'aucun ne peut verrouiller une valeur v avec un numéro de phase associé strictement supérieur à celui de la phase courante. □

Nous montrons ensuite que, si $\mathcal{P}_{cc}$ est satisfait, alors deux valeurs distinctes ne peuvent pas être verrouillées avec un même numéro de phase.

Lemme 3.3.2 *Dans toute exécution de la machine $(DLS_{vf}, \mathcal{P}_{cc})$, deux valeurs distinctes ne peuvent pas être verrouillées avec un même numéro de phase associé.*

Démonstration: Soit ρ une exécution de $(DLS_{vf}, \mathcal{P}_{cc})$ et soit $k \geq 1$ un numéro de phase. Supposons qu'il existe deux processus p et q et deux rounds

r et r' de ρ tels que $Lock_p^{(r)} = \{(v,k)\}$ et $Lock_q^{(r')} = \{(v',k)\}$. Supposons, par contradiction, que v et v' sont distinctes.

Le code implique alors que p a reçu un message $\langle -,-,v\rangle$ de la part de $coord(k)$ au round $4k-2$ et que q a reçu un message $\langle -,-,v'\rangle$ de la part de $coord(k)$ à ce même round, une contradiction puisque $\mathcal{P}_{cc}$ est satisfait dans ρ. □

Notre lemme suivant garantit que, sous certaines conditions sur les paramètres T et E, une fois qu'un processus a décidé une valeur v à une phase k alors, pour toute phase postérieure, aucun processus ne peut avoir un verrou sur une valeur autre que v avec numéro de phase associé supérieur à k.

Lemme 3.3.3 *Si $T > n+2\alpha-E$, alors dans toute exécution de $(DLS_{vf}, \mathcal{P}_{cc})$ telle qu'un processus p décide une valeur v à une phase k, on a :*

$$\forall k' \geq k \ : \left| \begin{array}{l} |\{q \in \Pi : \quad Lock_q^{(4k')} = \{(v,k_q)\},\ k_q \geq k\}| \geq E-\alpha \\ \\ |\{q \in \Pi : \quad Lock_q^{(4k'-1)} = \{(w_q,k_q)\},\ w_q \neq v,\ k_q \geq k\}| = 0 \end{array} \right.$$

Démonstration: Soit ρ une exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_{\alpha})$ telle qu'une décision est prise. Soit p un processus qui décide une valeur v à une phase k. On raisonne par récurrence sur $k' \geq k$.

- *Cas de base : $k' = k$.* Puisque p décide v au round $4k-1$, le code implique que $vote_p^{(4k-3)} = v$. Puisque $\mathcal{P}_{cc}$ est satisfait dans ρ, aucun processus ne peut recevoir de message $\langle -,-,v'\rangle$, $v' \neq v$, de la part de p au round $4k-2$. On en déduit qu'aucun processus ne verrouille une valeur différente de v à la phase k. Ainsi

  $$|\{q \in \Pi : \quad Lock_q^{(4k-1)} = \{(w_q,k_q)\},\ w_q \neq v,\ k_q k\}| = 0.$$

  De plus, le Lemme 3.3.1 assure que

  $$|\{q \in \Pi : \quad Lock_q^{(4k-1)} = \{(w_q,k_q)\},\ w_q \neq v,\ k_q > k\}| = 0.$$

  Par conséquent, on a bien

  $$|\{q \in \Pi : \quad Lock_q^{(4k-1)} = \{(w_q,k_q)\},\ w_q \neq v,\ k_q \geq k\}| = 0.$$

D'autre part, puisque p décide au round $4k-1$, le code implique que p a reçu au moins E messages $\langle ack \rangle$ à ce round. Sous $\mathcal{P}_\alpha$, on en déduit qu'au moins $E-\alpha$ processus ont effectivement envoyé un tel message, ce qui implique qu'ils ont reçu un message $\langle -, -, v' \rangle$ de la part de p au round $4k-2$. Puisque $\mathcal{P}_{cc}$ est satisfait, on a nécessairement $v' = vote_p^{(4k-3)} = v$. Le code implique alors que tous ces processus ont verrouillé v avec numéro de phase associé k. On en conclut donc

$$|\{q \in \Pi : \quad Lock_q^{(4k-1)} = \{(v, k_q)\},\ k_q \geq k\}| \geq E - \alpha.$$

Il reste à démontrer que ce dernier résultat est toujours valable à la fin du round $4k$.
Soit donc q tel que $Lock_q^{(4k-1)} = \{(v, k)\}$. Sous $\mathcal{P}_\alpha$, puisque

$$|\{q \in \Pi : \quad Lock_q^{(4k-1)} = \{(w_q, k_q)\},\ w_q \neq v,\ k_q > k\}| = 0,$$

le processus q reçoit au plus α messages contenant une valeur différente de v, verrouillée avec un numéro de phase associé supérieur à k. Le code implique alors que $Lock_q^{(4k)} = Lock_q^{(4k-1)}$. Par conséquent, on en conclut qu'on a bien

$$|\{q \in \Pi : \quad Lock_q^{(4k)} = \{(v, k_q)\},\ k_q \geq k\}| \geq E - \alpha.$$

- *Étape de récurrence :* $k' > k$. Supposons que, pour tout entier l tel que $k \leq l < k'$, on a

$$|\{q \in \Pi : \quad Lock_q^{(4l)} = \{(v, k_q)\},\ k_q \geq k\}| \geq E - \alpha$$

et

$$|\{q \in \Pi : \quad Lock_q^{(4l-1)} = \{(w_q, k_q)\},\ w_q \neq v,\ k_q \geq k\}| = 0.$$

Pour tout processus q tel que $Lock_q^{(4(k'-1))} = \{(v, -)\}$, la procédure Update($Acceptable_q$) implique $Acceptable_q^{(4(k'-1))} = \{v\}$. On a donc

$$|\{q \in \Pi : \quad Acceptable_q^{(4(k'-1))} = \{v\}\}| \geq E - \alpha,$$

et par conséquent

$$|\{q \in \Pi : \ \exists w \neq v \text{ t.q. } w \in Acceptable_q^{(4(k'-1))}\}| \leq n-(E-\alpha) = n+\alpha-E.$$

Sous $\mathcal{P}_\alpha$, on en déduit que $coord(k')$ reçoit, au round $4k'-3$, au plus $n+2\alpha-E$ ensembles $Acceptable_q$ contenant une valeur différente de v. Ainsi, si $T > n+2\alpha-E$, le code implique que $coord(k')$ ne peut pas voter pour une autre valeur que v à ce round.

Soit à présent $q \in \Pi$. Si q reçoit un vote de la part de $coord(k')$ au round $4k'-2$, alors puisque $\mathcal{P}_{cc}$ est satisfait, il reçoit nécessairement la valeur v. Le code implique alors que q verrouille v avec numéro de phase associé k'. Si q ne reçoit pas de vote, alors $Lock_q^{(4k'-2)} = Lock_q^{(4(k'-1))}$. Par conséquent, on a

$$|\{q \in \Pi : \quad Lock_q^{(4k'-1)} = \{(v, k_q)\},\ k_q \geq k\}| \geq E - \alpha,$$

et

$$|\{q \in \Pi : \quad Lock_q^{(4k'-1)} = \{(w_q, k_q)\},\ w_q \neq v,\ k_q \geq k\}| = 0.$$

Le reste de la preuve est identique au cas de base.

□

Nous déduisons du lemme précédent que deux valeurs distinctes ne peuvent pas être décidées au cours d'une même exécution.

Proposition 3.3.4 (Accord) *Si $T > n+2\alpha-E$, alors toute exécution de la machine $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha)$ satisfait la clause d'Accord du Consensus.*

Démonstration: Soit ρ une exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha)$ telle qu'une décision est prise. Soit $k \geq 1$ la première phase de ρ au cours de laquelle un processus p décide une valeur v. Supposons qu'il existe q distinct de p qui décide une valeur v' au cours d'une phase k'. Puisqu'à chaque phase seul le coordinateur peut décider, et par définition de k, on a $k' > k$.

Si $T > n+2\alpha-E$, le Lemme 3.3.3 assure

$$|\{q \in \Pi : \quad Lock_q^{(4(k'-1))} = \{(v, k_q)\},\ k_q \geq k\}| \geq E - \alpha.$$

Par conséquent, pour toute valeur $w \neq v$, au plus $n+\alpha-E$ processus trouvent w acceptable au début de la phase k'. Sous $\mathcal{P}_\alpha$, on en déduit que $q = coord(k')$ reçoit, au round $4k'-3$, au plus $n+2\alpha-E$ ensembles $Acceptable_q$ contenant une valeur différente de v.

Si $T > n + 2\alpha - E$, le code implique alors que q ne peut voter que pour v. On en conclut ainsi $v' = v$. $\square$

Les deux lemmes suivants nous permettent d'établir une condition suffisante sur T pour que toute exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha)$ satisfasse la clause d'Intégrité.

Le premier d'entre eux démontre que toute valeur acceptable pour un processus p est nécessairement propre pour p.

Lemme 3.3.5 *Dans toute exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha)$, pour tout round r et tout processus p, $Acceptable_p^{(r)} \subseteq Proper_p^{(r)}$.*

Démonstration: Soit ρ une exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha)$. Soit $p \in \Pi$ et soit $r \geq 1$ un round de ρ.

S'il existe une valeur v et un entier $l \geq 1$ tels que $Lock_p^{(r)} = \{(v, l)\}$ alors le code de la procédure Update($Acceptable_p$) implique $Acceptable_p^{(r)} = \{v\}$. Le code de DLS_{vf} implique, de plus, que p a reçu un message $\langle -, -, v\rangle$ de $coord(l)$ au round $4l - 2$ et donc $v \in Proper_p^{(4l-2)}$. Comme enfin $Proper_p^{(r)} \subseteq Proper_p^{(4l-2)}$, on a bien $Acceptable_p^{(r)} \subseteq Proper_p^{(r)}$.

Si $Lock_p^{(r)} = \emptyset$, alors le code de la procédure Update($Acceptable_p$) implique $Acceptable_p^{(r)} = Proper_p^{(r)}$. $\square$

Le lemme suivant assure que si toutes les valeurs initiales sont égales à une valeur v, alors v est la seule valeur propre durant toute l'exécution.

Lemme 3.3.6 *Si $T \geq \alpha + 1$, alors, dans toute exécution de la machine $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha)$ telle que toutes les valeurs initiales sont égales à une valeur v, pour tout round $r \geq 1$ et pour tout processus $p \in \Pi$, on a $Proper_p^{(r)} = \{v\}$.*

Démonstration: Soit ρ une exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha)$ telle que toutes les valeurs initiales sont égales à une même valeur v. On raisonne par récurrence sur $r \geq 1$.

- *Cas de base :* $r = 1$. Soit p un processus quelconque. Puisque toutes les valeurs initiales sont égales à v, et puisque $\mathcal{P}_\alpha$ est satisfait dans ρ, le processus p reçoit au plus α ensembles $Proper_q$ contenant une valeur différente de v.
  De plus, si $|HO(p,r)| \geq 2\alpha + 1$ alors, sous $\mathcal{P}_\alpha$, p reçoit au moins $\alpha + 1$ fois la valeur v au round 1.
  Par conséquent, le code de la procédure Update($Proper_p$) assure que p ne modifie pas son ensemble $Proper_p$ à ce round. Ainsi, $Proper_p^{(1)} = \{v\}$.

- *Étape de récurrence :* $r > 1$. Supposons que pour tout round r' tel que $1 \leq r' < r$, pour tout processus $q \in \Pi$, on ait $Proper_q^{(r')} = \{v\}$. Soit p un processus quelconque.

  Si r est le deuxième round d'une phase k, sous $\mathcal{P}_\alpha \wedge \mathcal{P}_{cc}$, la combinaison de l'hypothèse de récurrence et du Lemme 3.3.5 implique que $coord(k)$ a reçu, au round $r-1$, au plus α message $\langle -, -, Acceptable_q \rangle$ tels que $Acceptable_q \neq \{v\}$.
  Si $T \geq \alpha + 1$, le code implique que $coord(k)$ ne peut pas voter pour une autre valeur que v à ce round. Sous $\mathcal{P}_{cc}$, on en déduit que p ne peut pas exécuter la ligne 23 pour une autre valeur que v au round r.
  De plus, sous $\mathcal{P}_\alpha$, le processus p reçoit au plus α ensembles $Proper_q$ contenant une valeur différente de v à ce round. Si $|HO(p,r)| \geq 2\alpha + 1$ alors, sous $\mathcal{P}_\alpha$, p reçoit au moins $\alpha + 1$ ensembles $Proper_q$ contenant v.
  Par conséquent, l'exécution de la procédure Update($Proper_p$) au round r laisse $Proper_p$ inchangé.
  On en conclut finalement $Proper_p^{(r)} = Proper_p^{(r-1)} = \{v\}$.

  Si r n'est pas le deuxième round d'une phase, alors on montre de la même manière que l'exécution de la procédure Update($Proper_p$) au round r laisse $Proper_p$ inchangé.
  On a ainsi $Proper_p^{(r)} = Proper_p^{(r-1)} = \{v\}$.

□

La combinaison des deux lemmes précédents nous permet d'énoncer la proposition suivante :

$$\forall k > 0,\ \exists k_0 \geq k : \begin{cases} \forall p \in \Pi,\ SK(4k_0) = \Pi \\ |SHO(coord(k_0+1), 4k_0+1)| \geq T + \alpha \\ TT(coord(k_0+1), 4k_0+2) = \Pi \\ |SHO(coord(k_0+1), 4k_0+3)| \geq E \end{cases}$$

$$\wedge$$

$$\forall k > 0,\ \forall p \in \Pi,\ \exists k_p \geq k : \begin{cases} p = coord(k_p) \\ |SHO(p, 4k_p - 3)| \geq T \\ TT(p, 4k_p - 2) = \Pi \\ |SHO(p, 4k_p - 1)| \geq E \end{cases}$$

FIG. 3.6 – Prédicat $\mathcal{P}^{live}_{DLS_{vf}}$

Proposition 3.3.7 (Intégrité) *Si $T \geq \alpha + 1$ alors, si toutes les valeurs initiales sont égales à une même valeur v, la seule valeur de décision possible est v.*

Démonstration: Soit ρ une exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha)$ telle que toutes les valeurs initiales sont égales à une même valeur v. Supposons qu'il existe un processus p qui décide une valeur v' au cours d'une phase $k \geq 1$.

Sous $\mathcal{P}_\alpha \wedge \mathcal{P}_{cc}$, les Lemmes 3.3.5 et 3.3.6 impliquent que pour tout processus q, $Acceptable_q^{(4(k-1))} = \{v\}$, et donc que p reçoit, au round $4k-3$, au plus α ensembles $Acceptable_q$ contenant une valeur différente de v.

Si $T \geq \alpha + 1$, le code implique alors que p ne peut pas voter pour une autre valeur que v. On en conclut donc $v' = v$. □

Nous venons de déterminer des conditions sur les paramètres T et E qui garantissent que toute exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha)$ est sûre pour le Consensus.

Pour la vivacité, nous introduisons le prédicat de communication $\mathcal{P}^{live}_{DLS_{vf}}$, donné en Figure 3.6. La première partie force l'uniformisation des ensembles $Proper_p$ à une phase ainsi que la décision du coordinateur de la phase suivante.

La seconde assure, pour chaque processus, l'existence d'une phase dont il est le coordinateur et telle que les communications sont assez vivaces pour lui permettre de décider.

Lemme 3.3.8 *Si $n \geq 2\alpha + 1$ et $T \geq 2\alpha + 1$ alors, pour toute exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_{\alpha} \wedge \mathcal{P}^{live}_{DLS_{vf}})$, si $k > 0$ est une phase telle que $SK(4k) = \Pi$, alors (1) il existe une valeur v qui appartient à tous les ensembles $Proper_q^{(4k)}$, et (2) pour tout processus $p \in \Pi$, l'ensemble $Proper_p^{(4k)}$ contient toutes les valeurs verrouillées au début du round $4k$.*

Démonstration: Soit ρ une exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_{\alpha} \wedge \mathcal{P}^{live}_{DLS_{vf}})$. Soit $k > 0$ telle que $SK(4k) = \Pi$.

S'il existe une valeur v qu'au moins $\alpha + 1$ processus trouvent propre à la fin du round $4k - 1$ alors, par définition de k, tous les processus reçoivent, au round $4k$, au moins $\alpha + 1$ messages $\langle -, Proper_q, - \rangle$ tels que $v \in Proper_q$. Le code de la procédure Update($Proper_p$) assure alors que, pour tout processus p de Π, on a $v \in Proper_p^{(4k)}$.

S'il n'existe pas de telle valeur v alors, par définition de k, aucun processus ne peut recevoir $\alpha + 1$ ensembles $Proper_q$ contenant une valeur commune au round $4k$. Si $n \geq 2\alpha + 1$, le code de la procédure Update($Proper_p$) assure alors que, pour tout processus p de Π, on a $Proper_p^{(4k)} = V$, ce qui achève de montrer (1).

Soit à présent p un processus quelconque de Π. Supposons qu'il existe un processus q tel que $Lock_q^{(4k-1)} = \{(v, l)\}$ pour une certaine valeur v et un entier $l \leq k$. Le code implique que q a reçu un message $\langle -, -, v \rangle$ de $coord(l)$ au round $4l - 2$.

Puisque $\mathcal{P}_{cc}$ est satisfait, cela implique $vote_{coord_l}^{(4l-3)} = v$. Le code implique alors que $coord(l)$ a reçu au moins T ensembles $Acceptable_q$ contenant v au round $4l - 3$. Sous $\mathcal{P}_{\alpha}$, on en déduit qu'au moins $T - \alpha$ processus trouvaient v acceptable et par conséquent propre au début du round $4l - 3$.

Comme, de plus, pour tout processus q, on a $Proper_q^{(4(l-1))} \subseteq Proper_q^{(4k-1)}$, il s'ensuit qu'au moins $T - \alpha$ processus trouvent v propre au début du round $4k$. On en déduit donc que p reçoit, au round $4k$, au moins $T - \alpha$ ensembles $Proper_q$ contenant v.

Si $T \geq 2\alpha + 1$, le code de la procédure Update($Proper_p$) implique finalement $v \in Proper_p^{(4k)}$, ce qui achève de démontrer (2). □

Le lemme suivant démontre qu'à la fin d'une phase k telle que $SK(4k) = \Pi$, il existe un ensemble non vide de valeurs trouvées acceptables par au moins $n - \alpha$ processus à la fin du round $4k$.

Lemme 3.3.9 *Si $n \geq 2\alpha + 1$ et $T \geq 2\alpha + 1$ alors, pour toute exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_{\alpha} \wedge \mathcal{P}^{live}_{DLS_{vf}})$ si k est une phase telle que $SK(4k) = \Pi$, alors il existe un ensemble non-vide V_k tel que*

$$|\{q \in \Pi \ : \quad V_k \subseteq Acceptable_q^{(4k)}\}| \geq n - \alpha.$$

Démonstration: Soit ρ une exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_{\alpha} \wedge \mathcal{P}^{live}_{DLS_{vf}})$ et soit k une phase telle que $SK(4k) = \Pi$.

Si, pour tout $p \in \Pi$, on a $Lock_p^{(4k-1)} = \emptyset$ alors le code implique que, pour tout $p \in \Pi$, on a $Acceptable_p^{(4k)} = Proper_p^{(4k)}$.

Si $n \geq 2\alpha + 1$ et $T \geq 2\alpha + 1$ alors le Lemme 3.3.8 assure qu'il existe une valeur v qui appartient à tous les ensembles $Proper_p^{(4k)}$. On en déduit donc que v appartient à tous les ensembles $Acceptable_p^{(4k)}$.

S'il existe un processus p' tel que $Lock_{p'}^{(4k-1)} = \{(v, l)\}$ pour une certaine valeur v et un entier $l \leq k$, alors on définit les ensembles L_0 et L_1 de la manière suivante :

- $L_0 = \{q \in \Pi : \quad Lock_q^{(4k-1)} = \{(w_q, k_q)\}\}$,
- $L_1 = \{q \in L_0 : \quad |\{q' \in L_0 : \ w_{q'} \neq w_q, \ k_{q'} > k_q\}| \leq \alpha\}$

Soit $q_1 \in L_1$ l'un des processus tels que $k_{q_1} = \min\{k_{q'} : \ q' \in L_1\}$. Nous voulons montrer que

$$|\{q \in \Pi \ : \quad Lock_q^{(4k)} : \{(w_q, k_q)\}, \ w_q \neq w_{q_1}\}| \leq \alpha.$$

Supposons, par contradiction, qu'il existe au moins $\alpha + 1$ processus q qui ont un verrou sur une autre valeur que w_{q_1} à la fin du round $4k$. Le code implique que tous ces processus appartiennent à L_0 puisqu'aucun verrou n'est mis au round $4k$.

Par définition de q_1, il existe donc un processus $q_0 \in L_0$ tel que $w_{q_0} \neq w_{q_1}$ et $k_{q_0} \leq k_{q_1}$.

Puisque $\mathcal{P}_{cc}$ est satisfait dans ρ, le Lemme 3.3.2 implique $k_{q_0} < k_{q_1}$. Ainsi, par définition de q_1, on en déduit

$$|\{q \in L_0 \ : \ w_q \neq w_{q_0} \ \wedge \ k_q > k_{q_0}\}| \geq \alpha + 1.$$

Puisque $SK(4k) = \Pi$, le code implique finalement $Lock_{q_0}^{(4k)} = \emptyset$.

On a montré $|\{q \in \Pi \ : \ Lock_q^{(4k)} : \{(w_q, k_q)\}, \ w_q \neq w_{q_1}\}| \leq \alpha$. Par conséquent, à la fin du round $4k$, au moins $n-\alpha$ processus ont, soit un verrou sur w_{q_1}, soit aucun verrou. Le code de la procédure Update($Acceptable_p$) et le Lemme 3.3.8 assurent alors qu'au moins $n - \alpha$ processus trouvent w_{q_1} acceptable à la fin du round $4k$. □

La proposition suivante démontre que toute exécution de la machine $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{DLS_{vf}})$ satisfait la clause de Terminaison du Consensus.

Proposition 3.3.10 (Terminaison) *Si $n \geq T+\alpha$, $T \geq 2\alpha+1$ et si $n \geq E$ alors, dans toute exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{DLS_{vf}})$, tous les processus finissent par décider.*

Démonstration: On note tout d'abord que $n \geq T + \alpha$ et $n \geq E$, l'ensemble des collections $\big(HO(p,r); SHO(p,r)\big)_{p\in\Pi, r>0}$ qui satisfont $\mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{DLS_{vf}}$ est non-vide. Soit donc ρ une exécution de la machine $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{DLS_{vf}})$. Soit k_0 la première phase de ρ qui satisfait la première partie de $\mathcal{P}^{live}_{DLS_{vf}}$.

Si $n \geq T$ et $T \geq 2\alpha + 1$, le Lemme 3.3.9 assure qu'il existe une valeur v telle que

$$|\{q \in \Pi \ : \ v \in Acceptable_q^{(4k)}\}| \geq n - \alpha.$$

Soit $p_0 = coord(k_0 + 1)$. Puisque $|SHO(p_0, 4k_0 + 1)| \geq T + \alpha$ et que $\mathcal{P}_\alpha$ est satisfait dans ρ, le processus p_0 reçoit au round $4k_0 + 1$ au moins T ensembles $Acceptable_q$ contenant v. Le code implique alors que p_0 vote pour v à ce round.

Comme $TT(p_0, 4k_0 + 2) = \Pi$ et que $\mathcal{P}_{cc}$ est satisfait dans ρ, tous les processus reçoivent un message $\langle -, -, v\rangle$ de p au round $4k_0+2$. Par conséquent, le code implique que tous les processus envoient un message $\langle ack\rangle$ à p_0 au round $4k_0 + 3$.

Enfin, puisque $|SHO(p_0, 4k_0+3)| \geq E$, p_0 reçoit au moins E tels messages au round $4k_0 + 3$. Le code implique finalement que p_0 décide v à ce round.

D'autre part, puisque tous les processus reçoivent le vote de p_0 au round $4k_0 + 2$ on a, pour tout $q \in \Pi$, $Acceptable_q^{(4k_0+2)} = \{v\}$. Un argument similaire à celui qui est utilisé dans le Lemme 3.3.6 démontre que, pour tout $r \geq 4k_0 + 2$ et pour tout $q \in \Pi$, on a $Acceptable_q^{(r)} = \{v\}$.

Soit p_1 un processus qui n'a pas encore décidé à la fin de la phase $k_0 + 1$ et soit $k_1 > k_0 + 1$ une phase qui satisfait la deuxième partie de $\mathcal{P}_{DLS_{vf}}^{live}$ et telle que $p_1 = coord(k_1)$.

On vient de démontrer que, pour tout q de Π, on a $Acceptable_q^{(4(k_1-1))} = \{v\}$.

Puisque $|SHO(p_1, 4k_1 - 3)| \geq T$, on en déduit que p_1 reçoit au moins T ensembles $Acceptable_q$ contenant v au round $4k_1 - 3$. Le code implique alors que p_1 vote pour v à ce round.

Le reste de la preuve est identique au premier cas. □

La combinaison des Propositions 3.3.4, 3.3.7 et 3.3.10 nous fournit une condition suffisante sur les paramètres T et E pour que la machine $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}_{DLS_{vf}}^{live})$ résolve le Consensus.

Théorème 3.3.11 *Si les paramètres T et E vérifient les inéquations suivantes,*

$$n \geq T + \alpha \geq 3\alpha + 1 \tag{3.27}$$

$$n \geq E \tag{3.28}$$

$$T > n + 2\alpha - E \tag{3.29}$$

alors la machine $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}_{DLS_{vf}}^{live})$ résout le Consensus.

Démonstration: On remarque que, si T et E vérifient (3.27), (3.28) et (3.29), l'ensemble des collections $\big(HO(p,r); SHO(p,r)\big)_{p\in\Pi, r>0}$ qui satisfont $\mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}_{DLS_{vf}}^{live}$ est non-vide.

Soit donc ρ une exécution de la machine $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{DLS_{vf}})$. On vérifie trivialement que si T et E vérifient (3.27), (3.28) et (3.29), alors ils vérifient les conditions des Propositions 3.3.4, 3.3.7 et 3.3.10. □

Quelles valeurs pour T et E ?

Il nous reste, comme c'était le cas pour $\mathcal{U}_{T,M,E}$ et $\mathcal{A}_{T,E}$, à déterminer pour quels entiers α, $0 \leq \alpha \leq n$, il est possible de trouver des valeurs pour T et E de manière que $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{DLS_{vf}})$ résolve le Consensus.

Le Théorème 3.3.11 nous indique que cela revient à déterminer, pour tout entier α, $0 \leq \alpha \leq n$, si le système suivant admet des solutions :

$$n \geq T + \alpha \geq 3\alpha + 1 \tag{3.30}$$

$$n \geq E \tag{3.31}$$

$$T > n + 2\alpha - E \tag{3.32}$$

De manière évidente, (3.30) implique $\alpha \leq \frac{n-1}{3}$ tandis que (3.31) et (3.32) impliquent $\alpha < \frac{n}{2}$. Réciproquement, on vérifie aisément que si $\alpha = \frac{n-1}{3}$ alors $T = 2\alpha + 1$ et $E = n$ sont solutions du système.

Par conséquent, pour tout entier $\alpha \leq \frac{n-1}{3}$, il existe des valeurs pour T et E telles que $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{DLS_{vf}})$ résout Consensus.

Il est important de remarquer qu'en ce qui concerne la sûreté, il suffit de trouver des valeurs pour T et E qui soient solutions de l'inéquation (3.32), ce qui implique $\alpha < \frac{n}{2}$.

Réciproquement, si on pose $\alpha = \frac{n}{2} - \epsilon$, avec $0 < \epsilon \leq \frac{n}{2}$, on a trivialement que $T = E = n - \epsilon$ est une solution de (3.32).

Ainsi, il existe des valeurs pour T et E telles que toute exécution de $(DLS_{vf}, \mathcal{P}_{cc} \wedge \mathcal{P}_\alpha \wedge \mathcal{P}^{live}_{DLS_{vf}})$ soit sûre en présence d'une minorité de pannes.

3.4 Algorithmes coordonnés

Dans [28], Lamport décrit un algorithme, qu'il appelle *Paxos*, dont la forme est très semblable à celle de l'algorithme de Dwork *et al.* [19]. Les

exécutions sont décomposées en phases de quatre rounds et ont recours à des coordinateurs. Les coordinateurs votent pour des valeurs s'ils ont reçu suffisamment de messages et envoient leurs votes aux autres processus. Après réception d'un vote, un processus envoie un accusé de réception à son coordinateur. Si assez d'accusés de réception sont reçus, alors le coordinateur confirme son vote et l'envoie de nouveau à tous les autres qui, s'ils le reçoivent, décident la valeur reçue.

Deux différences majeures sont pourtant à noter. Tout d'abord, la sûreté de *Paxos* ne requiert pas que les phases soient uniformément coordonnées, c'est-à-dire menées par un coordinateur commun à tous les processus, tandis que l'algorithme original de [19] repose sur la stratégie du coordinateur tournant qui assurait cette propriété[2].

D'autre part, alors que le coordinateur d'une phase d'une exécution de l'algorithme de Dwork *et al.* vote pour une valeur "majoritaire", c'est-à-dire une valeur reçue par une stricte majorité de processus au premier round de la phase, un coordinateur dans *Paxos* vote pour la valeur reçue la plus *récente*.

3.4.1 Algorithme *LastVoting*

Charron-Bost et Schiper présentent un algorithme coordonné original, inspiré de *Paxos*. Cet algorithme, appelé *LastVoting*, est donné en Figure 3.10. Le premier round de chaque phase du *Paxos* original, qui assure la coordination uniforme de la phase, a été supprimé. *LastVoting* tolère donc de multiples coordinateurs au cours d'une même phase.

De manière informelle, une exécution de *LastVoting* se déroule comme suit : chaque processus p met à jour une variable x_p qui désigne sa "préférence" concernant les valeurs initiales, ainsi qu'un compteur ts_p qui désigne la phase au cours de laquelle p choisit sa valeur courante de x_p comme étant sa préférence.

Au premier round de chaque phase, p envoie sa préférence x_p ainsi que la valeur associée de ts_p à son coordinateur. Si un coordinateur reçoit à ce round une majorité stricte de messages "non-vides", alors il choisit une valeur parmi celles dont le numéro de phase associé est le plus grand, et vote pour

[2]Charron-Bost et Schiper montrent qu'en fait l'algorithme de Dwork *et al.* supporte d'autres politiques de choix des coordinateurs, et que l'existence de multiples coordinateurs dans *Paxos* est empêchée dès le premier round au cours duquel les processus sont forcés de choisir un coordinateur commun.

cette valeur[3].

Au deuxième round, un coordinateur qui a voté pour une valeur au round précédent envoie son vote à tous les autres. Un processus qui reçoit un vote de son coordinateur adopte la valeur votée comme étant sa nouvelle préférence et affecte à sa variable ts_p le numéro de la phase courante.

Au troisième round, chaque processus qui a reçu un vote de son coordinateur au round précédent lui envoie un accusé de réception. Tout coordinateur qui reçoit une majorité stricte d'accusés de réception[4] confirme son vote et l'envoie à tous au round suivant.

Au dernier round, tout processus qui reçoit un vote confirmé de son coordinateur décide la valeur votée.

Charron-Bost et Schiper montrent que cet algorithme est toujours sûr, même en présence de coordinateurs multiples dans une même phase. En effet, une fois qu'une valeur v est décidée, la procédure de vote garantit alors qu'aucun processus ne peut plus voter pour une valeur autre que v. Cependant, la Terminaison requiert l'existence d'une phase uniformément coordonnée.

3.4.2 Algorithme $\mathcal{LV}_{vf}$

Lorsque nous nous sommes attaché à modifier *LastVoting* afin de tolérer des erreurs de transmission par valeurs, nous nous sommes retrouvé confronté à plusieurs problèmes.

Tout d'abord, le fait que la décision d'un processus au cours d'une phase k soit impliquée par la réception, au round $4k$, d'un message de son coordinateur pour cette phase nous oblige, comme c'est le cas dans DLS_{vf}, à supposer que les possibles coordinateurs d'une phase ne peuvent pas appartenir à la partie altérée du quatrième round de cette phase. En d'autres termes, on est amené à requérir

$$\forall k > 0,\ \forall p \in \Pi \quad : \quad Coord(p, k) \notin AS(4k).$$

Mais alors, la possible multiplicité des coordinateurs au cours d'une même phase et l'absence d'hypothèse sur le choix des coordinateurs pour une phase

[3]Bien que l'algorithme tolère plusieurs coordinateurs au cours d'une même phase, cette procédure de vote assure qu'au plus un seul est susceptible de voter.

[4]La procédure de vote assure qu'il existe au plus un tel coordinateur.

Algorithme 3.10 Algorithme *LastVoting*

```
1:  Initialisation :
2:    x_p ∈ V, initially v_p
3:    vote_p ∈ V ∪ {?}, initialement ?
4:    commit_p un booléen, initialement false
5:    ready_p un booléen, initialement false
6:    ts_p ∈ ℕ, initialement 0

7:  Round r = 4k − 3 :
8:    S_p^r :
9:      envoyer ⟨x_p, ts_p⟩ à coord(p, k)

10:   T_p^r :
11:     if p = coord(p, k) et > n/2 messages ⟨v, θ⟩
      reçus then
12:       θ̄ := le plus grand θ parmi ⟨−, θ⟩ reçus
13:       vote_p := une valeur v t.q. ⟨v, θ̄⟩ reçu
14:       commit_p := true

15: Round r = 4k − 2 :
16:   S_p^r :
17:     if p = coord(p, k) et commit_p then
18:       envoyer ⟨vote_p⟩ à tous

19:   T_p^r :
20:     if un message ⟨v⟩ est reçu de coord(p, k) then
21:       x_p := v ;
22:       ts_p := k

23: Round r = 4k − 1 :
24:   S_p^r :
25:     if ts_p = k then
26:       envoyer ⟨ack⟩ à coord(p, k)

27:   T_p^r :
28:     if p = coord(p, k) et > n/2 messages ⟨ack⟩
      reçus then
29:       ready_p := true

30: Round r = 4k :
31:   S_p^r :
32:     if p = coord(p, k) et ready_p then
33:       envoyer ⟨vote_p⟩ à tous

34:   S_p^r :
35:     if un message ⟨v⟩ est reçu de coord(p, k) then
36:       DECIDE(v)
37:     if p = coord(p, k) then
38:       ready_p := false
39:       commit_p := false
```

donnée nous contraindrait à supposer qu'aucun processus n'appartienne à la partie altérée du quatrième round de chaque phase, ou encore

$$\forall k > 0 \quad : \quad AS(4k) = \emptyset.$$

Or, si on suppose qu'il est possible de garantir ce dernier prédicat dans un système réel, alors il est aussi possible de garantir qu'il est vrai à chaque round. Cette hypothèse impliquerait, par conséquent, qu'on ne tolère pas d'erreurs de transmission par valeurs. Nous prenons donc le parti de supposer que chaque phase est uniformément coordonnée, c'est-à-dire telle qu'un coordinateur est commun à tous les processus.

Le deuxième point délicat a été de déterminer une procédure de vote au premier round de chaque phase qui garantisse la sûreté des exécutions tout en conservant l'une des caractéristiques fondamentales de *Paxos*, à savoir que tout coordinateur qui vote choisit la valeur la plus récente à sa connaissance. Nous sommes parvenu à conserver cette notion de récence, sans toutefois garantir qu'un coordinateur vote pour "la plus récente", mais pour la plus

$$\mathcal{P}_{cc}^{\mathcal{LV}_{vf}} \quad :: \quad \forall k > 0 \ : \quad coord_k \notin AS(4k-2) \cup AS(4k)$$

FIG. 3.7 – Prédicat $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$

récente qui lui assure qu'il ne viole pas la clause d'Integrité.

Ces remarques nous ont finalement conduit à décrire l'algorithme $\mathcal{LV}_{vf}$, donné comme Algorithme 3.11, dont le code est censé être exécuté sous $\mathcal{P}_\alpha$ et $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$ donné en Figure 3.7, qui garantit que le coordinateur de chaque phase n'appartient pas aux parties altérées des deuxième et quatrième rounds de la phase qu'il coordonne.

Algorithme 3.11 Algorithme $\mathcal{LV}_{vf}$

```
1:  Initialisation :
2:    x_p ∈ V, initially v_p
3:    vote_p ∈ V ∪ {?}, initialement ?
4:    commit_p un booléen, initialement false
5:    ready_p un booléen, initialement false
6:    ts_p ∈ ℕ, initialement 0

7:  Round r = 4k − 3 :
8:    S_p^r :
9:      envoyer ⟨x_p, ts_p⟩ à coord(k)

10:   T_p^r :
11:     if p = coord(k) then
12:       if ∀θ, au plus 3α messages ⟨−, θ⟩ reçus then
13:         vote_p := ⊥
14:       else
15:         θ̄ := le plus grand θ t.q. au moins 3α + 1 messages ⟨−, θ⟩ reçus
16:         vote_p := v t.q. ⟨v, θ̄⟩ reçu le plus fréquemment
17:         commit_p := true

18: Round r = 4k − 2 :
19:   S_p^r :
20:     if p = coord(k) et commit_p then
21:       envoyer ⟨vote_p⟩ à tous

22:   T_p^r :
23:     if un message ⟨v⟩ est reçu de coord(k) then
24:       x_p := v ;
25:       ts_p := k

26: Round r = 4k − 1 :
27:   S_p^r :
28:     if ts_p = k then
29:       envoyer ⟨ack⟩ à coord(k)

30:   T_p^r :
31:     if p = coord_k et
        au moins n − α messages ⟨ack⟩ reçus then
32:       ready_p := true

33: Round r = 4k :
34:   S_p^r :
35:     if p = coord_k et ready_p then
36:       envoyer ⟨vote_p⟩ à tous

37:   S_p^r :
38:     if un message ⟨v⟩ est reçu de coord(k) then
39:       DECIDE(v)
40:     if p = coord_k then
41:       ready_p := false
42:       commit_p := false
```

Preuve de correction

Le premier de nos lemmes démontre que dans une exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ telle que toute les valeurs initiales sont égales à une même valeur v, à chaque phase $k > 0$, le coordinateur $coord(k)$ ne peut voter que pour v.

Lemme 3.4.1 *Dans toute exécution de la machine* $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ *telle que toute les valeurs initiales sont égales à une même valeur* v *on a*

$$\forall k > 0 \quad : \quad \bigcup_{k'<k} Q^{4k-3}(\langle v, k' \rangle) = \Pi.$$

Démonstration: Soit ρ une exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ telle que toutes les valeurs initiales sont égales à une même valeur v. On raisonne par récurrence sur $k > 0$.

- *Cas de base :* $k = 1$. Puisque toutes les variables ts_q sont initialement égales à 0 et que toutes les valeurs initiales sont égales, on a $Q^{4(k-1)-3}(\langle v, 0 \rangle) = \Pi$.

- *Étape de récurrence :* $k > 1$. Supposons que l'on ait

  $$\bigcup_{k'<k} Q^{4(k-1)-3}(\langle v, k' \rangle) = \Pi.$$

  Sous $\mathcal{P}_\alpha$, le processus $coord(k-1)$ reçoit, au round $4(k-1)-3$, au plus α messages $\langle v, k' \rangle$ avec $k' \geq k-1$.
  S'il n'existe pas d'entier $k' < k-1$ tel que $coord(k-1)$ reçoive au moins $3\alpha + 1$ messages $\langle -, k' \rangle$, alors il ne vote pas.
  Par conséquent, sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, le code implique qu'aucun processus q ne modifie x_q ni ts_q au round $4(k-1)-2$. Ainsi

  $$\{q \in \Pi : \quad x_q^{(4(k-1)-2)} = v \text{ et } ts_q^{(4(k-1)-2)} < k-1\} = \Pi.$$

  Le code nous permet alors de conclure $\bigcup_{k'<k} Q^{4k-3}(\langle v, k' \rangle) = \Pi$.
  Sinon, soit $\overline{k}$ le plus grand tel entier k'. Puisque tous les processus q envoient $x_q = v$ au round $4(k-1)-3$, on en déduit que, sous $\mathcal{P}_\alpha$, $coord(k-1)$ reçoit au plus α messages $\langle v', \overline{k} \rangle$ avec $v' \neq v$, et donc au moins $2\alpha + 1$ messages $\langle v, k \rangle$ à ce round. Le code implique alors que $coord(k-1)$ vote pour v.

Sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, il s'ensuit que tout processus q qui modifie x_q au round $4(k-1)-2$ lui affecte nécessairement la valeur v. On a donc

$$\{q \in \Pi : \quad x_q^{(4(k-1)-2)} = v \text{ et } ts_q^{(4(k-1)-2)} \leq k-1\} = \Pi.$$

Le code nous permet finalement de conclure $\bigcup_{k'<k} Q^{4k-3}(\langle v, k' \rangle) = \Pi$.

□

La correction de ce lemme nous autorise à énoncer la proposition suivante qui montre que toute exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ satisfait la clause d'Intégrité du Consensus.

Proposition 3.4.2 (Intégrité) *Dans toute exécution de la machine* $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ *telle que toutes les valeurs initiales sont égales à une même valeur* v*, la seule valeur de décision possible est* v*.*

Démonstration: Soit ρ une exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ telle que toutes les valeurs initiales sont égales à une même valeur v. Supposons qu'il existe un entier $k \geq 1$, un processus p et une valeur v' tels que p décide v' au cours de la phase k de ρ.

Le code implique que p a reçu un message $\langle v' \rangle$ de $coord(k)$ au dernier round de la phase k. Sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, on en déduit que $coord(k)$ a voté pour v' au round $4k-3$.

Sous $\mathcal{P}_\alpha$, le Lemme 3.4.1 assure que $coord(k)$ a reçu à ce round au plus α messages $\langle -, k' \rangle$ ave $k' \geq k$.

Soit, de plus, $\overline{k}$ le plus grand entier k tel que $coord(k)$ a reçu au moins $3\alpha+1$ messages $\langle -, \overline{k} \rangle$. Un tel entier existe puisque $coord(k)$ vote au round $4k-3$.

Sous $\mathcal{P}_\alpha$, le Lemme 3.4.1 assure alors que $coord(k)$ a reçu à ce round au plus α messages $\langle w, \overline{k} \rangle$ avec $w \neq v$, et donc au moins $2\alpha+1$ messages $\langle v, \overline{k} \rangle$. Le code implique par conséquent que $coord(k)$ a voté pour v, et donc $v' = v$.
□

Le lemme suivant démontre que si une valeur v est décidée au cours d'une phase k, alors aucun processus ne peut voter pour une valeur autre que v au cours d'une phase $k' \geq k$.

Lemme 3.4.3 *Pour toute exécution de la machine* $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$*, si une valeur* v *est décidée au cours d'une phase* $k \geq 1$*, alors on a*

$$\forall k' \geq k \quad : \quad vote_{coord_{k'}}^{(4k'-3)} \in \{v; \perp\}.$$

Démonstration: Soit ρ une exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ dans laquelle une décision est prise. Soit p un processus qui décide v au cours d'une phase k.

On raisonne par récurrence sur $k' \geq k$.

- *Cas de base :* $k' = k$. Puisque p décide v au round $4k$, le code implique sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$ que $coord(k)$ a voté pour v au round $4k - 3$.

- *Étape de récurrence :* $k' > k$. Supposons que pour tout entier l, $k \leq l < k'$, on ait $vote_{coord_l}^{(4l-3)} \in \{v; \perp\}$.

  Puisque p décide v à la phase k, le code implique que $coord(k)$ a reçu au moins $n - \alpha$ messages $\langle ack \rangle$ au round $4k - 1$.
  Sous $\mathcal{P}_\alpha$, cela implique qu'au moins $n - 2\alpha$ processus ont envoyé un tel message à ce round. Le code assure donc

  $$|\{q \in \Pi \quad : \quad x_q^{(4k-2)} = v \text{ et } ts_q^{(4k-2)} = k\}| \geq n - 2\alpha.$$

  Sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, l'hypothèse de récurrence permet de déduire que l'on a

  $$\forall l,\ k \leq l < k' :\ |\{q \in \Pi :\ x_q^{(4l-2)} = v \text{ et } k \leq ts_q^{(4l-3)} \leq l\}| \geq n - 2\alpha,$$

  et que, pour tout q tel que $ts_q^{(4(k'-1)-2)} \in [\![k, \ldots, k'-1]\!]$, on a $x_q^{(4(k'-1)-2)} = v$.

  Supposons que $coord(k')$ vote au round $4k' - 3$ pour une valeur $v' \neq \perp$. On a montré

  $$\Big| \bigcup_{\theta \notin [\![k,\ldots,k'-1]\!]} Q^{4k'-3}(\langle -, \theta \rangle) \Big| \leq 2\alpha;$$

  ce qui, sous $\mathcal{P}_\alpha$, implique que $coord(k')$ reçoit au plus 3α messages $\langle -, \theta \rangle$ avec $\theta \notin [\![k, \ldots, k' - 1]\!]$. Le code assure par conséquent que la valeur v' votée à ce round est choisie parmi celles qui sont associées à un entier $\theta \in [\![k, \ldots, k' - 1]\!]$.

Soit $\overline{\theta}$ le plus grand entier de $[\![k, \ldots, k'-1]\!]$ tel que $coord(k')$ reçoive au moins $3\alpha + 1$ messages $\langle -, \overline{\theta} \rangle$.
On a montré que, pour tout q tel que $ts_q^{(4(k'-1)-2)} \in [\![k, \ldots, k'-1]\!]$, on a $x_q^{(4(k'-1)-2)} = v$. Le code implique alors que, pour tout q tel que $ts_q^{(4(k'-1))} \in [\![k, \ldots, k'-1]\!]$, on a $x_q^{(4(k'-1))} = v$.
Sous $\mathcal{P}_\alpha$, on en déduit que $coord(k')$ reçoit au plus α messages $\langle w, \overline{\theta} \rangle$ avec $w \neq v$, et donc au moins $2\alpha + 1$ messages $\langle v, \overline{\theta} \rangle$.
Le code assure alors $v' = v$.

□

$$\forall k > 0,\ |TT(coord(k), 4k-2)| \geq 3\alpha + 1 \tag{3.33}$$

$$\wedge$$

$$\forall k > 0,\ \exists k' > k : \begin{cases} SHO(coord(k'), 4k'-3) = \Pi \\ coord(k') \in K(k') \\ HO(coord(k'), 4k'-1) = \Pi \end{cases} \tag{3.34}$$

FIG. 3.8 – Prédicat $\mathcal{P}^{live}_{\mathcal{LV}_{vf}}$

Nous déduisons du lemme précédent une proposition qui démontre que, dans toute exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$, deux valeurs distinctes ne peuvent pas être décidées.

Proposition 3.4.4 (Accord) *Toute exécution de la machine* $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ *satisfait la clause d'Accord du Consensus.*

Démonstration: Soit ρ une exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ dans laquelle une décision est prise. Soit k_0 la première phase de ρ au cours de laquelle un processus p décide une valeur v.

Supposons qu'il existe q qui décide une valeur v' au cours d'une phase k. Par définition de k_0 on a $k \geq k_0$.

Si $k = k_0$, alors le code implique que p et q ont reçu de $coord(k_0)$, au round $4k_0$, un message $\langle v \rangle$ et $\langle v' \rangle$ respectivement. Sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, le code implique $v' = v$.

Si $k > k_0$, le code implique que q a reçu au round $4k$ un message $\langle v' \rangle$ de $coord(k)$. Sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, le code implique que $coord(k)$ a voté pour v' au round $4k - 3$.

Or, sous $\mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, le Lemme 3.4.3 implique $vote_{coord(k)}^{(4k-3)} = v$. On en conclut donc $v' = v$.

□

Nous avons achevé de démontrer que toute exécution de la machine $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}})$ est sûre. Pour la vivacité, nous introduisons le prédicat de communication $\mathcal{P}_{\mathcal{LV}_{vf}}^{live}$, donné en Figure 3.8.

De manière informelle, la première partie (3.33) assure qu'au début de chaque phase, au moins $3\alpha + 1$ variables ts_q sont égales. La seconde (3.34), quant à elle, garantit l'existence d'une phase au cours de laquelle tous les processus votent.

Lemme 3.4.5 *Si $n \geq 3\alpha + 1$, alors dans toute exécution de la machine $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}} \wedge \mathcal{P}_{\mathcal{LV}_{vf}}^{live})$, pour toute phase $k \geq 1$, on a*

$$\exists k' \leq k \ : \ |\{q \in \Pi : \ ts_q^{(4k)} = k'\}| \geq 3\alpha + 1.$$

Démonstration: Soit ρ une exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}} \wedge \mathcal{P}_{\mathcal{LV}_{vf}}^{live})$. On raisonne par récurrence sur $k \geq 1$.

- *Cas de base : $k = 1$.* Si $coord(1)$ ne vote pas alors, sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, aucun processus q ne met à jour sa variable ts_q au round 2. Par conséquent, on en déduit que, pour tout q, on a $ts_q^{(4)} = 0$.
  Si $n \geq 3\alpha + 1$, alors on a bien le résultat souhaité.

  Si $coord(1)$ vote pour une valeur v alors, sous $\mathcal{P}_{\mathcal{LV}_{vf}}^{live} \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, au moins $3\alpha + 1$ processus q reçoivent un message $\langle v \rangle$ de $coord(1)$ au round 2. Le code implique alors que chacun d'entre eux affecte la valeur 1 à sa variable ts_q à ce round et ne la modifie plus jusqu'à la fin de la phase, ce qui achève de démontrer le cas de base.

- *Étape de récurrence :* $k > 1$. Si $coord(k)$ ne vote pas alors, sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, aucun processus q ne met à jour sa variable ts_q au round $4k-2$. Par conséquent, on en déduit que, pour tout q, on a $ts_q^{(4k)} = ts_q^{(4(k-1))}$. L'hypothèse de récurrence implique alors qu'il existe un entier $k' \leq k-1$ tel que
$$|\{q \in \Pi : \ ts_q^{(4k)} = k'\}| \geq 3\alpha + 1.$$
Si $coord(k)$ vote pour une valeur v alors, sous $\mathcal{P}_{\mathcal{LV}_{vf}}^{live} \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$, au moins $3\alpha+1$ processus q reçoivent un message $\langle v \rangle$ de $coord(k)$ au round $4k-2$. Le code implique alors que chacun d'entre eux affecte la valeur k à sa variable ts_q à ce round et ne la modifie plus jusqu'à la fin de la phase. On a donc
$$|\{q \in \Pi : \ ts_q^{(4k)} = k\}| \geq 3\alpha + 1,$$
ce qui achève de démontrer le résultat énoncé.

□

Nous énonçons ici notre dernière proposition qui démontre que, dans toute exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}} \wedge \mathcal{P}_{\mathcal{LV}_{vf}}^{live})$, tous les processus finissent par décider.

Proposition 3.4.6 (Terminaison) *Toute exécution de la machine* $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}} \wedge \mathcal{P}_{\mathcal{LV}_{vf}}^{live})$ *satisfait la clause de Terminaison du Consensus.*

Démonstration: Soit ρ une exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}_{cc}^{\mathcal{LV}_{vf}} \wedge \mathcal{P}_{\mathcal{LV}_{vf}}^{live})$ et soit $k \geq 1$ la première phase de ρ qui satisfait la seconde partie (3.34) de $\mathcal{P}_{\mathcal{LV}_{vf}}^{live}$. Soit $p = coord(k)$.

Sous $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}} \wedge \mathcal{P}_{\mathcal{LV}_{vf}}^{live}$, le Lemme 3.4.5 assure qu'il existe un entier $k' < k$ tel que
$$\Big| \bigcup_{v \in V} Q^{4k-3}(\langle v, k' \rangle) \Big| \geq 3\alpha + 1.$$

Par conséquent, puisque $SHO(p, 4k-3) = \Pi$, le processus p reçoit au moins $3\alpha + 1$ messages $\langle -, k' \rangle$. Le code implique alors que p vote pour une valeur que nous notons v_p.

Comme $p \in K(k)$, et puisque $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$ est satisfait dans ρ, tous les processus reçoivent un message $\langle v_p \rangle$ de p au round $4k-2$. Le code implique alors que tous envoient un message $\langle ack \rangle$ à p au round $4k-1$.

Le prédicat $\mathcal{P}^{live}_{\mathcal{LV}_{vf}}$ implique que $HO(p, 4k-1) = \Pi$. On en déduit donc que, sous $\mathcal{P}_\alpha$, p reçoit au moins $n - \alpha$ tels messages. Le code assure que p envoie un message $\langle v_p \rangle$ à tous au round $4k$.

Finalement, puisque $p \in K(k)$ et sous $\mathcal{P}^{\mathcal{LV}_{vf}}_{cc}$, il s'ensuit que tous les processus reçoivent un message $\langle v_p \rangle$ de p à ce round. Le code implique alors que tous décident au round $4k$. □

La combinaison des trois propositions précédentes nous conduit à énoncer le théorème suivant :

Théorème 3.4.7 *Si $n \geq 3\alpha+1$, alors la machine $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{\mathcal{LV}_{vf}}_{cc} \wedge \mathcal{P}^{live}_{\mathcal{LV}_{vf}})$ résout le Consensus.*

Démonstration: On vérifie aisément que, si $n \geq 3\alpha + 1$, l'ensemble des collections $\big(HO(p,r); SHO(p,r)\big)_{p\in\Pi, r>0}$ qui satisfont $\mathcal{P}_\alpha \wedge \mathcal{P}^{\mathcal{LV}_{vf}}_{cc} \wedge \mathcal{P}^{live}_{\mathcal{LV}_{vf}}$ est non-vide.

Soit ρ une exécution de $(\mathcal{LV}_{vf}, \mathcal{P}_\alpha \wedge \mathcal{P}^{\mathcal{LV}_{vf}}_{cc} \wedge \mathcal{P}^{live}_{\mathcal{LV}_{vf}})$.

Les Propositions 3.4.4 et 3.4.2 assurent respectivement que ρ satisfait les clauses d'Accord et d'Intégrité.

Si $n \geq 3\alpha + 1$, la Terminaison est établie par la Proposition 3.4.6. □

3.5 Discussion et relations avec des travaux antérieurs

De nombreux travaux se sont attachés à l'étude du Consensus dans un contexte de pannes par valeurs. Cependant, la plupart d'entre eux s'intéressent au cas des processus byzantins, et donc à des pannes *permanentes* et *statiques*. Comme nous l'avons précisé, les travaux les plus anciens sont ceux présentés dans [30, 38] pour le cas synchrone, et [19] pour le cas partiellement synchrone. Une solution fondée sur des hypothèses plus faibles sur le synchronisme des liens a été donnée dans [2].

D'autres variantes que $\mathcal{LV}_{vf}$ de l'algorithme *Paxos* ont été proposées dans [10, 31, 1, 47]. Un algorithme pour la résolution de *Fast Byzantine Paxos* existe dans [36]. Il est toutefois important de noter que la définition de Byzantine Paxos présente une clause d'Intégrité plus faible que celle considérée

dans le présent chapitre : il suffit qu'une valeur soit *proposée* par l'un des processus pour qu'elle soit considérée comme une valeur de décision autorisée. Si le processus concerné est byzantin, alors la valeur proposée n'est pas nécessairement la valeur initiale d'un des processus. Ainsi, aucun de ces algorithmes n'est, à proprement parler, une solution au problème de Consensus dans sa définition classique.

3.5.1 Correspondance avec les bornes inférieures

Santoro et Widmayer ont montré [42] qu'il était impossible de résoudre Consensus en présence de $\lfloor n/2 \rfloor$ communications corrompues par round, sous l'hypothèse que chaque processus reçoit un message de chacun des autres à chaque round. Le cas de figure problématique est celui dans lequel ces $\lfloor n/2 \rfloor$ transmissions fautives forment un bloc, c'est-à-dire qu'à chaque round, les communications incorrectes sont originaires d'un seul processus.

D'un autre côté, nous avons présenté trois algorithmes non-coordonnés $\mathcal{U}_{T,M,E}$, $\mathcal{UV}_{vf}$ et $\mathcal{A}_{T,E}$ qui autorisent respectivement $n^2/3$, $n^2/2$ et $n^2/4$ corruptions par round en général, sous l'hypothèse que chaque processus entend tous les autres à chaque round. Cet état de fait résulte de la séparation que nous avons opérée entre la Sûreté (clauses d'Accord et d'Intégrité) et la Vivacité (clause de Terminaison) de chacun de nos algorithmes, et du caractère transitoire des pannes que nous considérons. Ainsi, par exemple, il est suffisant, pour qu'$\mathcal{A}_{T,E}$ soit sûr, que chaque processus reçoive moins de $n/4$ messages corrompus à chaque round. Mais, en ce qui concerne la Terminaison, le prédicat $\mathcal{P}^{live}_{\mathcal{A}_{T,E}}$ impose l'existence de deux rounds au cours desquels les hypothèses sur les communications sont plus fortes que celle de Santoro et Widmayer.

Dans [43], Schmid *et al.* présentent une autre manière de contourner le résultat d'impossibilité de Santoro et Widmayer dans des systèmes synchrones. A chaque round et pour chaque processus correct p, ils restreignent le nombre de transmissions impliquant p (soit initiateur, soit destinataire) qui sont susceptibles d'être fautives. De cette manière, ils empêchent l'existence des blocs de fautes de Santoro et Widmayer.

Notre algorithme $\mathcal{A}_{T,E}$ est *rapide* en ce sens que, pour toute configuration initiale, il existe une exécution de la machine $(\mathcal{A}_{T,E}, \mathcal{P}_\alpha)$ dans laquelle tous les processus décident en deux rounds [29, 16]. Pour ceci, Martin et Alvisi [36] ont démontré que $\frac{4n+1}{5}$ était une borne inférieure sur le nombre de processus corrects. De la même manière que nos résultats ne sont, à strictement

parler, pas comparables avec ceux de Santoro et Widmayer, ils ne contredisent pas la borne de Martin et Alvisi. Notre modèle permet une analyse plus fine que les leurs, tant sur un plan spatial que temporel. En effet, nous raisonnons sur le nombre d'erreurs de transmission et non sur le nombre de processus concernés par elles. D'autre part, notre analyse s'effectue round par round et ne suppose pas de pannes permanentes. C'est ce qui permet à $\mathcal{A}_{T,E}$ d'être rapide tout en autorisant jusqu'à $\frac{n-1}{4}$ processus à émettre une information corrompue à chaque round. Cependant, pour assurer la Terminaison, nous sommes contraint de ne considérer que des exécutions qui garantissent l'existence d'un round au cours duquel aucune information corrompue n'est transmise.

3.5.2 Sur le recours à des coordinateurs

Dans [17], Charron-Bost et Schiper ont insisté sur le fait que le recours à un schéma coordonné dans le cas bénin permettait de s'abstraire de la propriété invariante sur la vivacité des communications requise par certains algorithmes, comme par exemple *UniformVoting*. Lorsque l'on essaie de modifier un algorithme coordonné, décrit initialement pour tolérer des pannes bénignes, dans le but de donner une solution pour Consensus dans le cas des pannes par valeurs, on se heurte à un problème majeur : un coordinateur ne peut pas être l'initiateur d'une transmission corrompue sans risquer de violer les clauses de Sûreté.

Pour faire face à ce constat, certains ont jugé nécessaire de modifier la définition de Consensus [10, 36] en affaiblissant la clause d'Intégrité. D'autres ont eu recours à des techniques permettant de masquer les pannes par valeurs et de les transformer en omissions. Comme nous l'avons dit plus haut dans le présent chapitre, Dwork, Lynch et Stockmeyer [19] ont présenté deux algorithmes coordonnés tolérant des pannes byzantines. L'un d'entre eux a recours à une primitive de communication inspirée de celle qui est présentée dans [44], qui permet de ne prendre en compte que les messages reçus non-corrompus. Cependant, cette primitive est extrêmement coûteuse en ce qui concerne le nombre de messages échangés. Leur autre algorithme est *authentifié* et utilise des *signatures digitales* permettant de vérifier l'intégrité, ou non-corruption, de l'information échangée entre les processus. Cependant, il n'existe pas de formalisme rigoureux de l'authentification. À notre connaissance, il en existe autant de définitions que d'algorithmes qui l'utilisent. Les deux précédentes approches ont comme effet de masquer les corruptions en

omissions puisqu'elles permettent aux processus de ne pas considérer les messages provenant d'un processus fautif. Quant à nous, nous avons pris le parti de faire l'hypothèse que les coordinateurs ne peuvent être à l'origine d'une transmission corrompue ($\mathcal{P}_{cc}$ ou $\mathcal{P}_{cc}^{\mathcal{LV}_{vf}}$).

Il semble donc qu'avoir recours à des coordinateurs, et donc distinguer un (ou plusieurs) processus en lui donnant un impact particulier sur le calcul, ne soit pas réellement adapté au contexte des pannes par valeurs.

Alg.	Prédicat pour la Sûreté	Prédicat pour la Vivacité	Conditions
$\mathcal{U}_{T,M,E}$	$\forall r > 0,\ \forall p \in \Pi :\ \lvert AHO(p,r)\rvert \leq \alpha$ $\wedge$ $\forall r > 0,\ \forall p \in \Pi :\ \lvert HO(p,r)\rvert \geq A$	$\forall \phi > 0,\ \exists \phi_u \geq \phi, \exists \Pi_u,\ \forall q \in \Pi :$ $HO(q, 2\phi_u - 1) = SHO(q, 2\phi_u - 1) = \Pi_u \wedge$ $(\lvert SHO(p, 2\phi_u + 1)\rvert > T) \wedge (\lvert SHO(p, 2\phi_u + 2)\rvert > E)$	$n > T \geq \frac{n}{2} + \alpha$ $n > E \geq \frac{n}{2}$ $n \geq A \geq 2\alpha + 1$ $n \geq M > \alpha$ $E + A \geq n + M + 2\alpha$
$\mathcal{UV}_{vf}$	$\forall r > 0,\ \forall p \in \Pi :\ \lvert AHO(p,r)\rvert \leq \alpha$ $\wedge$ $\forall r > 0,\ \forall p, q :\ \lvert SHO(p,r) \cap SHO(q,r)\rvert \geq \alpha + 1$	$\forall \phi > 0,\ \exists \phi' \geq \phi, \exists \Pi' \subseteq \Pi$ $\forall q \in \Pi,\ HO(q, 2\phi' - 1) = SHO(q, 2\phi' - 1) = \Pi'$ $\forall q,\ \exists \phi_q > \phi' : (AS(2\phi_q - 1) = \emptyset) \wedge (AHO(q, 2\phi_q) = \emptyset)$	
$\mathcal{A}_{T,E}$	$\forall r > 0, \forall p \in \Pi : \lvert AHO(p,r)\rvert \leq \alpha$	$\forall r_0 > 0,\ \exists r \geq r_0,\ \exists \Pi_r^1, \Pi_r^2 \subseteq \Pi :$ $(\lvert \Pi_r^1\rvert > E - \alpha) \wedge (\lvert \Pi_r^2\rvert > T) \wedge$ $(\forall p \in \Pi_r^1,\ HO(p,r) = SHO(p,r) = \Pi_r^2)$ $\forall r > 0,\ \forall p \in \Pi,\ \exists r_p > r\ :\ \lvert HO(p, r_p)\rvert > T$ $\forall r > 0,\ \forall p \in \Pi,\ \exists r_p > r\ :\ \lvert SHO(p, r_p)\rvert > E$	$n > E \geq \frac{n}{2} + \alpha$ $T \geq 2(n + 2\alpha - E)$
DLS_{vf}	$\forall r > 0, \forall p \in \Pi : \lvert AHO(p,r)\rvert \leq \alpha$ $\wedge$ $\forall k > 0 :\ coord(k) \notin AS(4k - 2)$	$\forall k > 0,\ \exists k_{\geq} k :\ SK(4k_0) = \Pi \wedge$ $\lvert SHO(coord(k_0 + 1), 4k_0 + 1)\rvert \geq T + \alpha \wedge$ $TT(coord(k_0 + 1), 4k_0 + 2) = \Pi \wedge$ $\lvert SHO(coord(k_0 + 1), 4k_0 + 3)\rvert \geq E$ $\wedge$ $\forall k > 0,\ \forall p \in \Pi,\ \exists k_p \geq k : p = coord(k_p) \wedge$ $\lvert SHO(p, 4k_p - 3)\rvert \geq T \wedge$ $TT(p, 4k_p - 2) = \Pi \wedge$ $\lvert SHO(p, 4k_p - 1)\rvert \geq E$	$n \geq T + \alpha$ $T \geq 2\alpha + 1$ $n \geq E$ $T > n + 2\alpha - E$
$\mathcal{LV}_{vf}$	$\forall r > 0, \forall p \in \Pi : \lvert AHO(p,r)\rvert \leq \alpha$ $\wedge$ $\forall k > 0 :\ coord(k) \notin AS(4k - 2) \cup AS(4k)$	$\forall k > 0 :\ \lvert TT(coord(k), 4k - 2)\rvert \geq 3\alpha + 1$ $\wedge$ $\forall k > 0,\ \exists k' > k :\ SHO(coord(k'), 4k' - 3) = \Pi \wedge$ $coord_{k'} \in K(k') \wedge$ $HO(coord(k'), 4k' - 1) = \Pi$	

Tab. 3.1 – Résumé des résultats

Chapitre 4

Algorithme $\mathcal{LR}$

Comme nous l'avons précisé dans l'introduction, il existe différentes manières d'appréhender les erreurs de transmission dans les systèmes distribués. On peut prendre le parti d'étudier les possibilités de coopération entre processus d'un système dans lequel les messages peuvent ne pas parvenir à leurs destinataires ou peuvent être corrompus ; c'est l'approche communément adoptée dans le domaine du *Calcul Distribué tolérant aux pannes*. On peut aussi travailler sous l'hypothèse que le système satisfait certaines propriétés de fiabilité des communications, et étudier comment il est possible de garantir des périodes pendant lesquelles ces propriétés sont satisfaites, comme par exemple pour les *réseaux mobiles ad-hoc* dans lesquels la topologie du graphe de communication peut être modifiée par les mouvement des nœuds composant le réseau. Nous nous intéressons dans le présent chapitre à ce second aspect.

4.1 Motivations

En 1981, Gafni et Bertsekas [23] ont introduit une technique algorithmique élégante, appelée *renversements de liens*, pour le routage des messages dans des réseaux sujets à de fréquents changements de topologie.

Dans le problème considéré, le réseau contient une destination unique, et une direction virtuelle est associée à chacun des canaux de communication. Les directions des canaux doivent assurer que le graphe de communication est acyclique et que chaque nœud du réseau dispose d'un chemin dirigé vers la destination, c'est-à-dire que le graphe est *orienté vers la destination*. Les

messages peuvent ainsi être acheminés de tout nœud du réseau vers la destination en suivant les directions associées aux canaux de communication. Si, pour une raison quelconque[1], un nœud ne dispose plus d'un chemin vers la destination, le réseau doit s'auto-ajuster afin de restaurer cette propriété. Pour ce faire, certains nœuds *renversent* les directions associées à un ou plusieurs de leurs liens incidents. Les nœuds doivent, de plus, être capables de déterminer de manière autonome, en se fondant uniquement sur une information locale, s'il leur est nécessaire d'exécuter de tels renversements.

Gafni et Bertsekas [23] ont restreint leur étude au cas des graphes acycliques et ont présenté deux algorithmes, chacun avec une description abstraite ainsi qu'une implémentation. Dans chacune des descriptions abstraites, chacun des nœuds (autre que la destination), qui devient un *puits* (c'est-à-dire dont tous les liens incidents sont entrants), renverse certains de ses liens incidents. Les deux algorithmes diffèrent dans le choix des liens renversés par les puits. Dans l'algorithme *full reversal* (*FR*), tous les liens incidents sont renversés, tandis que dans l'algorithme *partial reversal* (*PR*), un puits ne renverse que les liens qui n'ont pas été renversés depuis la dernière fois qu'il a été un puits, s'il en existe, et les renverse tous sinon. Dans les deux implémentations proposées par Gafni et Bertsekas, les nœuds mettent à jour une variable, appelée *hauteur*, à valeurs dans un ensemble totalement ordonné, telle que deux nœuds distincts ont des hauteurs différentes. Un lien entre deux nœuds est orienté du nœud de plus grande hauteur vers celui de plus basse hauteur. Les renversements des liens sont alors implémentés par l'accroissement des hauteurs des nœuds considérés. Les hauteurs dans l'implémentation de *FR* sont des paires ordonnées, dont la première composante est un compteur non-borné, et la deuxième est l'identifiant unique du nœud. Les hauteurs dans l'implémentation de *PR* sont des triplets ordonnés dont les deux premières composantes sont des compteurs non-bornés, et la troisième est l'identifiant unique du nœud. Un nœud dont aucun lien incident n'est sortant applique une fonction, qui prend en arguments sa hauteur courante et celle de ses voisins, pour calculer sa nouvelle hauteur. Dans l'implémentation de *full reversal*, la fonction consiste simplement à rendre le compteur du nœud plus grand que les compteurs de tous ses voisins, alors que celle de *partial reversal* fait croître les compteurs d'une manière plus fine.

Gafni et Bertsekas ont démontré que leurs deux implémentations étaient

[1]Arrêt du fonctionnement d'un processus, changement dans la topologie du graphe de communication, etc.

correctes, c'est-à-dire que toutes deux terminent et, une fois que plus aucun nœud n'effectue de renversement, le graphe obtenu est acyclique et orienté vers la destination. Dans leur preuve de correction, ils considèrent une généralisation des hauteurs et des fonctions qui les manipulent et démontrent le résultat dans le cas général.

De nombreux travaux ont utilisé cette méthode de renversements de liens pour des applications variées dans les réseaux dynamiques. Certains se sont penchés sur le problème de la k-exclusion mutuelle [41, 46, 45], dans lequel les n processus d'un système doivent accéder de façon intermittente à une même partie de code appelée *section critique*. Au plus k, $1 \leq k \leq n$, peuvent se trouver en section critique simultanément. Les différents algorithmes proposés pour résoudre ce problème utilisent des *jetons*, qui définissent des privilèges entre les processus. Un processus ne peut accéder à la section critique que s'il est en possession d'un jeton. La méthode de renversements de liens est utilisée pour maintenir de façon dynamique une structure de graphe dirigé acyclique orienté vers le ou les détenteurs courants d'un jeton, et assurer que les processus sont tous privilégiés de façon *équitable*[2] au cours de l'exécution. Park et Corson [37] proposent un algorithme appelé TORA (pour "Temporally Oriented Routing Algorithm"), inspiré des algorithmes de renversements de liens de Gafni et Bertsekas, qui résout le problème de l'orientation vers la destination ainsi que la détection d'éventuels partitionnements du graphe de communication. Malpani *et al.* adaptent dans [35] l'algorithme TORA de manière à résoudre le problème de l'Élection de leader. Leur algorithme détecte d'éventuels partitionnements du graphe de communication, résout le problème de l'orientation vers la destination dans chacune des composantes connexes et assure que tous les processus "connaissent" l'identité de la destination (ou leader) de la composante connexe à laquelle ils appartiennent. Enfin, d'autres se sont intéressés au problème de l'allocation de ressource [12, 5, 34]. Dans ce problème, les processus d'un système partagent une ou plusieurs ressources qu'ils ne peuvent utiliser que l'un après l'autre. Une solution doit garantir que tous les processus peuvent utiliser les ressources dont ils ont besoin de façon équitable, et qu'il n'y ait pas de blocage, c'est-à-dire que tous les processus accèdent une infinité de fois à toutes leurs ressources au cours d'une exécution infinie. Comme l'ont démontré Chandy et Misra dans [12], il n'existe pas de solution à ce problème si l'on ne suppose pas l'existence d'une propriété permettant de distinguer les processus parta-

[2]Dans une exécution infinie, chaque processus doit obtenir le jeton une infinité de fois.

geant une ressource donnée. De manière informelle, les solutions proposées font intervenir un graphe dont les sommets sont les processus et tel qu'il existe un lien entre deux sommets si et seulement si les processus associés partagent une ressource. Comme dans le travail de Gafni et Bertsekas, chaque processus met à jour une *hauteur* à valeurs dans un ensemble totalement ordonné. Des directions virtuelles sont affectées aux liens, qui représentent une priorité pour l'accès à la ressource, induisant une structure dirigée sur le graphe. D'autre part, les hauteurs sont définies de telle manière que deux processus distincts ne peuvent pas avoir la même hauteur. Ainsi, le graphe est non seulement dirigé mais aussi acyclique. La méthode de renversements de liens est alors utilisée pour assurer, d'une part, que les graphes dirigés engendrés par l'exécution sont tous acycliques et donc qu'il n'y a pas de blocage et, d'autre part, que les priorités sont transmises équitablement à tous les processus partageant une ressource. Contrairement aux exemples précédents, on ne requiert pas ici la résolution du problème d'orientation vers la destination.

Nous nous proposons, dans le présent chapitre, d'étudier attentivement les algorithmes de Gafni et Bertsekas. Notre but premier était de déterminer s'il existait une autre manière d'implémenter FR et PR. L'approche adoptée par Gafni et Bertsekas, fondée sur l'utilisation de hauteurs deux à deux distinctes à valeurs dans un ensemble totalement ordonné, présente de nombreux avantages. Tout d'abord, elle permet d'unifier de manière élégante les expressions des deux algorithmes FR et PR. Leur convergence est ainsi démontrée en ne considérant que des propriétés abstraites des hauteurs et des fonctions qui les modifient. De plus, en variant la représentation des hauteurs ainsi que celle des fonctions qui les modifient, on peut s'attendre à obtenir d'autres algorithmes. Enfin, la structure totalement ordonnée de l'ensemble des valeurs prises par les hauteurs, combinée avec le fait que les hauteurs sont deux à deux distinctes, garantit que tous les graphes engendrés au cours d'une exécution sont acycliques.

Cependant, plusieurs inconvénients apparaissent lors d'une étude plus approfondie. En premier lieu, la détermination distribuée des hauteurs initiales peut s'avérer problématique. Ensuite, la correction de l'approche dépend du fait que les hauteurs sont non-bornées, ce qui, d'un point de vue pratique, limite fortement l'utilité de l'approche. Enfin, malgré l'apparente flexibilité résultant des différents types de hauteurs pouvant être considérés, la description d'algorithmes dont les exécutions diffèrent de celles des implémentations de FR et PR n'est pas aisée.

Nous présentons une nouvelle formalisation de ces algorithmes, fondée

uniquement sur un étiquetage binaire des *liens*. En utilisant cette approche plus directe, nous définissons un algorithme simple destiné à établir des routes dans des graphes acycliques, et nous déterminons des conditions sur l'étiquetage du graphe d'entrée suffisantes pour assurer la correction de l'algorithme. Les algorithmes *FR* et *PR* de Gafni et Bertsekas peuvent alors être vus comme des cas particuliers de notre algorithme pour des étiquetages initiaux uniformes. De plus, en considérant des étiquetages initiaux non-uniformes, notre algorithme capture plus que simplement *FR* et *PR*. Pour terminer, cette formalisation nous permet de fournir une preuve simple du fait que notre algorithme résout le problème de l'orientation vers la destination. À notre connaissance, c'est la seule preuve directe du fait que l'algorithme abstrait *PR*, qui ne suppose pas l'existence de hauteurs totalement ordonnées, préserve l'acyclicité des graphes engendrés par une exécution.

Curieusement, le seul travail portant sur la complexité des algorithmes de Gafni et Bertsekas est dû à Busch *et al.* [8, 9]. Dans ces travaux, les auteurs ont analysé la *complexité en travail* (le nombre de renversements effectués par chacun des nœuds au cours d'une exécution), la complexité en travail *globale* (la somme des complexités en travail de tous les nœuds) ainsi que la *complexité en temps* (le nombre d'itérations au cours desquelles tous les puits prennent un pas de calcul nécessaires à la convergence). Ils ont donné une formule exacte pour la complexité en travail de l'algorithme avec des paires et des bornes inférieure et supérieure pour celle de l'algorithme avec des triplets (correspondant respectivement à *FR* et *PR*). Si n représente le nombre de nœuds qui initialement ne disposent pas d'un chemin vers la destination, ils ont aussi montré que, pour ces deux algorithmes, la complexité en travail globale est $\Theta(n^2)$. Enfin, ils ont utilisé ces résultats pour déterminer des bornes sur la complexité en temps.

Notre formalisation de l'algorithme général de renversements de liens nous permet d'exprimer le nombre *exact* de renversements effectués par chaque nœud au cours d'une exécution donnée. L'expression dépend uniquement du graphe étiqueté initial, et donne lieu à des formules simples dans les cas particuliers de *FR* et *PR*. Le fait d'avoir une formule exacte facilite la déterminaion des graphes *pire cas* et *meilleur cas*.

La suite du présent chapitre est organisée de la manière suivante. Dans la Section 4.2, nous définissons formellement le problème et passons en revue les précédentes solutions. Dans la Section 4.3, nous présentons notre solution au problème et démontrons sa correction. Nous étudions les complexités en travail et en temps de cette solution dans la Section 4.4.

4.2 Le problème original : Orientation vers la Destination

4.2.1 Notations

Nous rappelons tout d'abord la terminologie usuelle de la théorie des graphes et introduisons quelques notations. Soit G un graphe dirigé connexe contenant un nœud distingué, la *destination*, noté D. L'ensemble des nœuds de G, autres que D est fini et noté V. Le graphe G est dit *D(estination)-orienté* si, pour tout nœud v de V, il existe un chemin dirigé de v vers D. On considère alors le problème $\mathcal{P}_D$ suivant :

$\boldsymbol{\mathcal{P}_D}$: Transformer le graphe dirigé connexe G en un graphe dirigé D-orienté en renversant la direction de certains liens.

Pour chaque nœud v de V, on note N_v l'ensemble des voisins (entrants et sortants) de v dans G. Si v n'a pas de lien sortant, alors v est un *puits* de G. Une *chaîne* dans G est une suite de nœuds $v_0, \cdots, v_k$ telle que, pour tout indice i, $0 \leq i \leq k-1$, soit (v_i, v_{i+1}), soit (v_{i+1}, v_i) est un lien de G. Un *chemin* dans G est une suite de nœuds $v_0, \cdots, v_k$ telle que, pour tout indice i, $0 \leq i \leq k-1$, (v_i, v_{i+1}) est un lien de G. Un *circuit* (resp. un *cycle*) est une chaîne fermée (resp. un chemin fermé), c'est-à-dire une chaîne (resp. un chemin) $v_0, \cdots, v_k$ avec $v_0 = v_k$.

De manière évidente, si G est D-orienté, alors D est le seul puits dans G. Comme l'ont démontré Gafni et Bertsekas dans [23], la réciproque est vraie si G est acyclique :

Lemme 4.2.1 *Soit G un graphe dirigé acyclique contenant un nœud distingué D. Si D est le seul puits de G, alors G est D-orienté.*

Démonstration: Considérons un nœud quelconque v différent de D. Puisque G est connexe, il existe une chaîne reliant v à D dans G. Raisonnons par l'absurde et supposons qu'aucune des chaînes reliant v à D ne soit un chemin dirigé de v vers D. Soit N_v^s l'ensemble des voisins sortants de v. Puisque v n'est pas un puits dans G et que, par hypothèse, il n'existe pas de chemin dirigé de v vers D, on a $N_v^s \neq \emptyset$ et $D \notin N_v^s$. Soit v_1 dans N_v^s. On montre de la même manière que l'ensemble $N_{v_1}^s$ des voisins sortants de v_1 est non-vide et ne contient pas D ni v. Soit à présent $v_2 \in N_{v_1}^s$. Puisque v_2 n'est pas un puits, et qu'il n'existe pas de chemin dirigé de v_2 vers D, on a à nouveau que

$N^s_{v_2}$ est non-vide et ne contient pas D. De plus, puisque G est acyclique, on a aussi que $N^s_{v_2}$ ne contient pas v ni v_1. En répétant le même argument, on montre qu'il existe un chemin dirigé $v_0(= v), v_1, v_2, \cdots$ de longueur infinie tel que, pour tout indice i, $i \leq 0$, $v_{i+1} \notin \{v_0, \cdots, v_i\} \cup \{D\}$, une contradiction puisque G est fini. □

Dans un contexte distribué, le Lemme 4.2.1 laisse entrevoir une stratégie prometteuse pour la résolution de $\mathcal{P}_D$: elle consiste à "combattre" les puits - c'est-à-dire changer les directions des liens incidents aux puits - tout en conservant l'acyclicité. Une telle stratégie nécessite de travailler sous l'hypothèse qu'un processus est associé à chaque nœud du graphe et que, pour chaque nœud v dans V, le processus associé à v est capable (a) de déterminer la direction des liens incidents à v et (b) de changer la direction de chacun d'eux. Dans ce cas, tout processus associé à un puits peut détecter localement que le graphe n'est pas orienté vers la destination, et renverse donc la direction d'un ou plusieurs de ses liens incidents de manière à ne plus être un puits. Il reste alors à vérifier que ce schéma préserve bien l'acylicité et termine.

Dans la suite, nous travaillerons dans le contexte de (a) et (b) et nous identifierons un nœud v et le processus qui lui est associé. Nous considèrerons des *exécutions asynchrones* de notre algorithme de renversements de liens : si un nœud v est un puits à un moment de l'exécution, nous requérons seulement que v effectue finalement un renversement. On note que les directions des liens incidents à un puits restent inchangées jusqu'à ce que le puits en question effectue un renversement.

4.2.2 L'approche de Gafni et Bertsekas

En suivant l'approche décrite ci-dessus, Gafni et Bertsekas [23] ont décrit deux algorithmes distribués, qu'ils ont appelés *full reversal* (FR) et *partial reversal* (PR) :

FR : "A chaque itération, un certain nombre de puits autres que D prennent un pas de calcul. Chaque puits qui prend un pas de calcul renverse les directions de tous ses liens incidents."

PR : "Chaque nœud met à jour une liste de ses liens incidents qui viennent d'être renversés par ses voisins. A chaque itération, un certain nombre de puits autres que D prennent un pas de calcul. Chaque puits qui prend un pas de calcul renverse les directions de ses liens incidents qui

n'apparaissent pas dans sa liste, puis vide la liste. Si la liste est pleine, il renverse les directions de tous ses liens incidents puis vide la liste."

Note:

Il est important de remarquer ici que la description de *PR* donnée par Gafni et Bertsekas pose deux problèmes majeurs. Tout d'abord, on peut penser en la lisant que seul les puits peuvent prendre un pas de calcul. Or le mode de mise à jour des listes tel qu'il est décrit autorise d'autres nœuds que les puits à prendre des pas de calcul. D'autre part, ce même mode de mise à jour des listes suppose implicitement que, dans toute exécution, tous les nœuds du graphe ont connaissance des directions de leurs liens incidents dans le graphe initial. Dans un contexte asynchrone, en effet, si on ne fait pas cette hypothèse, il n'est pas possible pour un puits qui prend son premier pas de calcul de déterminer si ses liens entrants ont précédemment été renversés ou non. Pour éviter le premier point, on peut modifier cette description comme suit :

Chaque nœud met à jour une liste de ses liens incidents qu'il n'a pas renversés. A chaque itération, un certain nombre de puits autres que D prennent un pas de calcul. Chaque puits qui prend un pas de calcul renverse les directions de ses liens incidents qui apparaissent dans sa liste, vide la liste puis la met à jour en y plaçant les liens qu'il n'a pas renversés. Si la liste est vide, il renverse les directions de tous ses liens incidents et laisse sa liste vide.

Cette description impose qu'un nœud ne peut mettre à jour sa liste que lorqu'il est un puits. Cependant, cela ne règle pas le problème du premier pas de calcul pris par un nœud. Cette nouvelle description impose en effet que la liste de chacun des nœuds contienne initialement l'ensemble de ses liens entrants, ce qui revient à supposer que tous les nœuds ont connaissance des directions de leurs liens incidents dans le graphe initial.

Puisque *FR* impose que tout nœud qui prend un pas de calcul renverse tous ses liens incidents, on a trivialement que tous les graphes engendrés par une exécution à partir d'un graphe acyclique sont acycliques. En ce qui concerne *PR*, au contraire, une démonstration de cette propriété apparaît extrêmement compliquée à mener et, à notre connaissance, aucune preuve directe n'existe. En effet, l'argument donné dans [23] est indirect : Gafni et Bertsekas ont d'abord démontré que *FR* et *PR* étaient des cas particuliers

d'une solution générale à un problème fondé sur un ordre, dual de $\mathcal{P}_D$, que nous appelons $\mathcal{P}_h$. L'acyclicité des graphes engendrés découle alors de la propriété d'antisymétrie de tout ordre :

$\boldsymbol{\mathcal{P}_h}$: Soit $\overline{G} = (V, \overline{E})$ un graphe fini connexe (non-dirigé) contenant un nœud spécial D. On affecte à chaque nœud v de V une hauteur $\mathtt{h}_v$ à valeurs dans un ensemble A_v de façon que :

- pour tout $v \in V$, l'ensemble A_v est infini dénombrable et non-borné (il n'a pas d'élément maximal),
- pour tous nœuds distincts v et w de V, on a $A_v \cap A_w = \emptyset$,
- $A = \cup_{v \in V} A_v$ est totalement ordonné par une relation $<$,
- et chaque ensemble A_v est équipé d'une opération d'addition qui en fait un groupe.

En *faisant croître certaines des hauteurs*, trouver des hauteurs pour les nœuds de telle sorte que la hauteur de chaque nœud autre que D n'est pas plus petite que celles de tous ses voisins.

En d'autres termes, leur approche consiste à plonger le graphe dirigé dans un ordre total. Un nœud v met à jour une variable, notée $\mathtt{h}_v$, à valeurs dans un ensemble totalement ordonné $(A, <)$. L'ordre sur les hauteurs détermine les *directions* des liens par la règle suivante : *si* $\mathtt{h}_v > \mathtt{h}_w$*, alors le lien reliant* v *à* w *est dirigé de* v *vers* w. Ainsi, un puits peut être vu comme un minimum local au regard des hauteurs, en ce sens que sa hauteur est plus petite que celle de chacun de ses voisins. Chaque nœud $v \in V$ est supposé accroître sa hauteur, quand celle-ci est inférieure à celle de chacun de ses voisins (minimum local), selon une certaine fonction g_v. Sous l'hypothèse que chaque nœud peut lire les hauteurs de tous ses voisins, cela mène naturellement à un algorithme distribué général $\mathcal{IH}$ (pour *Increasing Heights*).

Gafni et Bertsekas ont supposé que l'ensemble A était non-borné et ont placé une condition sur le comportement asymptotique des fonctions g_v. Ils ont démontré que, sous ces conditions, l'algorithme $\mathcal{IH}$ résolvait $\mathcal{P}_h$, et par conséquent $\mathcal{P}_D$ - puisqu'on peut toujours interpréter la relation entre deux hauteurs comme la direction d'un lien. D'autre part, ils ont démontré que le graphe solution d'une exécution ne dépendait que du graphe initial, et non de l'ordre dans lequel les processus effectuaient leurs pas de calcul.

Si on suppose l'existence d'identifiants uniques pour les nœuds, alors *FR* et *PR* peuvent être vus comme des cas particuliers de l'algorithme $\mathcal{IH}$. En

effet, FR peut être implémenté en représentant la hauteur d'un nœud v par une paire (α_v, i_v), où α_v est un entier et i_v est l'identifiant unique du nœud v. La relation $<$ est l'ordre lexicographique et les fonctions g_v transforment les minima locaux en maxima locaux. Plus précisément, pour tout nœud v, on a :

$$g_v(h_v) = (\max\{\alpha_w : \ w \text{ voisin de } v\} + 1 \ ; \ i_v).$$

De manière similaire, PR peut être implémenté en représentant les hauteurs par des triplets (α_v, β_v, i_v) avec initialement $\alpha_v = 0$ et la fonction g_v est définie comme suit :

$$g_v(h_v) = (\overline{\alpha}_v; \overline{\beta}_v; i_v),$$

où $\overline{\alpha}_v = \min\{\alpha_w : \ w \text{ voisin de } v\} + 1$ et

$$\overline{\beta}_v = \begin{cases} \min\{\beta_w : \ w \text{ voisin de } v \text{ et } \overline{\alpha}_v = \alpha_w\} - 1 & \text{si un tel } w \text{ existe,} \\ \beta_v & \text{sinon.} \end{cases}$$

Comme nous l'avons expliqué dans la Section 4.1, l'approche de Gafni et Bertsekas présente de nombreux avantages. Tout d'abord, elle permet d'unifier FR et PR de manière élégante. Ainsi, la convergence des deux algorithmes est démontrée dans [23] en ne considérant que des propriétés abstraites des hauteurs et des fonctions g_v. De plus, la structure ordonnée des hauteurs, c'est-à-dire l'ordre sur l'ensemble A, garantit trivialement l'acyclicité de la solution. Enfin, en faisant varier les représentations possibles des hauteurs ainsi que les fonctions, on peut s'attendre à décrire d'autres solutions que les seules FR et PR.

Cependant, de nombreux inconvénients sont aussi à prendre en compte. En premier lieu, une initialisation distribuée des hauteurs qui représente les directions du graphe dirigé d'entrée n'est pas évidente. Dans les cas particuliers des implémentations de FR et PR décrites plus haut, cela revient à affecter des identités *distinctes* aux nœuds, et donc à résoudre le problème du *nommage*. Ensuite, la correction de l'approche dépend du fait que les hauteurs sont *a priori* non-bornées : étant donnée une implémentation de $\mathcal{IH}$ avec un certain ensemble A, un processus ne peut déterminer à l'avance une borne supérieure sur les hauteurs qu'il utilisera. Enfin, il apparaît que la flexibilité attendue de l'approche ne fournit en fait pas de solution réellement différente de FR et PR. À notre connaissance, le seul algorithme différent des implémentations de FR et PR a été donné par Busch *et al.* dans [8, 9]. Dans cet algorithme, les hauteurs sont représentées par des triplets, comme dans

l'implémentation de PR, mais les premières composantes ne sont pas supposées égales initialement. Bien que différentes de celles de l'implémentation de PR, les exécutions de cet algorithme s'avèrent en fait être équivalentes à ces dernières après une phase préliminaire tendant à uniformiser les premières composantes des hauteurs. Il apparaît donc que la démarche adoptée par Gafni et Bertsekas [23] utilise un formalisme puissant sans réellement exploiter son expressivité. De plus, elle ne donne pas de nouveaux éclaircissements sur la nature profonde de $\mathcal{P}_D$ ni sur la complexité des algorithmes de graphes FR et PR.

4.3 L'algorithme $\mathcal{LR}$

Dans cette section, nous présentons un algorithme distribué sur des graphes, que nous appelons $\mathcal{LR}$, qui couvre à la fois FR et PR. Afin d'expliquer notre approche basée sur des graphes, étudions plus attentivement les propriétés des listes utilisées dans PR. Au vu de la remarque faite sur la description donnée par Gafni et Bertsekas dans [23], nous considérons la description alternative que nous avons proposée. On peut observer que, pour toute paire de voisins v et w, le lien reliant v et w est dans la liste de w, notée *w.liste*, seulement si w ne l'a pas renversé lors de son dernier pas de calcul, et donc s'il est dirigé de v vers w. Ainsi, on ne peut avoir $v \in w.list$ et $w \in v.list$ simultanément. Par conséquent, pour chaque lien dirigé de v vers w, il existe seulement deux cas de figure possibles : soit $(v, w) \notin w.list$ et $(v, w) \notin v.list$, ou bien $(v, w) \in w.list$ et $(v, w) \notin v.list$. Notre idée est de coder ces deux cas avec des étiquettes 0 et 1 sur le lien de v vers w, respectivement. Nous proposons ainsi un algorithme qui s'exécute sur des graphes dirigés munis d'un étiquetage *binaire* sur les *liens*, contrairement à $\mathcal{IH}$ qui affecte des hauteurs *non-bornées* aux *nœuds*.

Plus généralement, soit $\mu : E \longrightarrow \{0, 1\}$ un étiquetage binaire des liens de G. Considérons le graphe dirigé étiqueté résultant. Soit e un lien de G. Si $\mu(e) = 1$, on dit que e est *marqué*; sinon e est dit *non-marqué*. Dans toute la suite, l'exposant "†" est utilisé pour représenter les graphes dirigés lorsqu'ils sont munis d'un étiquetage binaire des liens.

Pour chaque sommet v, soit In_v et Out_v les ensembles des liens entrants et sortants de v, respectivement. A l'intérieur de In_v, on distingue le sous-ensemble des liens entrants marqués In_v^1 et celui des liens entrants non-marqués In_v^0. L'algorithme $\mathcal{LR}$ peut alors être décrit par les deux règles de

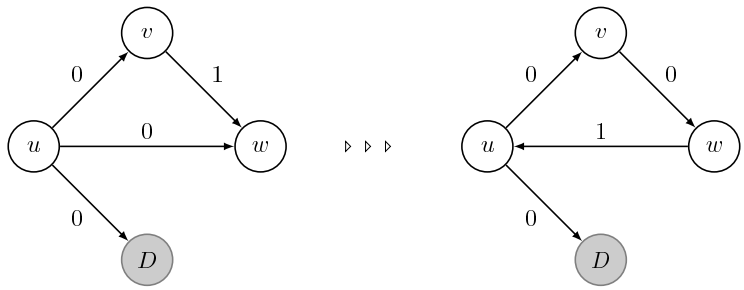

FIG. 4.1 – Un exemple d'exécution de $\mathcal{LR}$ différente de toute exécution de $\mathcal{IH}$.

transition exécutées par tout *puits* v différent de D :

LR1 : Si $In_v^0 \neq \emptyset$, alors v renverse les directions de tous les liens de In_v^0 et les marque ; de plus v démarque tous les liens de In_v^1.

LR2 : Sinon (c'est-à-dire si $In_v^0 = \emptyset$), v renverse les directions de tous ses liens incidents et les étiquettes restent inchangées.

On remarque que, si toutes les étiquettes sont initialement égales à 1, alors les liens restent marqués tout au long de l'exécution, et seule la règle LR2 est exécutée. Ainsi, notre algorithme $\mathcal{LR}$ correspond à FR d'un point de vue opérationnel. D'un autre côté, si toutes les étiquettes sont initialement égales à 0, la discussion précédente concernant les exécutions de PR, combinée avec l'hypothèse que la liste de tout nœud contient initialement tous ses liens entrants, indique que $\mathcal{LR}$ s'exécute identiquement à PR. Comme avec la formalisation de Gafni et Bertsekas [23], notre approche unifie donc FR et PR : dans [23], on obtient différentes implémentations en faisant varier l'ordre total $(A, <)$ et les fonctions g_v, tandis que notre unification consiste seulement à faire varier les étiquetages initiaux.

Remarque : Une autre manière d'unifier FR et PR consiste considérer des graphes dirigés, aux liens e desquels on affecte des étiquettes $\mu(e)$ à valeurs dans $\{0, \ldots, k\}$, $k \geq 0$, et sur lesquels s'exécute l'algorithme dans lequel tout puits exécute finalement la règle de transition atomique suivante :

1. Si $\mu(e) = k$ pour tout lien incident e, alors renverser tous les liens. Sinon, renverser les liens e tels que $\mu(e) = 0$.
2. Pour tout lien e, $\mu(e) := \mu(e) + 1 \mod (k+1)$.

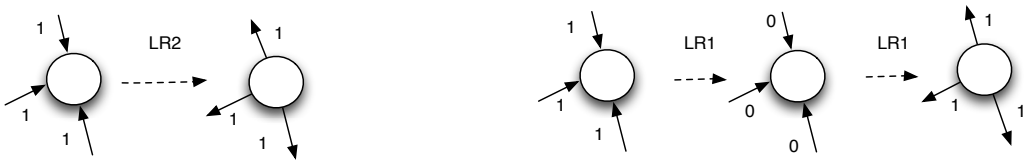

FIG. 4.2 – Une exécution de LR2 est équivalente à deux exécutions consécutives de LR1.

L'algorithme FR peut ainsi être implémenté par cet algorithme exécuté sur des graphes muni d'un étiquetage unaire $\mu : E \longrightarrow \{0\}$. En effet, tous les liens d'un puits ont pour étiquette 0. Lorsque ce puits prend un pas de calcul, ils les renverse tous et leur affecte l'étiquette $1 \mod (1) = 0$. On retrouve donc LR2 dans ce cas. De la même manière, PR peut être implémenté par cet algorithme pour $k = 1$, exécuté sur des graphes dont tous les liens ont l'étiquette 0 initialement. En effet, notre règle est alors équivalente à (LR1 ;LR2).

On utilisera dans la suite la première approche, qui consiste à considérer des graphes munis d'un étiquetage binaire et à faire varier les étiquetages initiaux. Il est intéressant de remarquer que cette approche permet aussi d'exprimer plus d'implémentations que $\mathcal{IH}$, comme le montre l'exemple donné en Figure 4.1. L'algorithme $\mathcal{LR}$ converge vers un graphe D-orienté après un unique renversement effectué par le nœud w. Il est évident que cette exécution ne peut correspondre à aucune implémentation de $\mathcal{IH}$ puisque le graphe D-orienté final contient un cycle. Ainsi, malgré l'utilisation d'une structure combinatoire plus complexe et de règles de transition plus élaborées, il apparaît que l'algorithme $\mathcal{IH}$ n'est pas plus expressif que $\mathcal{LR}$.

Si on étudie attentivement les deux règles LR1 et LR2, on s'aperçoit que toute exécution de LR2 est en fait équivalente à l'exécution atomique de deux transitions LR1 consécutives, comme le montre la Figure 4.2. Si on prend le parti de casser l'atomicité de LR2, il est donc possible de donner une autre description de notre algorithme, qui consiste en une unique règle de transition T exécutée par les puits :

T : Renverser les directions de tous les liens de In_v^0 et les marquer ; démarquer tous les liens de In_v^1.

Bien qu'identique à LR1, nous préférons la nommer T car, comme le montre la Figure 4.2, certaines exécutions de cette règle ne donnent pas lieu au renversement effectif d'un ou plusieurs liens. On peut d'ailleurs penser que c'est pour cette dernière raison que Gafni et Bertsekas ont décrit *FR* et *PR* de la sorte : pour assurer que tout pas de calcul pris par un puits modifie la topologie du graphe.

4.3.1 Convergence et insensibilité au délai

Nous ne considérons ici que des exécutions de $\mathcal{LR}$ dans lesquelles seule la règle de transition T est exécutée. Nous prouvons que $\mathcal{LR}$ converge, c'est-à-dire que toute exécution à partir d'un graphe dirigé étiqueté connexe est finie. D'autre part, comme nous l'avons précisé plus haut, nous considérons des exécutions asynchrones de cet algorithme, dans lesquelles un puits est simplement contraint d'effectuer finalement un pas de calcul. Ainsi, à partir d'un graphe dirigé étiqueté $G^{\dagger}$, il existe plusieurs exécutions possibles de $\mathcal{LR}$ qui dépendent de l'ordonnanceur. Nous montrerons aussi que l'algorithme est *insensible au délai*, en ce sens que le graphe final d'une exécution - ce graphe existe puisque l'exécution est finie - ne dépend que du graphe initial et non de l'ordonnanceur. Il faut remarquer que, par définition de la règle T régissant les exécutions de $\mathcal{LR}$, ce graphe final ne comporte aucun puits, excepté D.

Théorème 4.3.1 *Soit $G_0^{\dagger}$ un graphe dirigé dont les liens sont étiquetés binairement. Dans toute exécution de $\mathcal{LR}$ à partir de $G_0^{\dagger}$, chaque nœud prend un nombre fini de pas de calcul.*

Démonstration: Considérons une exécution de $\mathcal{LR}$, et un nœud v différent de D. Soit $T_i(v)$ le i-ième pas de calcul du nœud v dans cette exécution. La preuve est décomposée en les deux lemmes suivants :

Lemme 4.3.2 *Si v est un voisin de D, alors v effectue au plus deux pas de calcul.*

Démonstration: La règle T assure qu'après au plus deux pas de calcul effectués par v, le lien entre v et D est orienté de v vers D. Puisque D ne prend aucun pas de calcul, v n'est plus jamais un puits, et donc ne prend plus de pas de calcul. □

Lemme 4.3.3 *Pour tout nœud v différent de D, tout nœud $w \neq D$ voisin de v prend au moins un pas de calcul entre $T_i(v)$ et $T_{i+2}(v)$.*

Démonstration: Soit $v \neq D$ et $w \neq D$ un voisin de v. Il y a deux cas à considérer selon l'étiquette du lien e reliant v et w juste avant $T_i(v)$.

1. Si e est marqué, alors v le démarque sans le renverser à $T_i(v)$, puis le renverse et le marque à $T_{i+1}(v)$. Le nœud v doit attendre d'être de nouveau un puits pour prendre un pas de calcul. En particulier, il doit attendre que e soit renversé. Comme les nœuds ne renversent que leurs liens incidents, w doit prendre au moins un pas de calcul avant $T_{i+2}(v)$.

2. Si e est non-marqué, alors v le renverse et le marque à $T_i(v)$. Le même argument que celui qui est utilisé dans le cas précédent montre que w doit nécessairement prendre au moins un pas de calcul entre $T_i(v)$ et $T_{i+1}(v)$.

□

Nous raisonnons par contradiction et supposons qu'il existe un nœud v qui prend une infinité de pas de calcul au cours d'une exécution de $\mathcal{LR}$. Puisque le graphe d'entrée est connexe, il existe une chaîne reliant v à D. Le Lemme 4.3.3 implique alors que tous les nœuds de cette chaîne prennent une infinité de pas de calcul, ce qui contredit le Lemme 4.3.2 □

En ce qui concerne l'insensibilité au délai, le résultat est une conséquence directe du lemme de commutativité suivant :

Lemme 4.3.4 *Soient v et w deux puits distincts de $G^\dagger$. Soit $G^\dagger_v$ et $G^\dagger_w$ les graphes obtenus, à partir de $G^\dagger$, par l'exécution d'un pas de calcul par v et par w, respectivement.*

Alors v est un puits dans $G^\dagger_w$ et w est un puits dans $G^\dagger_v$. De plus, les graphes $G^\dagger_{w.v}$ et $G^\dagger_{v.w}$ obtenus à partir de $G^\dagger_w$ par l'exécution d'un pas de calcul par v et à partir de $G^\dagger_v$ par l'exécution d'un pas de calcul par w, respectivement, sont identiques.

Démonstration: Puisque v et w sont des puits distincts de $G^\dagger$, ils ne sont pas voisins. Puisque, de plus, un puits ne peut modifier que les directions de ses liens incidents, on en déduit que v est un puits dans $G^\dagger_w$ et que w est un puits dans $G^\dagger_v$.

D'autre part, pour les mêmes raisons, les étiquettes et les directions des liens incidents à v sont identiques dans $G^\dagger$ et dans $G^\dagger_w$. Par conséquent, les étiquettes et les directions des liens incidents à v sont identiques dans $G^\dagger_v$ et dans $G^\dagger_{w.v}$. Enfin, puisqu'un pas de calcul de w n'affecte que ses liens incidents, on en conclut que les directions et les étiquettes des liens incidents à v sont identiques dans $G^\dagger_{w.v}$ et dans $G^\dagger_{v.w}$. On montre de la même manière que les directions et les étiquettes des liens incidents à w sont identiques dans $G^\dagger_{w.v}$ et dans $G^\dagger_{v.w}$, ce qui nous permet de conclure que les graphes $G^\dagger_{w.v}$ et $G^\dagger_{v.w}$ sont identiques. □

On peut donc énoncer le théorème suivant :

Théorème 4.3.5 *Soit $G^\dagger_0$ un graphe dirigé dont les liens sont étiquetés binairement. Les graphes finaux de toutes les exécutions de $\mathcal{LR}$ à partir de $G^\dagger_0$ ne dépendent que de $G^\dagger_0$ et sont, par conséquent, identiques.*

4.3.2 La question de l'acyclicité

Nous venons de montrer que toute exécution de $\mathcal{LR}$ à partir d'un graphe dirigé connexe fini, acyclique ou non, converge, c'est-à-dire produit en un nombre fini d'itérations un graphe ne contenant pas d'autre puits que D. Cependant, "combattre les puits" ne représente qu'une partie de la solution requise. Il reste donc à étudier dans quelle mesure l'acyclicité de la solution peut être garantie par $\mathcal{LR}$. Pour cela, notre stratégie consiste à réduire certaines propriétés de l'algorithme à des invariants du graphe, ou plus précisément à certains de ses sous-graphes comme les circuits.

Pour commencer, nous introduisons quelques notations supplémentaires. Puisqu'aucune confusion ne peut survenir, on note $v.G^\dagger$ le graphe obtenu après que le nœud v (et seulement v) a pris un pas de calcul de $\mathcal{LR}$. On note que $v.G^\dagger$ a le même graphe non-dirigé sous-jacent (appelé *support*) que $G^\dagger$. Pour tout circuit $C^\dagger$ de $G^\dagger$, on note $v.C^\dagger$ le circuit correspondant de $v.G^\dagger$ après le pas de calcul de v.

Pour chaque circuit $C^\dagger$, on fixe une orientation arbitraire de son support - qui en fait un cycle - et on considère les nombres de liens marqués de $C^\dagger$ qui sont orientés selon cette orientation, et contre elle. Ces deux nombres sont notés $l^1(C^\dagger)$ et $r^1(C^\dagger)$, respectivement. Soit $s^0(C^\dagger)$ le nombre de nœuds de $C^\dagger$ qui ont deux liens entrants distincts non-marqués, *par rapport à* $C^\dagger$. On définit alors deux mesures λ et ρ :

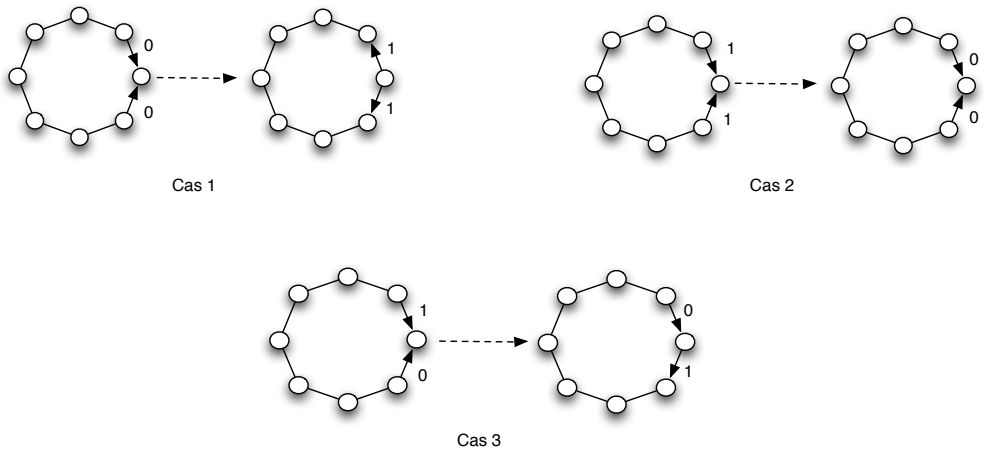

FIG. 4.3 – Influence d'un pas de calcul sur un circuit.

$$\lambda(C^\dagger) = l^1(C^\dagger) + s^0(C^\dagger) \quad \text{et} \quad \rho(C^\dagger) = r^1(C^\dagger) + s^0(C^\dagger).$$

Proposition 4.3.6 *Soit $C^\dagger$ un circuit de $G^\dagger$. Pour tout nœud v, on a*

$$\lambda(v.C^\dagger) = \lambda(C^\dagger) \quad \textit{et} \quad \rho(v.C^\dagger) = \rho(C^\dagger).$$

Démonstration: Si v n'est pas un nœud de $C^\dagger$, alors le pas de calcul effectué par v ne modifie pas $C^\dagger$, c'est-à-dire $v.C^\dagger = C^\dagger$, et le résultat suit immédiatement. Sinon, v est un puits de $C^\dagger$ et il y a trois cas à considérer, illustrés par la Figure 4.3, qui dépendent des étiquettes des liens de $C^\dagger$ incidents à v.

1. Aucun des deux liens n'est marqué. Alors v les renverse et les marque tous les deux. Ainsi

   $$l^1(v.C^\dagger) = l^1(C^\dagger) + 1, \quad r^1(v.C^\dagger) = r^1(C^\dagger) + 1, \quad s^0(v.C^\dagger) = s^0(C^\dagger) - 1.$$

2. Les deux liens sont marqués. Alors v les démarque tous les deux, les directions restant inchangées. Ainsi

   $$l^1(v.C^\dagger) = l^1(C^\dagger) - 1, \quad r^1(v.C^\dagger) = r^1(C^\dagger) - 1, \quad s^0(v.C^\dagger) = s^0(C^\dagger) + 1.$$

3. Un seul des deux liens est marqué. Le nœud v démarque donc celui-ci alors qu'il renverse et marque l'autre. On a donc

$$l^1(v.C^\dagger) = l^1(C^\dagger), \quad r^1(v.C^\dagger) = r^1(C^\dagger), \quad s^0(v.C^\dagger) = s^0(C^\dagger).$$

□

Proposition 4.3.7 *Si $C^\dagger$ est un circuit tel que*

$$\lambda(C^\dagger) \cdot \rho(C^\dagger) > 0, \tag{4.1}$$

alors $C^\dagger$ n'est pas un cycle.

Démonstration: On note que si $C^\dagger$ est un cycle, alors $s^0(C^\dagger) = 0$. De plus, $l^1(C^\dagger) = 0$ ou $r^1(C^\dagger) = 0$. Il s'ensuit donc que la condition (4.1) est suffisante pour garantir que $C^\dagger$ n'est pas un cycle. □

A partir des Propositions 4.3.6 et 4.3.7, on déduit immédiatement le résultat d'acyclicité suivant :

Théorème 4.3.8 *Si le graphe d'entrée G_0 est équipé d'un étiquetage initial tel que tout circuit du graphe dirigé etiqueté correspondant $G_0^\dagger$ satisfait la condition (4.1), alors tous les graphes dirigés engendrés par toute exécution de $\mathcal{LR}$ sont acycliques.*

En combinant les Théorèmes 4.3.1, 4.3.5 et 4.3.8, on obtient le corollaire suivant :

Corollaire 4.3.9 *Soit G_0 un graphe dirigé connexe. Si G_0 est muni d'un étiquetage initial tel que tout circuit du graphe dirigé étiqueté correspondant $G_0^\dagger$ satisfait (4.1), alors $\mathcal{LR}$ résout $\mathcal{P}_D$ pour G_0.*

Si on considère un graphe acyclique G, la politique d'étiquetage, qui consiste à ce que, pour chaque nœud v de G, tous les liens entrants de v soient étiquetés de manière uniforme, garantit que tout circuit $C^\dagger$ du graphe étiqueté induit $G^\dagger$ satisfait la condition (4.1). Pour nous en convaincre, supposons qu'il existe un circuit C dans G. Puisque G est acyclique, il existe un nœud v de C qui a deux liens entrants par rapport à C. Si on affecte à ces deux liens l'étiquette 0, on a $s^0(C^\dagger) > 0$ et donc $\lambda(C^\dagger) \cdot \rho(C^\dagger) > 0$.

Si, au contraire, on affecte à ces deux liens l'étiquette 1, on a $l^1(C^\dagger) > 0$ et $r^1(C^\dagger) > 0$, et ainsi $\lambda(C^\dagger) \cdot \rho(C^\dagger) > 0$.

Considérons la description de $\mathcal{LR}$ faisant intervenir les règles LR1 et LR2. On rappelle que toute exécution de cet algorithme à partir d'un graphe $G^\dagger$ dont tous les liens sont marqués (cet étiquetage est en accord avec la politique que nous venons d'introduire) est identique à une exécution de *FR* à partir du support dirigé non-étiqueté de $G^\dagger$. De la même manière, les exécutions de $\mathcal{LR}$ à partir de graphes dont tous les liens sont non-marqués (étiquetage lui aussi conforme à notre politique) sont identiques aux exécutions de *PR*. Cette dernière remarque fournit donc une preuve directe du fait qu'une exécution de *PR* à partir d'un graphe acyclique n'engendre que des graphes acycliques.

4.4 Complexité

Nous analysons dans cette section la complexité de l'algorithme $\mathcal{LR}$. Nous nous attacherons tout d'abord à déterminer, pour toute exécution de $\mathcal{LR}$ et pour tout nœud du graphe initial, le nombre de *pas de calcul* pris par ce nœud dans cette exécution, puis le nombre de *renversements* qu'il effectue. Dans un deuxième temps, nous nous intéresserons au nombre de rounds nécessaires à la convergence d'une exécution *complète* de $\mathcal{LR}$ - dans laquelle, à chaque round, tous les puits prennent un pas de calcul - à partir d'un graphe dont tous les liens sont marqués. Nous donnerons une formule exacte pour le cas des chaînes et en déduirons une borne inférieure pour le cas général. Nous appliquerons ensuite ces résultats aux cas particuliers de *FR* et *PR*. Nous terminerons par une comparaison de nos résultats avec ceux de Busch *et al.* [8, 9].

4.4.1 Complexité en travail

On définit la *complexité en travail d'un nœud* dans une exécution comme le nombre de pas de calcul effectués par ce nœud, et la *complexité en travail globale* d'une exécution comme la somme des complexités en travail de tous les nœuds. Par les résultats montrés dans la section précédente, il apparaît que la complexité en travail d'un nœud donné dans une exécution ne dépend que du graphe initial, et donc qu'il en est de même pour la complexité en travail globale de cette exécution,.

Nous étendons tout d'abord les notions définies pour les circuits dans la

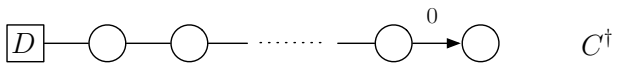

FIG. 4.4 – $Res(C^\dagger) = 1$.

sous-section 4.3.2 aux chaînes. Puisque le graphe d'entrée est connexe et que $\mathcal{LR}$ préserve la connexité, il existe, pour tout nœud v, une chaîne reliant v et D avec un support stable durant toute exécution de $\mathcal{LR}$. L'énoncé du problème d'orientation vers la destination induit naturellement une orientation d'une telle chaîne de v vers D ; on parlera de *D-chaîne depuis v*. Si chaque lien d'une D-chaîne est dirigé vers D, alors la D-chaîne est appelée un D-chemin. On notera $\mathcal{C}(v, G^\dagger)$ l'ensemble des D-chaînes depuis v avec les directions et les étiquettes héritées de $G^\dagger$.

Pour toute D-chaîne $C^\dagger$, nous définissons le *résidu* de $C^\dagger$, noté $Res(C^\dagger)$, comme étant égal à 1 si le premier lien (selon l'orientation naturelle vers D) de $C^\dagger$ est non-marqué et dirigé contre l'orientation (cf. Figure 4.4), ou à 0 sinon. Comme pour les circuits, nous considérons les deux quantités $\lambda(C^\dagger) = l^1(C^\dagger) + s^0(C^\dagger)$ et $\rho(C^\dagger) = r^1(C^\dagger) + s^0(C^\dagger)$, et nous définissons deux quantités supplémentaires :

$$\delta(C^\dagger) = \lambda(C^\dagger) - \rho(C^\dagger) \quad \text{et} \quad \gamma(C^\dagger) = \lambda(C^\dagger) + \rho(C^\dagger) + Res(C^\dagger).$$

A partir de ces définitions, nous déduisons trivialement la proposition suivante :

Proposition 4.4.1 *Pour toute D-chaîne $C^\dagger$, on a $\delta(C^\dagger) \leq \gamma(C^\dagger)$. De plus, si $C^\dagger$ est un D-chemin, alors $\delta(C^\dagger) = \gamma(C^\dagger)$.*

qui nous conduit naturellement à considérer la quantité positive :

$$\omega(C^\dagger) = \gamma(C^\dagger) - \delta(C^\dagger) = 2\ \rho(C^\dagger) + Res(C^\dagger).$$

Nous étudions à présent l'évolution respective de $\delta(C^\dagger)$ et $\gamma(C^\dagger)$ au cours d'une exécution de $\mathcal{LR}$.

Proposition 4.4.2 *Pour tous nœuds v et w et toute D-chaîne $C^\dagger$ depuis w, on a $\gamma(v.C^\dagger) = \gamma(C^\dagger)$.*

Démonstration: Puisque v prend un pas de calcul, v est un puits. On distingue trois cas :

1. Le nœud v n'appartient pas à $C^\dagger$. Puisque le pas de calcul de v n'affecte que ses liens incidents, on a $v.C^\dagger = C^\dagger$. Par conséquent $\gamma(v.C^\dagger) = \gamma(C^\dagger)$.

2. Le nœud v appartient à $C^\dagger$ et $v \neq w$. La même preuve que celle de la Proposition 4.3.6 montre que, dans ce cas, $\lambda(v.C^\dagger) = \lambda(C^\dagger)$ et $\rho(v.C^\dagger) = \rho(C^\dagger)$. Il reste à démontrer que $Res(v.C^\dagger) = Res(C^\dagger)$. Si v n'est pas un voisin de w, alors le résultat suit directement du fait que, dans une exécution de $\mathcal{LR}$, un nœud ne peut changer les directions que de ses liens incidents. Sinon, puisque v est un puits, le lien e reliant w à v est dirigé vers v, et donc $Res(C^\dagger) = 0$. De plus, selon l'étiquette de e dans $C^\dagger$, le lien e est, dans $v.C^\dagger$, soit dirigé vers v et non-marqué, soit dirigé vers w et marqué. Dans les deux cas, on a $Res(v.C^\dagger) = 0$ et donc $Res(v.C^\dagger) = Res(C^\dagger)$, comme souhaité.

3. Le nœud v appartient à $C^\dagger$ et $v = w$. Soit e le premier lien de $C^\dagger$. Il y a deux cas à considérer, selon l'étiquette de e dans $C^\dagger$:

   (a) Si e est marqué dans $C^\dagger$, alors v le démarque sans le renverser. Dans ce cas, $Res(v.C^\dagger) = Res(C^\dagger) + 1$. De plus, on a :

   $$l^1(v.C^\dagger) = l^1(C^\dagger), \quad r^1(v.C^\dagger) = r^1(C^\dagger) - 1, \quad s^0(v.C^\dagger) = s^0(C^\dagger).$$

   (b) Si e est non-marqué dans $C^\dagger$, alors v le renverse et le marque. Il s'ensuit que

   $$Res(v.C^\dagger) = Res(C^\dagger) - 1, \;\; l^1(v.C^\dagger) = l^1(C^\dagger) + 1, \;\; r^1(v.C^\dagger) = r^1(C^\dagger).$$

   Puisque e est marqué dans $v.C^\dagger$, son renversement ne modifie pas la quantité s^0, c'est-à-dire $s^0(v.C^\dagger) = s^0(C^\dagger)$.

Dans les deux cas, nous concluons $\gamma(v.C^\dagger) = \gamma(C^\dagger)$.

□

Ainsi $\gamma(C^\dagger)$ est invariant dans toute exécution de $\mathcal{LR}$. Étudions maintenant le comportement de $\delta(C^\dagger)$.

Proposition 4.4.3 *Pour tous nœuds v et w et toute D-chaîne $C^\dagger$ depuis w, on a $\delta(v.C^\dagger) = \delta(C^\dagger)$ si $v \neq w$, et $\delta(v.C^\dagger) = \delta(C^\dagger) + 1$ sinon.*

Démonstration: On distingue trois cas :

1. Si v n'appartient pas à $C^\dagger$, ce qui implique $v \neq w$, alors $v.C^\dagger = C^\dagger$. Par conséquent, on a bien $\delta(v.C^\dagger) = \delta(C^\dagger)$.

2. Si v appartient à $C^\dagger$ et $v \neq w$, alors les Propositions 4.3.6 et 4.4.2 assurent $\lambda(v.C^\dagger) = \lambda(C^\dagger)$ et $\rho(v.C^\dagger) = \rho(C^\dagger)$. Ainsi, $\delta(v.C^\dagger) = \delta(C^\dagger)$.

3. Si $v = w$, on distingue deux cas selon l'étiquette du lien e de $C^\dagger$ incident à v :

   (a) Si e est marqué, la Proposition 4.4.2 assure

   $$l^1(v.C^\dagger) = l^1(C^\dagger), \quad r^1(v.C^\dagger) = r^1(C^\dagger) - 1,$$

   ce qui implique $\delta(v.C^\dagger) = \delta(C^\dagger) + 1$.

   (b) Si e est non-marqué, alors par cette même proposition on a

   $$l^1(v.C^\dagger) = l^1(C^\dagger) + 1, \quad r^1(v.C^\dagger) = r^1(C^\dagger),$$

   ce qui implique $\delta(v.C^\dagger) = \delta(C^\dagger) + 1$.

□

On remarque ici que, pour tout nœud w et pour toute D-chaîne $C^\dagger$ depuis w, la valeur de $\delta(C^\dagger)$ ne peut être modifiée que lors d'un pas de calcul effectué par w. On en déduit que cette évolution de $\delta(C^\dagger)$ ne dépend pas du nombre de puits différents de w qui effectuent un pas de calcul simultanément. La Proposition 4.4.3 peut ainsi être étendue au cas où plusieurs puits effectuent des pas de calcul simultanément. On pourra par conséquent être amenés à considérer dans la suite la notation $S.G^\dagger$, où S est un sous-ensemble de puits de $G^\dagger$.

Proposition 4.4.4 *Pour tout nœud w, pour toute D-chaîne $C^\dagger$ depuis w, et pour tout ensemble de puits S, on a $\delta(S.C^\dagger) = \delta(C^\dagger)$ si $w \notin S$, et $\delta(v.C^\dagger) = \delta(C^\dagger) + 1$ sinon.*

Nous sommes à présent en mesure de déterminer le nombre exact de pas de calcul effectués par chacun des nœuds au cours de *toute* exécution *maximale* de $\mathcal{LR}$ - qui a atteint un graphe D-orienté - à partir d'un graphe dirigé étiqueté dont les circuits satisfont tous la condition (4.1). Pour ce faire, nous avons besoin d'une nouvelle définition. Pour chaque nœud v, nous considérons l'ensemble des D-chaînes depuis v et définissons la quantité :

$$\omega(v, G^\dagger) = \min_{C^\dagger \in \mathcal{C}(v, G^\dagger)} \big(\omega(C^\dagger)\big).$$

Comme conséquence immédiate des Propositions 4.4.2 et 4.4.3, nous déduisons le lemme suivant :

Lemme 4.4.5 *Les D-chaînes qui réalisent le minimum des $\omega(C^\dagger)$ sont les mêmes tout au long de l'exécution.*

Soit $G_0^\dagger$ un graphe dirigé dont tous les circuits vérifient la condition (4.1). On fixe une exécution arbitraire de l'algorithme $\mathcal{LR}$ à partir de $G_0^\dagger$ et on note

$$G_0^\dagger, \ldots, G_i^\dagger, \ldots$$

la suite de graphes dirigés étiquetés connexes engendrés au cours de l'exécution. Par le Corollaire 4.3.9, cette suite est finie et, puisque $G_0^\dagger$ est acyclique, le dernier graphe, noté $G_k^\dagger$, est D-orienté.

Théorème 4.4.6 *Dans toute exécution de $\mathcal{LR}$ à partir de $G_0^\dagger$ dont tous les circuits satisfont la condition (4.1), pour tout nœud v de $G_0^\dagger$ différent de D, la complexité en travail de v est égale à $\omega(v, G_0^\dagger)$.*

Démonstration: Soit $v \neq D$ un nœud de $G_0^\dagger$. Considérons la suite finie d'entiers positifs

$$\omega(v, G_0^\dagger), \omega(v, G_1^\dagger), \ldots, \omega(v, G_k^\dagger),$$

et la sous-suite

$$\omega(v, G_0^\dagger), \omega(v, G_{i_1}^\dagger), \ldots, \omega(v, G_{i_{k_v}}^\dagger)$$

obtenue en ne considérant que les indices $i \geq 1$ tels que v prend un pas de calcul dans $G_{i-1}^\dagger$.

Par le Lemme 4.4.5 et les Propositions 4.4.2 et 4.4.4, il s'ensuit que la première est décroissante, et que la seconde est strictement décroissante.

Plus précisément, on déduit des Propositions 4.4.2 et 4.4.4 que cette dernière décroît régulièrement de 1 à chaque étape.

Puisque tous les circuits de $G_0^\dagger$ satisfont la condition (4.1), le Corollaire 4.3.9 implique que $G_k^\dagger$ est D-orienté. La Proposition 4.4.1 assure alors que $\omega(v, G_{i_{k_v}}^\dagger) = \omega(v, G_k^\dagger) = 0$. On en conclut finalement que la sous-suite

$$\omega(v, G_0^\dagger), \omega(v, G_{i_1}^\dagger), \ldots, \omega(v, G_{i_{k_v}}^\dagger)$$

décroît par pas de 1 depuis $\omega(v, G_0^\dagger)$ jusqu'à 0, ce qui nous permet de déduire la complexité en travail de v dans cette exécution. □

Pas de calcul et renversements

On se propose d'utiliser le résultat du Théorème 4.4.6 pour déterminer le nombre de pas de calcul effectués par chacun des nœuds desquels résulte le renversement d'un lien dans une exécution de $\mathcal{LR}$ où seule la règle T est exécutée. On note que l'exécution de la règle T ne modifie pas la direction des liens incidents à un puits dans un seul cas, quand tous les liens incidents à ce puits sont marqués. Cette observation nous conduit à partitionner l'ensemble des nœuds de V en trois sous-ensembles : $\mathcal{S}^0(G^\dagger)$ est l'ensemble des puits dont tous les liens entrants sont non-marqués, $\mathcal{N}(G^\dagger)$ est l'ensemble des nœuds n'ayant aucun lien entrant non-marqué et $\mathcal{O}(G^\dagger)$ regroupe tous les autres nœuds. La proposition suivante montre comment évolue ce partitionnement au cours d'une exécution de $\mathcal{LR}$.

Proposition 4.4.7 *Si v est un puits de $G^\dagger$ alors*

$$\mathcal{S}^0(v.G^\dagger) = \mathcal{S}^0(G^\dagger) \setminus \{v\} \cup \big(\mathcal{N}(G^\dagger) \cap \{v\}\big),$$
$$\mathcal{N}(v.G^\dagger) = \mathcal{N}(G^\dagger) \setminus \{v\} \cup \big(\mathcal{S}^0(G^\dagger) \cap \{v\}\big),$$
$$\mathcal{O}(v.G^\dagger) = \mathcal{O}(G^\dagger).$$

Démonstration: Soit v un puits de $G^\dagger$, qui prend un pas de calcul. Soit, de plus, w un voisin de v et z un nœud différent de v et n'appartenant pas à son voisinage. La Figure 4.5 montre comment v migre d'un ensemble à l'autre lors d'un pas de calcul.

1. Si v appartient à $\mathcal{O}(G^\dagger)$, alors v a au moins un lien non-marqué et au moins un lien marqué dans $G^\dagger$. La règle T assure que v a au moins un

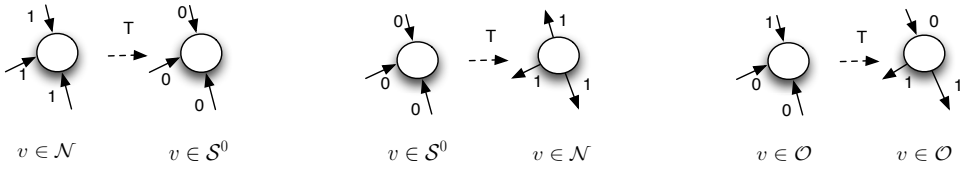

FIG. 4.5 – Pas de calcul et renversements.

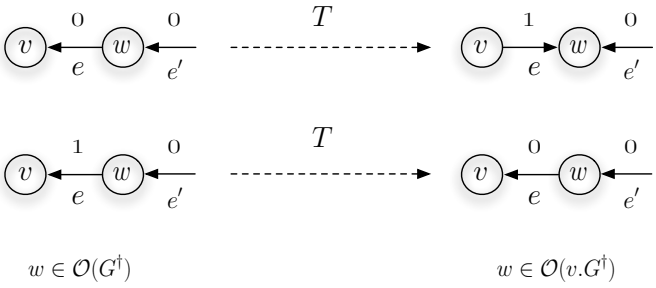

FIG. 4.6 –

lien entrant non-marqué et un lien sortant dans $v.G^\dagger$. On en conclut que v appartient à $\mathcal{O}(v.G^\dagger)$.

2. Si v appartient à $\mathcal{N}(G^\dagger)$, alors tous les liens incidents à v sont marqués dans $G^\dagger$. Il démarque donc tous ses liens entrants sans les renverser. Ainsi, dans $v.G^\dagger$, tous les liens de v sont entrants et non-marqués. Par conséquent $v \in \mathcal{S}^0(v.G^\dagger)$.

3. Si v appartient à $\mathcal{S}^0(G^\dagger)$, alors tous les liens incidents à v sont marqués dans $G^\dagger$. La règle T impose que v les renverse et les marque tous. Ainsi, v n'a aucun lien entrant non-marqué dans $v.G^\dagger$ et donc $v \in \mathcal{N}(v.G^\dagger)$.

Considérons à présent w. Soit e le lien reliant v et w. Puisque v est un puits dans $G^\dagger$, w n'appartient pas à $\mathcal{S}^0(G^\dagger)$. On distingue deux cas :

1. S $w \in \mathcal{N}(G^\dagger)$ alors w n'a aucun lien entrant non-marqué dans $G^\dagger$. Dans $v.G^\dagger$, le lien e est soit dirigé de v vers w et marqué, soit dirigé de w

vers v et non-marqué. Puisque les autres liens incidents à w ne sont pas modifiés par le pas de calcul de v, on en conclut que w n'a aucun lien entrant non-marqué dans $v.G^\dagger$, et donc que $w \in \mathcal{N}(v.G^\dagger)$.

2. Si $w \in \mathcal{O}(G^\dagger)$, alors, puisque e est dirigé vers v dans $G^\dagger$, le nœud w a un lien e' entrant non-marqué dans $G^\dagger$. Comme dans le cas précédent, dans $v.G^\dagger$, le lien e est, soit dirigé de v vers w et marqué, soit dirigé de w vers v et non-marqué. Puisque e' n'est pas modifié par le pas de calcul de v, on en déduit que dans $v.G^\dagger$, le nœud w a, soit un lien entrant marqué et un lien entrant non-marqué, soit un lien entrant non-marqué et un lien sortant. Dans les deux cas, illustrés par la Figure 4.6, on a $w \in \mathcal{O}(v.G^\dagger)$.

Pour terminer, on remarque que les liens incidents à z ne sont pas modifiés par le pas de calcul de v, et donc que z appartient au même ensemble dans $G^\dagger$ et dans $v.G^\dagger$. □

On voit ainsi que si $v \in \mathcal{O}$, alors un de ses pas de calcul ne modifie pas les ensembles $\mathcal{S}^0$, $\mathcal{N}$ et $\mathcal{O}$ et que, si ce n'est pas le cas, v migre de $\mathcal{S}^0$ vers $\mathcal{N}$ ou de $\mathcal{N}$ vers $\mathcal{S}^0$ à chaque pas de calcul qu'il effectue. Enfin, comme on l'a remarqué plus haut, les seuls pas de calcul qui n'ont pas pour résultat le renversement d'un ou plusieurs liens sont ceux qui sont effectués pas des nœuds dans $\mathcal{N}$. Ces deux observations nous indiquent donc que, dans toute exécution de $\mathcal{LR}$ à partir d'un graphe $G_0^\dagger$, pour tout nœud v appartenant à $\mathcal{S}^0(G_0^\dagger)$ ou $\mathcal{N}(G_0^\dagger)$, seul un pas de calcul sur deux effectué par v dans cette exécution a pour effet de renverser les liens incidents à v.

Comme dans la sous-section précédente, on remarque que l'énoncé de la Proposition 4.4.7 peut être étendu au cas ou plusieurs nœuds effectuent un renversement simultanément. D'autre part, le résultat enoncé par le Lemme 4.4.5 est toujours valable. Par conséquent, on peut réénoncer le Théorème 4.4.6 de la manière suivante :

Théorème 4.4.8 *Dans toute exécution de $\mathcal{LR}$ à partir de $G_0^\dagger$ dont tous les circuits satisfont la condition (4.1), le nombre de* **renversements** *effectués par tout nœud v est $\omega(v, G_0^\dagger)/2 + 1/2$, $\omega(v, G_0^\dagger)/2$ ou $\omega(v, G_0^\dagger)$ selon que v appartient à $\mathcal{S}(G_0^\dagger)$, à $\mathcal{N}(v.G_0^\dagger)$ ou à $\mathcal{O}(G_0^\dagger)$, respectivement.*

Démonstration: Soit v un nœud de V. On distingue trois cas :

1. Si v appartient à $\mathcal{O}(G_0^\dagger)$, la Proposition 4.4.7 assure que $v \in \mathcal{O}(G_i^\dagger)$ pour tout indice i, $0 \leq i \leq k$. La règle T assure alors que v renverse au moins un de ses liens entrants à chaque pas de calcul qu'il effectue. Par conséquent, le nombre de renversements effectués par v est bien égal à $\omega(v, G_0^\dagger)$, qui est égal au nombre de pas de calcul effectués par v dans cette exécution.

2. Si v appartient à $\mathcal{N}(G_0^\dagger)$, la Proposition 4.4.7 implique, d'une part, $v \in \mathcal{N}(G_i^\dagger) \cup \mathcal{S}(G_i^\dagger)$ pour tout indice i, $0 \leq i \leq k$, et, d'autre part, que v migre de $\mathcal{S}^0$ vers $\mathcal{N}$ ou de $\mathcal{N}$ vers $\mathcal{S}^0$ lors de chacun de ses pas de calcul, et donc, comme on l'a remarqué plus haut, que v n'effectue un renversement qu'au cours d'un pas de calcul sur deux. Enfin, puisque v appartient à $\mathcal{N}(G_0^\dagger)$, pour toute D-chaîne $C_0^\dagger$ depuis v, on a $Res(C_0^\dagger) = 0$, ce qui implique $\omega(C_0^\dagger) = 2\rho(C_0^\dagger)$ et donc que le nombre de pas de calcul effectués par v est pair. On en conclut que le nombre de renversements effectué par v est égal à $\omega(v, G_0^\dagger)/2$.

3. Si v appartient à $\mathcal{S}(G_0^\dagger)$, il n'effectue un renversement qu'au cours d'un pas de calcul sur deux. Contrairement au cas précédent, pour toute D-chaîne $C_0^\dagger$ depuis v, on a ici $Res(C_0^\dagger) = 1$, ce qui implique $\omega(C_0^\dagger) = 2\rho(C_0^\dagger) + 1$ et donc que le nombre de pas de calcul effectués par v est impair. Puisqu'enfin la règle T assure que v effectue un renversement lors de son premier pas de calcul, on en conclut que le nombre de renversements effectués par v est égal à $1 + (\omega(v, G_0^\dagger) - 1)/2$, ou encore $\omega(v, G_0^\dagger)/2 + 1/2$.

□

Afin de mieux comprendre $\mathcal{LR}$, nous discutons ici l'influence que peut avoir l'étiquetage initial sur le nombre de renversements effectués. Pour cela, nous considérons à nouveau la politique d'étiquetage, introduite précédemment, qui consiste à ce que chacun des nœuds du graphe étiquette ses liens entrants de manière uniforme. Pour un nœud v donné, on note μ_v ($\mu_v \in \{0, 1\}$) cet étiquetage initial uniforme des liens entrants. Notons que si v et w sont deux nœuds distincts, cette stratégie autorise $\mu_v = 0$ et $\mu_w = 1$, et donc un étiquetage globalement non-uniforme du graphe. Comme nous l'avons montré, cette stratégie assure que tous les circuits d'un graphe dirigé acyclique vérifient la condition (4.1), suffisante pour notre solution. Étudions dans quelle mesure, pour un nœud u, le choix de μ_u influence le nombre de pas de calcul effectués par u et les autres nœuds au cours d'une exécution de

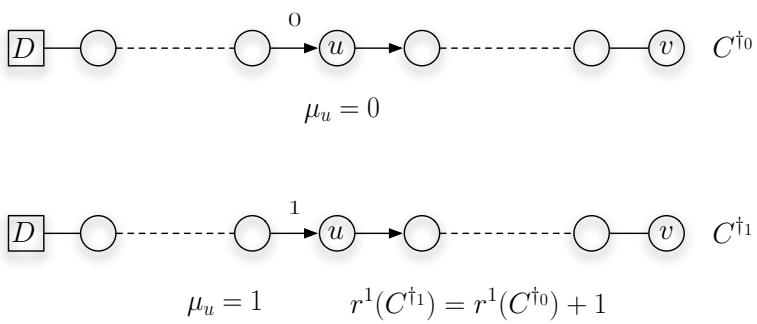

FIG. 4.7 –

$\mathcal{LR}$. On considère C, une D-chaîne depuis v, $v \neq u$, telle que u appartient à C. On note $C^{\dagger_0}$ et $C^{\dagger_1}$ les chaînes de support dirigé C telles que les liens entrants de u sont étiquetés uniformément avec des 0 et des 1, respectivement.

Considérons tout d'abord le cas dans lequel au moins l'un des liens de C incidents à u est sortant (u n'est pas un puits relativement à C). On a alors

$$l^1(C^{\dagger_1}) = l^1(C^{\dagger_0}) + 1$$

ou bien

$$l^1(C^{\dagger_1}) = l^1(C^{\dagger_0}) + 1.$$

Si on se place dans le deuxième cas, illustré en Figure 4.7, on a

$$\gamma(C^{\dagger_1}) - \delta(C^{\dagger_1}) \geq \gamma(C^{\dagger_0}) - \delta(C^{\dagger_0}).$$

Ainsi, la préférence de $\mu_u = 1$ par rapport à $\mu_u = 0$ accroît possiblement la complexité en travail de v (et par suite le nombre de renversements que ce dernier effectue). Sans connaissance globale de la topologie du graphe, le seul moyen de prévenir ce cas de figure consiste à choisir $\mu_u = 0$, ce qui implique que u appartient à $\mathcal{O}(G^{\dagger_0})$ et donc que le nombre de renversements effectués par u est multiplié par à peu près un facteur 2 par rapport au cas $\mu_u = 1$, dans lequel $u \in \mathcal{N}(G^{\dagger_1})$. Ainsi, un choix "altruiste" des étiquettes de u (qui diminue potentiellement le travail des autres nœuds) accroît le travail de u. Au contraire, l'approche égoïste qui consiste à privilégier la minimisation du travail de u (et donc à choisir $\mu_u = 1$) pourrait avoir comme conséquence d'augmenter le travail des autres nœuds.

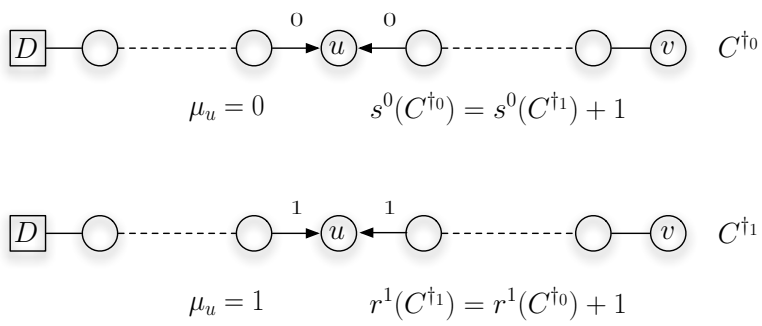

FIG. 4.8 –

Considérons à présent le cas où les deux liens de C incidents à u sont entrants (u est un puits relativement à C). Comme le montre la Figure 4.8, on a

$$s^0(C^{\dagger_0}) = s^0(C^{\dagger_1}) + 1 \quad \text{et}$$
$$r^1(C^{\dagger_1}) = r^1(C^{\dagger_0}) + 1.$$

Comme les étiquettes des liens entrants de u n'ont pas d'influence sur les valeurs $Res(C^{\dagger_0})$ et $Res(C^{\dagger_1})$, on en déduit que

$$\gamma(C^{\dagger_0}) - \delta(C^{\dagger_0}) = \gamma(C^{\dagger_1}) - \delta(C^{\dagger_1}).$$

Par conséquent, le travail de v ne dépend pas du choix de μ_u.

Dans le cas particulier où u est un puits de G, selon le choix de μ_u, on a u dans $\mathcal{N}(G^{\dagger_1})$, ou alors u appartient à $\mathcal{S}^0(G^{\dagger_0})$ (cf. Figure 4.9). La discussion concernant le nombre de renversements effectués par des nœuds initialement dans $\mathcal{S}^0$ ou $\mathcal{N}$ assure que le choix de μ_u dans ce cas n'a pas d'influence sur le nombre de renversements effectués par u. De plus, on a vu dans le paragraphe précédent que les deux choix possibles pour μ_u conduisent à la même complexité en travail des autres nœuds. Ainsi, dans le cas particulier d'un puits initial u, le choix de μ_u n'a d'influence sur la complexité en travail d'aucun nœud du graphe.

Lorsqu'on essaie de déterminer un étiquetage initial, la seule connaissance locale accessible par un nœud u est de savoir s'il est un puits ou non. Si c'est le cas, nous avons vu que le choix de μ_u n'a pas d'influence sur la complexité en travail globale. Sinon, il n'est pas possible de décider lequel des deux choix aura pour effet de diminuer le travail global en se fondant uniquement

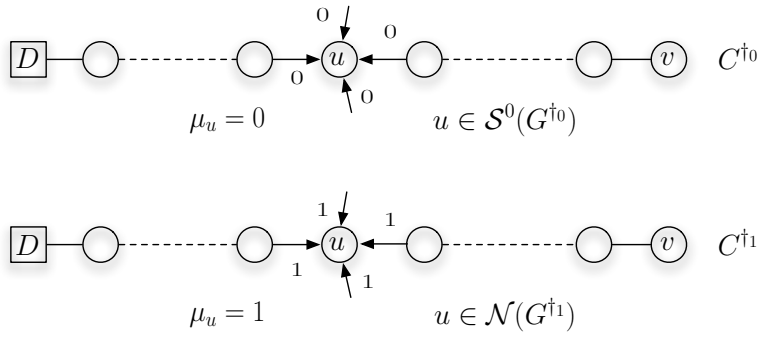

FIG. 4.9 –

sur la connaissance locale accessible par les nœuds. On en conclut que la minimisation de la complexité en travail globale nécessite une connaissance globale du graphe, et donc ne peut être aisément obtenue par une solution distribuée.

4.4.2 Complexité en temps : résultat partiel

Jusqu'ici, nous nous sommes concentré sur l'analyse de la complexité en travail de notre algorithme, et avons déterminé des formules exactes pour le nombre de pas de calcul et de renversements opérés par chacun des nœuds dans une exécution. Un autre sujet naturel d'étude est la *complexité en temps* qui peut être définie comme le nombre de rounds nécessaires à la convergence, au cours desquels plusieurs puits sont susceptibles d'effectuer un pas de calcul simultanément. On note que la complexité en travail globale correspond à la complexité en temps d'une exécution séquentielle, au cours de laquelle un seul puits prend un pas de calcul à chaque round.

Une question plus intéressante est de déterminer combien de rounds sont nécessaires pour qu'une exécution *complète* converge - on rappelle qu'à chaque round d'une exécution complète *tous les puits* prennent un pas de calcul. Notre approche nous a permis de déterminer une formule exacte pour la complexité en temps de $\mathcal{LR}$ sur une chaîne dont tous les liens sont marqués. Nous avons, pour cela, exhibé une fonction qui prend en argument une chaîne dirigée dont tous les liens sont marqués et dont la valeur, toujours positive, décroît de 1 un round sur deux jusqu'à être égale à 1 avant le dernier round.

A l'aide de cette formule, nous avons déduit une borne inférieure sur la complexité en temps de toute exécution de $\mathcal{LR}$ sur des graphes dont tous les liens sont initialement marqués.

Le cas des chaînes uniformément marquées

Soit $C_0^\dagger$ une chaîne dirigée dont tous les liens sont initialement marqués, et soit

$$C_0^\dagger, C_1^\dagger, \cdots, C_k^\dagger$$

la suite de chaînes dirigées induite par une telle exécution de $\mathcal{LR}$ sur $C_0^\dagger$, où $k \geq 0$ est le plus petit indice tel que $C_k^\dagger$ est un chemin dirigé vers la destination. On rappelle que le Théorème 4.3.1 implique l'existence d'un tel indice. Si $k = 0$, l'exécution est triviale et le nombre de rounds nécessaires à sa convergence est nul. On supposera donc dans la suite que $C_0^\dagger$ n'est pas un D-chemin, ou encore $k \geq 1$. Pour tout indice i, $0 \leq i \leq k$, et pour tout nœud u de $C_i^\dagger$, on note $C_i^\dagger(u)$ la sous-chaîne minimale de $C_i^\dagger$ contenant D et u. De plus, pour tout indice i, $0 \leq i \leq k$, on note $S(C_i^\dagger)$ l'ensemble des puits de $C_i^\dagger$, autres que D. Nous montrons tout d'abord que k est pair.

Lemme 4.4.9 *Pour tout indice i, $0 \leq i \leq k-1$, on a*

Si i est pair, alors $S(C_i^\dagger) = S(C_{i+1}^\dagger)$.

Si i est impair, alors $S(C_i^\dagger) \cap S(C_{i+1}^\dagger) = \emptyset$.

Démonstration: Puisque tous les liens de $C_0^\dagger$ sont marqués, on en déduit que tous les nœuds appartiennent initialement à $\mathcal{N}(C_0^\dagger)$. Le résultat est alors une conséquence directe de la Proposition 4.4.7, qui assure que tout nœud v migre de $\mathcal{N}$ vers $\mathcal{S}^0$ ou de $\mathcal{S}^0$ vers $\mathcal{N}$ à chaque pas de calcul qu'il effectue, combinée au fait que tout puits de $\mathcal{N}$ qui prend un pas de calcul ne renverse aucun de ses liens et donc reste un puits, et que tout puits de $\mathcal{S}^0$ qui prend un pas de calcul renverse et marque tous ses liens. □

Corollaire 4.4.10 *Le nombre de rounds de toute exécution complète de $\mathcal{LR}$ sur une chaîne dont tous les liens sont initialement marqués est pair.*

Démonstration: On raisonne par l'absurde et on suppose k impair. Par définition de k, on a $S(C_{k-1}^\dagger) \neq \emptyset$. Le Lemme 4.4.9 implique alors $S(C_k^\dagger) \neq \emptyset$, ce qui contredit le fait que $C_k^\dagger$ est un D-chemin. □

Nous introduisons quelques quantités supplémentaires. Pour tout indice i, $0 \leq i \leq k$, et pour tout nœud u de $C_i^\dagger$, soit $\alpha_u(C_i^\dagger)$ et $\overline{\alpha}_u(C_i^\dagger)$ les quantités définies de la manière suivante :

$$\alpha_u(C_i^\dagger) = \gamma(C_0^\dagger) - 1 - \delta(C_i^\dagger(u)), \quad et$$

$$\overline{\alpha}_u(C_i^\dagger) = \begin{cases} \alpha_u(C_i^\dagger) & si\ u \in S(C_i^\dagger) \\ 0 & sinon. \end{cases}$$

On définit enfin, pour tout indice i, $0 \leq i \leq k-1$, la quantité $\tau(C_i^\dagger)$ par

$$\tau(C_i^\dagger) = \max_{u \in S(C_i^\dagger)} \overline{\alpha}_u(C_i^\dagger).$$

On note que la fonction τ est bien définie puisque, par définition de k, on a $S(C_i^\dagger) \neq \emptyset$ pour tout indice i, $0 \leq i \leq k-1$. Les deux lemmes suivants montrent comment évolue la suite de valeurs $\tau(C_0^\dagger), \cdots, \tau(C_{k-1}^\dagger)$.

Proposition 4.4.11 *Pour tout indice pair i, $0 \leq i < k$, on a*

$$\tau(C_{i+1}^\dagger) = \tau(C_i^\dagger) - 1.$$

Démonstration: Soit i un entier pair tel que $i < k$. Par le Corollaire 4.4.10, on a donc $i \leq k-2$. Soit $u \in S(C_i^\dagger)$ tel que $\overline{\alpha}_u(C_i^\dagger) = \tau(C_i^\dagger)$. Puisque i est pair, le Lemme 4.4.9 assure que $S(C_i^\dagger) = S(C_{i+1}^\dagger)$. Par définition de k, on a aussi $S(C_{i+1}^\dagger) \neq \emptyset$. De plus, puisque l'exécution est complète, la Proposition 4.4.4 implique que, pour tout nœud w de $S(C_i^\dagger)$, on a $\delta(C_{i+1}^\dagger(w)) = \delta(C_i^\dagger(w)) + 1$. On en déduit

$$\tau(C_{i+1}^\dagger) = \max_{w \in S(C_{i+1}^\dagger)} \overline{\alpha}_w(C_{i+1}^\dagger) = \max_{w \in S(C_i^\dagger)} (\overline{\alpha}_w(C_i^\dagger) - 1) = \tau(C_i^\dagger) - 1.$$

□

Proposition 4.4.12 *Pour tout indice impair i, $0 < i < k-1$, on a*

$$\tau(C_{i+1}^\dagger) = \tau(C_i^\dagger).$$

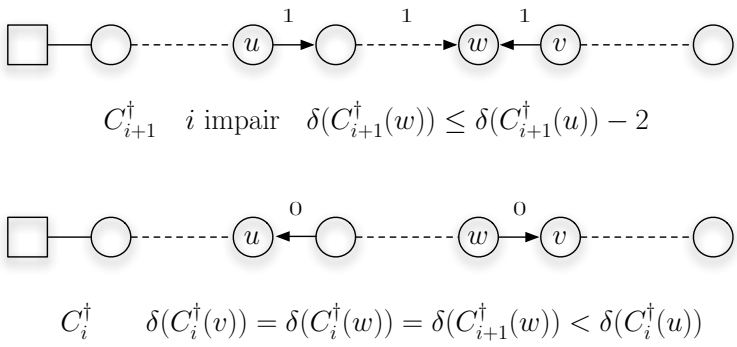

FIG. 4.10 –

Démonstration: Soit i un entier impair tel que $i < k$. Par le Corollaire 4.4.10, on a donc $i \leq k - 3$. Nous procédons en deux étapes.

On montre tout d'abord que $\tau(C^\dagger_{i+1}) \leq \tau(C^\dagger_i)$. Soit u dans $S(C^\dagger_i)$. Le Lemme 4.4.9 implique que u appartient à $S(C^\dagger_{i-1})$. La règle T assure alors que tous les liens (entrants) de u sont non-marqués dans $C^\dagger_i$. Par conséquent, u n'appartient pas à $S(C^\dagger_{i+1})$. On en déduit que tout nœud de $S(C^\dagger_{i+1})$ est un voisin d'un puits de $C^\dagger_i$. Soit donc $w \in S(C^\dagger_{i+1})$, u un voisin de w tel que $u \in S(C^\dagger_i)$, et e le lien reliant w à u. On a vu que e est non-marqué dans $C^\dagger_i$. Par conséquent, on a

$$\delta(C^\dagger_i(w)) = \delta(C^\dagger_i(u)).$$

D'autre part, la Proposition 4.4.4 assure

$$\delta(C^\dagger_{i+1}(w)) = \delta(C^\dagger_i(w)).$$

On obtient donc

$$\overline{\alpha}_w(C^\dagger_{i+1}) = \overline{\alpha}_u(C^\dagger_i).$$

On en conclut $\tau(C^\dagger_{i+1}) \leq \tau(C^\dagger_i)$.

On montre à présent qu'il existe un nœud $u \in S(C^\dagger_i)$ tel que

$$\overline{\alpha}_u(C^\dagger_i) = \tau(C^\dagger_i)$$

et tel qu'il existe un voisin w de u qui appartient à $S(C^\dagger_{i+1})$. On raisonne par l'absurde. On a alors que, pour tout nœud $u \in S(C^\dagger_i)$ tel que $\overline{\alpha}_u(C^\dagger_i) = \tau(C^\dagger_i)$,

il existe un nœud $w \in S(C^\dagger_{i+1})$ et un chemin dirigé de longueur l, $l \geq 2$, de u à w dans $C^\dagger_{i+1}$, comme illustré par la Figure 4.10. Puisque tous les liens sont marqués dans $C^\dagger_{i+1}$, on en déduit

$$\delta(C^\dagger_{i+1}(w)) \leq \delta(C^\dagger_{i+1}(u)) - 2.$$

Par la Proposition 4.4.4 , on obtient

$$\delta(C^\dagger_{i+1}(w)) \leq \delta(C^\dagger_i(u)) - 1.$$

L'argument utilisé ci-dessus implique alors qu'il existe un nœud v de $S(C^\dagger_i)$, voisin de w, tel que

$$\delta(C^\dagger_i(v)) \leq \delta(C^\dagger_i(u)) - 1$$

et donc, puisque u et v appartiennent tous deux à $S(C^\dagger_i)$,

$$\overline{\alpha}_v(C^\dagger_i) \geq \overline{\alpha}_u(C^\dagger_i) + 1,$$

ce qui contredit

$$\overline{\alpha}_u(C^\dagger_i) = \tau(C^\dagger_i) = \max_{z \in S(C^\dagger_i)} \overline{\alpha}_z(C^\dagger_i).$$

On en conclut donc qu'il existe un nœud u appartenant à $S(C^\dagger_i)$ tel que $\overline{\alpha}_u(C^\dagger_i) = \tau(C^\dagger_i)$ et tel qu'il existe un voisin w de u qui appartient à $S(C^\dagger_{i+1})$. Le même argument que précédemment montre que

$$\overline{\alpha}_w(C^\dagger_{i+1}) = \overline{\alpha}_u(C^\dagger_i),$$

et donc que $\tau(C^\dagger_{i+1}) \geq \tau(C^\dagger_i)$. On conclut finalement que $\tau(C^\dagger_{i+1}) = \tau(C^\dagger_i)$. $\square$

La dernière étape de notre raisonnement consiste à déterminer la valeur de $\tau(C^\dagger_{k-2})$. Soit v le premier nœud de $C^\dagger_{k-2}$ - c'est-à-dire le nœud tel que $C^\dagger_{k-2}$ est une D-chaîne depuis v - et soit e le lien incident à v. Le lemme suivant démontre que dans $C^\dagger_{k-2}$, tous les liens sont dirigés vers D et marqués, excepté e qui est dirigé vers v et marqué (cf. Figure 4.11).

Lemme 4.4.13 *Soit v le premier nœud de $C^\dagger_{k-2}$ et e le lien incident à v. Dans $C^\dagger_{k-1}$, tous les liens sont marqués et dirigés vers D excepté e qui est marqué mais dirigé vers v.*

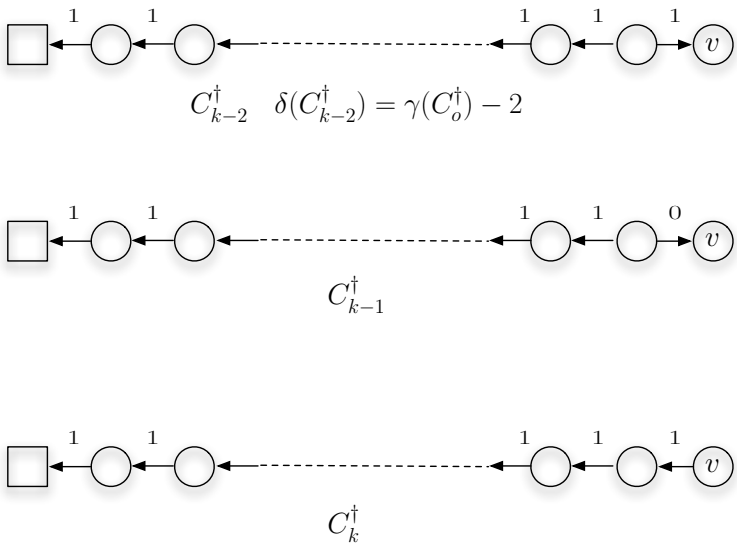

FIG. 4.11 –

Démonstration: On remarque tout d'abord que la règle T impose que tout lien entrant à un nœud qui n'est pas un puits est nécessairement marqué.

Supposons qu'il existe un nœud u différent de v tel que $C^\dagger_{k-2}(u)$ ne soit pas un D-chemin. Alors il existe un nœud w différent de v tel que $w \in S(C^\dagger_{k-2})$. Par le Corollaire 4.4.10, on a $k-2$ pair. Le Lemme 4.4.9 assure alors que w appartient à $S(C^\dagger_{k-1})$ et, par suite, que tous les liens incidents à w sont non-marqués dans $C^\dagger_{k-1}$. Par conséquent, la règle T impose que les deux liens incidents à w sont sortants dans $C^\dagger_k$, ce qui contredit le fait que $C^\dagger_k$ est un D-chemin. On en conclut donc que tous les liens différents de e sont dirigés vers D et marqués dans $C^\dagger_{k-2}$.

De plus, par définition de k, $C^\dagger_{k-2}$ n'est pas un D-chemin, et donc $S(C^\dagger_{k-2}) \neq \emptyset$. On en conclut donc que $S(C^\dagger_{k-2}) = \{v\}$. Si e est non-marqué dans $C^\dagger_{k-2}$, alors la règle T assure que v renverse e, et donc que $C^\dagger_{k-1}$ est un D-chemin, une contradiction. On en conclut que e est marqué dans $C^\dagger_{k-2}$. □

Proposition 4.4.14 *A la fin du round $k-2$, la fonction τ a pour valeur 1 ; en d'autres termes,*

$$\tau(C^\dagger_{k-2}) = 1.$$

Démonstration: Le Lemme 4.4.13 assure d'une part que v, le premier nœud de $C^\dagger_{k-2}$, est le seul puits de $C^\dagger_{k-2}$, ce qui implique

$$\tau(C^\dagger_{k-2}) = \overline{\alpha}_v(C^\dagger_{k-2}) = \gamma(C^\dagger_0) - 1 - \delta(C^\dagger_{k-2}(v)),$$

et, d'autre part, que tous les liens de $C^\dagger_{k-2}$ sont marqués et dirigés vers D, excepté le premier, qui est dirigé vers v et marqué, ce qui implique que $\delta(C^\dagger_{k-2}(v)) = \gamma(C^\dagger_0) - 2$. On en conclut donc

$$\tau(C^\dagger_{k-2}) = 1.$$

□

Pour toute D-chaîne $C^\dagger$, on note $\kappa(C^\dagger)$ la complexité en temps d'une exécution complète de $\mathcal{LR}$ sur $C^\dagger$. En combinant les Propositions 4.4.11, 4.4.12 et 4.4.14, on obtient le théorème suivant :

Théorème 4.4.15 *Soit $C^\dagger_0$ une D-chaîne dont tous les liens sont initialement marqués. On a*

$$\kappa(C^\dagger_0) = \begin{cases} 0 & \textit{si } C^\dagger_0 \textit{ est un } D\textit{-chemin,} \\ 2 \cdot \tau(C^\dagger_0) & \textit{sinon.} \end{cases}$$

Démonstration: Si $C^\dagger_0$ est un D-chemin, le résultat est immédiat. Supposons que $C^\dagger_0$ ne soit pas un D-chemin. On considère la suite de valeurs positives

$$\big(\tau(C^\dagger_0), \tau(C^\dagger_1)\tau(C^\dagger_{k-2}), \ldots, \tau(C^\dagger_{k-2})\big).$$

Les Propositions 4.4.11 et 4.4.12 assurent que, pour tout indice i, $0 \leq i \leq k-3$, on a :

$$\tau(C^\dagger_{i+1}) = \begin{cases} \tau(C^\dagger_i) - 1 & \text{si } i \text{ est pair,} \\ \tau(C^\dagger_i) & \text{sinon} \end{cases}$$

Par le Corollaire 4.4.10, $k-2$ est pair. La Proposition 4.4.14 implique alors

$$k - 2 = 2 \cdot (\tau(C^\dagger_0) - 1)$$

et, par conséquent,

$$\kappa(C^\dagger_0) = k = 2 \cdot \tau(C^\dagger_0).$$

□

Note:
Malgré de nombreuses tentatives, il nous a été impossible de trouver une fonction de potentiel telle que τ pour l'analyse de la complexité en temps d'une exécution complète de $\mathcal{LR}$ sur des chaînes dont tous les liens sont non-marqués. Nous avons également échoué pour le cas des arbres uniformément marqués. Dans ce cas particulier, nous n'avons pas été en mesure de déterminer une fonction qui capture la causalité entre les pas de calculs effectués par des nœuds n'appartenant pas à la même "branche". C'est la raison pour laquelle tous les résultats qui suivent ne s'appliquent, d'une part, qu'à des exécutions à partir de graphes uniformément marqués et, d'autre part, que nous ne fournissons qu'une borne inférieure et non une formule exacte pour la complexité en temps des exécutions à partir de graphes quelconques.

Borne inférieure pour les graphes uniformément marqués

Nous nous proposons ici d'utiliser le résultat précédent relatif aux chaînes uniformément marquées pour déduire une borne inférieure sur la complexité en temps d'une exécution complète de $\mathcal{LR}$ sur un graphe dirigé connexe acyclique dont tous les liens sont marqués.

Nous commençons par un résultat valable quel que soit l'étiquetage initial du graphe considéré, sous réserve qu'il satisfasse la condition (4.1). Soit $G_0^\dagger$ un graphe connexe dont les circuits satisfont tous la condition (4.1) et soit $G_0'^\dagger$ un sous-graphe connexe de $G_0^\dagger$ contenant tous les nœuds de $G_0^\dagger$ tel que, pour tout nœud $v \neq D$ de $G_0'^\dagger$, on a

$$\omega(v, G_0'^\dagger) = \omega(v, G_0^\dagger).$$

On considère une exécution complète de $\mathcal{LR}$ sur $G_0^\dagger$. Cette exécution induit un ordre partiel sur les pas de calcul effectués par les nœuds, par la relation de causalité *"est survenu avant"*. Si, pour tout nœud v, et pour tout entier i, $0 \leq i \leq \omega(v, G_0^\dagger)$, on note $T_i(v)$ le i-ième pas de calcul de v dans cette exécution, on peut ainsi définir cet ordre partiel par un graphe dirigé $\overline{G}$ dont l'ensemble des sommets est

$$\bigcup_{v \in V} \{T_i(v) : \ 0 \leq i \leq \omega(v, G_0^\dagger)\}$$

et tel qu'il existe un lien dirigé de $T_i(v)$ vers $T_j(w)$ si

1) v et w sont voisins dans $G_0^\dagger$ ou bien $v = w$,

2) $T_j(w)$ est le premier pas de calcul effectué par w après $T_i(v)$.

On définit la *profondeur* d'un sommet x de $\overline{G}$, notée $p(x)$, comme étant égale à 1 si x n'a pas de lien entrant (c'est le cas des pas de calcul effectués par les puits de $G_0^\dagger$) et, pour tout autre sommet y de $\overline{G}$, de la manière suivante :

$$p(y) = \max\{p(z) : \text{ il existe un lien dirigé de } z \text{ vers } y\} + 1.$$

La complexité en temps de l'exécution complète, notée $\kappa(G_0^\dagger)$, est alors égale à la profondeur de $\overline{G}$, notée $prof(\overline{G})$, définie comme le maximum des profondeurs des sommets de $\overline{G}$. En d'autres termes,

$$\kappa(G_0^\dagger) = prof(\overline{G}).$$

Considérons maintenant une exécution complète de $\mathcal{LR}$ sur $G_0'^\dagger$. On va montrer que la profondeur du graphe $\overline{G'}$ induit par cette exécution est au plus égale à celle de $\overline{G}$.

Proposition 4.4.16 *Soit $G_0^\dagger$ un graphe connexe dont les circuits satisfont tous la condition (4.1) et soit $G_0'^\dagger$ un sous-graphe connexe de $G_0^\dagger$ contenant tous les nœuds de $G_0^\dagger$ tel que, pour tout nœud $v \neq D$ de $G_0'^\dagger$, on a*

$$\omega(v, G_0'^\dagger) = \omega(v, G_0^\dagger).$$

Alors,

$$\kappa(G_0'^\dagger) \leq \kappa(G_0^\dagger).$$

Démonstration: Soit $\overline{G}$ et $\overline{G'}$ les graphes de dépendance induit par les exécutions complètes de $\mathcal{LR}$ sur $G_0^\dagger$ et sur $G_0'^\dagger$, respectivement. Par définition d'un graphe de dépendance, et puisque, par hypothèse, pour tout nœud $v \neq D$ de $G_0'^\dagger$, on a

$$\omega(v, G_0'^\dagger) = \omega(v, G_0^\dagger),$$

les graphes $\overline{G}$ et $\overline{G'}$ ont le même ensemble de sommets.

On va montrer que, pour tous sommets $x = T_i(v)$ et $y = T_j(w)$ de $\overline{G'}$, s'il existe un lien dirigé de x vers y dans $\overline{G'}$, alors il existe un lien dirigé de x vers y dans $\overline{G}$. On distingue deux cas :

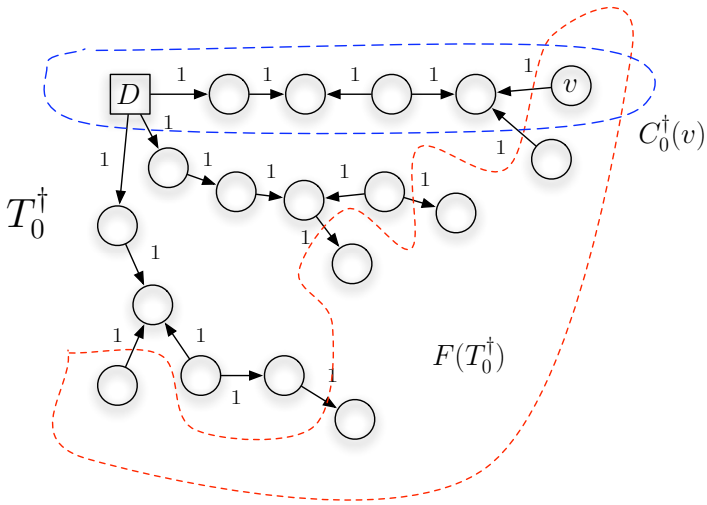

FIG. 4.12 – $F(T_0^\dagger)$ et $C_0^\dagger(v)$.

1. Si $v = w$, alors par définition de $\overline{G'}$, on a $j = i + 1$. Ainsi, x et y sont deux pas de calcul successifs de v. Par définition de $\overline{G}$, il existe bien un lien dirigé de x vers y dans $\overline{G}$.
2. Si $v \neq w$ alors v et w sont voisins dans $G_0'^\dagger$ et donc dans $G_0^\dagger$. Soit e le lien reliant v à w. Par hypothèse, la direction et l'étiquette de e sont identiques dans $G_0'^\dagger$ et dans $G_0^\dagger$. Puisque, d'autre part, la direction et l'étiquette de e ne peuvent être modifiées que par les pas de calcul effectués par v et par w, on en déduit que les pas de calculs de v et de w sont ordonnés de la même manière dans l'exécution sur $G_0^\dagger$ et dans celle sur $G_0'^\dagger$. On en conclut donc qu'il existe un lien entre x et y dans $\overline{G}$.

On vient de démontrer que $\overline{G'}$ est un sous-graphe de $\overline{G}$. Par conséquent, on a bien $prof(\overline{G'}) \leq prof(\overline{G})$, ce qui termine notre démonstration. □

La deuxième étape de notre raisonnement consiste à appliquer le résultat précédent au cas particulier d'un arbre $T_0^\dagger$ - pour chaque nœud v de $T_0^\dagger$, il existe une unique D-chaîne depuis v -, enraciné en D, dont tous les liens sont marqués. Par définition, $T_0^\dagger$ ne contient aucun circuit et donc satisfait

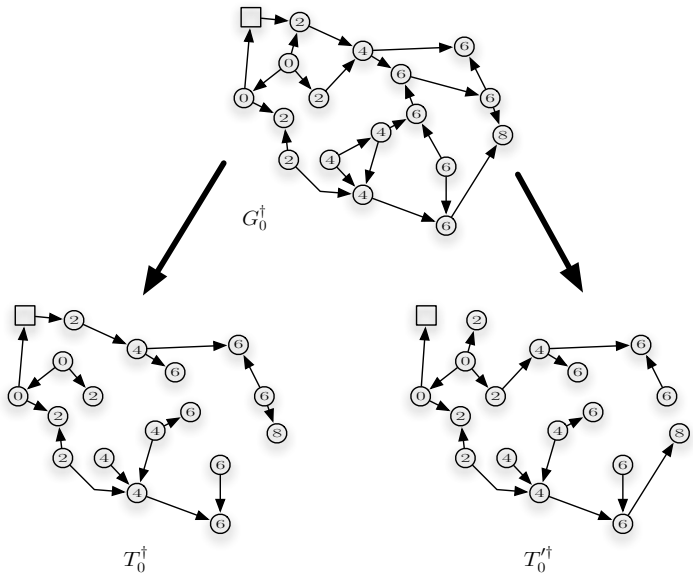

FIG. 4.13 – $G_0^\dagger$ et deux arbres couvrants qui préservent le travail des nœuds.

les conditions de la Proposition 4.4.16.

Un nœud v de $T_0^\dagger$, différent de D est appelé *feuille* si v n'a qu'un lien incident. On note $F(T_0^\dagger)$ l'ensemble des feuilles de $T_0^\dagger$ qui n'ont pas de chemin dirigé vers D dans $T_0^\dagger$. Un exemple est donné en Figure 4.12.

Soit $v \in F(T_0^\dagger)$ et soit $C_0^\dagger(v)$ l'unique D-chaîne depuis v dans $T_0^\dagger$. La définition du travail d'un nœud implique alors que $C_0^\dagger(v)$ est un sous-graphe de $T_0^\dagger$ qui satisfait les conditions de la Proposition 4.4.16. Cette dernière implique alors le résultat suivant :

Proposition 4.4.17 *Soit $T_0^\dagger$ un arbre, enraciné en D, dont tous les liens sont marqués et qui n'est pas orienté vers la destination. On a*

$$\kappa(T_0^\dagger) \geq \max_{v \in F(T_0^\dagger)} \kappa(C_0^\dagger(v)) = \max_{v \in F(T_0^\dagger)} 2 \cdot \tau(C_0^\dagger(v)).$$

Pour terminer, on déduit des Propositions 4.4.16 et 4.4.17 une borne inférieure sur la complexité en temps d'une exécution complète de $\mathcal{LR}$ sur un graphe acyclique connexe dont tous les liens sont marqués.

Soit $G_0^\dagger$ un tel graphe. On considère $T_0^\dagger$ un sous-graphe de $G_0^\dagger$ - il peut en exister plusieurs comme le montre la Figure 4.13 - tel que

1) $T_0^\dagger$ est un arbre enraciné en D,

2) $G_0^\dagger$ et $T_0^\dagger$ ont le même ensemble de sommets,

3) pour tout nœud $v \neq D$ on a $\omega(v, T_0^\dagger) = \omega(v, G_0^\dagger)$.

On dira que $T_0^\dagger$ est un *arbre couvrant qui préserve le travail*. Par définition, $G_0^\dagger$ et $T_0^\dagger$ satisfont les conditions de la Proposition 4.4.16. On en déduit donc la proposition suivante :

Proposition 4.4.18 *Soit $G_0^\dagger$ un graphe connexe acyclique dont tous les liens sont marqués et qui n'est pas orienté vers la destination. S'il existe $T_0^\dagger$ un arbre couvrant de $G_0^\dagger$ qui préserve le travail, on a*

$$\kappa(G_0^\dagger) \geq \kappa(T_0^\dagger).$$

Si on note $AC(G_0^\dagger)$ l'ensemble des arbres couvrants de $G_0^\dagger$ qui préservent le travail, en combinant le Théorème 4.4.15 et les Propositions 4.4.17 et 4.4.18, on obtient le théorème suivant :

Théorème 4.4.19 *Soit $G_0^\dagger$ un graphe connexe acyclique dont tous les liens sont marqués et qui n'est pas orienté vers la destination. On a :*

$$\kappa(G_0^\dagger) \geq \max_{T_0^\dagger \in AC(G_0^\dagger)} \left\{ \max_{v \in F(T_0^\dagger)} 2 \cdot \tau(C_0^\dagger(v)) \right\},$$

ou encore

$$\kappa(G_0^\dagger) \geq \max_{T_0^\dagger \in AC(G_0^\dagger)} \left\{ \max_{v \in F(T_0^\dagger)} \{2(\gamma(C_0^\dagger(v)) - 1 - \min_{w \in S(C_0^\dagger(v))} \delta(C_0^\dagger(w))\} \right\}.$$

Démonstration: Soit $G_0^\dagger$ un graphe connexe acyclique dont tous les liens sont marqués et qui n'est pas orienté vers la destination. On montre tout d'abord que $AC(G_0^\dagger)$ n'est pas vide. Pour cela, on construit un arbre couvrant de $G_0^\dagger$ qui préserve le travail des nœuds. On procède de manière récursive. Soit v un nœud de $G_0^\dagger$ dont le travail est supérieur au travail de tous les autres nœuds de $G_0^\dagger$, et soit $C_0^\dagger(v)$ une D-chaîne depuis v telle que

$$\omega(C_0^\dagger(v)) = \omega(v, G_0^\dagger).$$

Par définition de $C_0^\dagger$, pour tout nœud $w \in C_0^\dagger(v)$, on a

$$\omega(C_0^\dagger(w)) = \omega(w, G_0^\dagger).$$

En répétant ce processus sur l'ensemble des nœuds restants, on obtient bien un arbre couvrant de $G_0^\dagger$ qui préserve le travail. On en conclut que $AC(G_0^\dagger)$ est non-vide.

Soit $T_0^\dagger \in AC(G_0^\dagger)$. Puisque, par hypothèse, $G_0^\dagger$ n'est pas orienté vers D, il existe un nœud v qui n'a pas de chemin dirigé vers D dans $G_0^\dagger$. Puisque $T_0^\dagger$ est un sous-graphe de $G_0^\dagger$, on en déduit que w n'a pas de chemin dirigé vers D dans $T_0^\dagger$, et donc qu'il existe une feuille de $T_0^\dagger$ qui n'a pas de chemin dirigé vers D. On en conclut que $F(T_0^\dagger)$ n'est pas vide.

Enfin, soit $v \in F(T_0^\dagger)$. Par définition, $C_0^\dagger(v)$ n'est pas un D-chemin. Par conséquent, il existe $w \in C_0^\dagger(v)$ qui est un puits par rapport à $C_0^\dagger(v)$. Ainsi, $S(C_0^\dagger(v))$ est non-vide.

On vient de montrer que l'expression

$$\max_{T_0^\dagger \in AC(G_0^\dagger)} \left\{ \max_{v \in F(T_0^\dagger)} \{2(\gamma(C_0^\dagger(v)) - 1 - \min_{w \in S(C_0^\dagger(v))} \delta(C_0^\dagger(w))\} \right\}$$

est bien définie. Le résultat est alors une conséquence immédiate de la combinaison du Théorème 4.4.15 et des Propositions 4.4.17 et 4.4.18. □

4.4.3 Cas particuliers de *FR* et *PR*

Nous utilisons ici les résultats de la section précédente pour déterminer la complexité en travail de *FR* et *PR*. Nous déduisons d'autre part, dans le cas de *FR*, une formule exacte pour les cas particulier des chaînes, et une borne inférieure pour le cas général.

Complexité en travail de *FR* et *PR*

On rappelle que toute exécution *FR* (resp. *PR*) est identique à une éxécution de $\mathcal{LR}$ sur des graphes uniformément marqués (resp. non-marqués), dans laquelle tout processus applique la règle LR2 (resp. les règles LR1 et LR2). Comme nous l'avons vu, LR1 est équivalente à T, tandis que

l'exécution de LR2 est équivalente à l'exécution de deux transitions T successives. On en déduit donc que le nombre de pas de calcul effectué par un nœud dans une exécution de FR (resp. PR) est égal au nombre de renversements effectués par ce nœud dans une exécution de $\mathcal{LR}$ à partir d'un graphe dont tous les liens sont initialement marqués (resp. non-marqués).

Nous adaptons les notations, introduites précédemment, à ces deux cas particuliers d'étiquetages initiaux uniformes. Etant donné un graphe dirigé acyclique G, pour tout nœud v de V, on note $\mathcal{C}(v, G)$ l'ensemble des D-chaînes depuis v ; pour toute chaîne C de $\mathcal{C}(v, G)$, soit $r(C)$ le nombre de liens orientés contrairement à l'orientation naturelle de C, $l(C)$ le nombre de liens orientés selon cette orientation, et $s(C)$ le nombre de nœuds ayant deux liens entrants relativement à C. Nous définissons aussi $Res(C)$, le résidu de C, comme étant égal à 1 si et seulement si le premier lien de C est dirigé vers le premier nœud de C. On notera que, dans le cas d'une exécution à partir d'un graphe dont tous les liens sont initialement marqués (FR), l'ensemble $\mathcal{S}$ est initialement vide tandis que $\mathcal{O}$ est vide tout au long de l'exécution. D'autre part, dans le cas où tous les liens sont initialement non-marqués (PR), les ensembles $\mathcal{S}$ et $\mathcal{N}$ contiennent tous les puits et toutes les sources, respectivement. Le Théorème 4.4.8 pour a corollaire immédiat :

Corollaire 4.4.20 *1. Le nombre de renversements effectués par v dans une exécution de FR est*

$$\min_{C \in \mathcal{C}(v,G)} (r(C)) .$$

2. Le nombre de renversements effectués par v dans une exécution de PR est

$$\begin{array}{ll} \min_{C \in \mathcal{C}(v,G)} (s(C) + Res(C)) , & \textit{si } v \textit{ est un puits ou une source dans } G \\ \min_{C \in \mathcal{C}(v,G)} (2s(C) + Res(C)) , & \textit{sinon.} \end{array}$$

Ce corollaire nous permet de comprendre quelles propriétés des graphes d'entrée engendrent des pas de calcul (renversements) pour FR et PR. En particulier, cela nous conduit à décrire les deux chaînes dirigées données en Figure 4.14 pour lesquelles il existe une différence importante entre les complexités en travail globales de FR et PR, respectivement. En effet, pour la chaîne supérieure, le nombre de pas de calcul exécutés dans une exécution de FR est $O(n^2)$ alors qu'il est seulement n pour PR. Au contraire, pour la chaîne inférieure, la complexité globale en travail pour FR est égale à n, alors

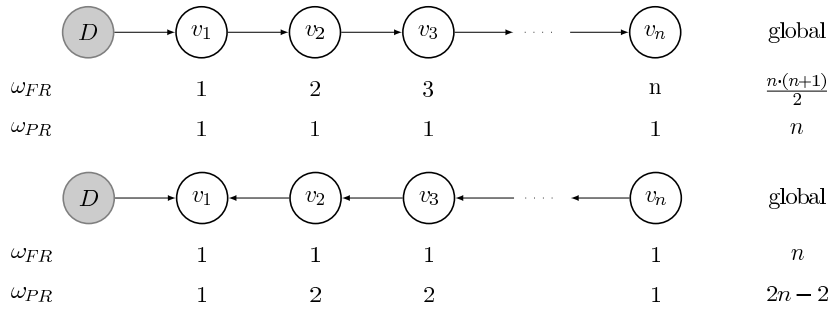

FIG. 4.14 – Exemple de chaînes montrant que ni FR ni PR n'est meilleur *en général*.

que celle de PR est égale à $2n-2$, c'est-à-dire que PR est "moins bon" d'à peu près un facteur 2 sur ce graphe.

De la même manière, il est possible de décrire d'autres graphes sur lesquels les performances de FR et PR seront mauvaises. Nous comprenons ainsi formellement les graphes utilisés par Busch *et al.* pour démontrer que leurs bornes sont atteintes. Leurs résultats seront discutés en détail dans la suite. En général, ni FR ni PR ne présente de meilleures performances, c'est-à-dire qu'il existe des graphes dirigés acycliques pour lesquels un étiquetage initial uniforme avec des 1 requiert moins de pas de calcul qu'un étiquetage uniforme avec des 0, et inversement. D'autre part, la discussion concernant l'influence de l'étiquetage initial sur le travail des nœuds indique que toute tentative d'optimisation de cette complexité par des variations de l'étiquetage initial nécessite une connaissance de la topologie globale du graphe.

Complexité en temps de FR : résultat partiel

Soit G_0 un graphe dirigé connexe acyclique et soit $G_0^\dagger$ le graphe dirigé étiqueté obtenu en affectant l'étiquette 1 à tous les liens de G_0. Puisque G_0 est acyclique, tous les circuits de $G_0^\dagger$ satisfont la condition (4.1). On montre tout d'abord que la complexité en temps d'une exécution complète de FR sur G_0, notée $\kappa_{FR}(G_0)$, est égale à la moitié de celle d'une exécution complète de $\mathcal{LR}$ sur $G_0^\dagger$. En d'autre termes :

$$\kappa_{FR}(G_0) = \frac{\kappa(G_0^\dagger)}{2}.$$

On suppose que G_0 (et par conséquent $G_0^\dagger$) n'est pas D-orienté. On considère les suites

$$G_0, G_1, \ldots, G_k$$

et

$$G_0^\dagger, G_1^\dagger, \ldots G_{k'}^\dagger$$

des graphes engendrés par une exécution complète maximale de FR sur G_0 et par une exécution complète maximale de $\mathcal{LR}$ sur $G_0^\dagger$, respectivement. On reprend les notations introduites précédemment et, pour tous indices $i \in \{0, 1, \ldots, k\}$ et $j \in \{0, 1, \ldots, k'\}$, on note $S(G_i)$ l'ensemble des puits de G_i, et $S(G_j^\dagger)$ l'ensemble de ceux de $G_j^\dagger$.

Proposition 4.4.21 *Pour tout indice i, $0 \leq i \leq k$, G_i est le support dirigé de $G_{2i}^\dagger$.*

Démonstration: On raisonne par récurrence sur i, $0 \leq i \leq k$.

- *Cas de base $i = 0$:* Par définition, G_0 est le support dirigé de $G_0^\dagger$.

- *Étape de récurrence $0 < i < k$:* Supposons que G_i est le support dirigé de $G_{2i}^\dagger$. Par définition de k, G_i n'est pas D-orienté, et donc $S(G_{2i}^\dagger) \neq \emptyset$. Par le Corollaire 4.4.10, on sait que k' est pair. On en déduit $2i \leq k' - 2$. Le Lemme 4.4.9 assure alors que $S(G_{2i}^\dagger) = S(G_{2i+1}^\dagger)$ et, puisqu'un pas de calcul effectué par un puits ne peut modifier que les directions de ses liens incidents, que $S(G_{2i}^\dagger)$ et $S(G_{2i+1}^\dagger)$ ont le même support dirigé G_i.
  D'autre part, la définition de la règle T implique que tous les liens entrants des puits de $S(G_{2i+1}^\dagger)$ sont non-marqués. Puisqu'on considère des exécutions complètes de FR et $\mathcal{LR}$, le fonctionnement des deux algorithmes assure que tous les liens incidents des puits de $S(G_i) = S(G_{2i+1}^\dagger)$ sont sortants dans G_{i+1} et $G_{2i+2}^\dagger$, respectivement. Enfin, puisqu'un pas de calcul effectué par un puits ne peut modifier que les directions de ses liens incidents, on en conclut que G_{i+1} est le support dirigé de $G_{2i+2}^\dagger$.

□

On en déduit immédiatement le théorème suivant

Théorème 4.4.22 *Soit G_0 un graphe dirigé connexe acyclique et soit $G_0^\dagger$ le graphe dirigé étiqueté obtenu en affectant l'étiquette 1 à tous les liens de G_0. On a :*

$$\kappa_{FR}(G_0) = \frac{\kappa(G_0^\dagger)}{2}.$$

Démonstration: Si $G_0^\dagger$ et G_0 sont D orientés, on a

$$\kappa_{FR}(G_0) = \kappa(G_0^\dagger) = 0.$$

Si $G_0^\dagger$ et G_0 ne sont pas D orientés, le résultat est celui énoncé dans la Proposition 4.4.21. □

On reprend ici les notations de la sous-section précédente. Etant donné un graphe dirigé acyclique G, pour tout nœud v de V, on note $\mathcal{C}(v, G)$ l'ensemble des D-chaînes depuis v ; pour toute chaîne C de $\mathcal{C}(v, G)$, soit $r(C)$ le nombre de liens orientés contrairement à l'orientation naturelle de C, $l(C)$ le nombre de liens orientés selon cette orientation, $S(C)$ l'ensemble des nœuds ayant deux liens entrants relativement à C et $s(C)$ le cardinal de $S(C)$. Nous définissons aussi $Res(C)$, le résidu de C, comme étant égal à 1 si et seulement si le premier lien de C est dirigé vers le premier nœud de C. On définit les quantités supplémentaires suivantes :

$$\delta(C) = l(C) = r(C) \quad \text{et} \quad \gamma(C) = l(C) + r(C) + Res(C).$$

En combinant les Théorèmes 4.4.15 et 4.4.22, on obtient le théorème suivant :

Théorème 4.4.23 *Soit C_0 une chaîne dirigée. On a :*

$$\kappa_{FR}(C_0) = \begin{cases} 0 & \textit{si } C_0 \textit{ est un } D\textit{-chemin}, \\ \gamma(C_0) - 1 - \min_{v \in S(C_0)} \delta(C_0(v)) & \textit{sinon} \end{cases}$$

Nous terminons en présentant une borne inférieure sur la complexité en temps d'une exécution complète de FR sur un graphe acyclique connexe quelconque. Pour cela, nous modifions les notations introduites dans la section 4.4.2. Soit G_0 un graphe acyclique connexe et soit $G_0^\dagger$ le graphe dirigé étiqueté obtenu en affectant l'étiquette 1 à tous les liens de G_0. On note $AC(G_0)$ l'ensemble des supports dirigés des graphes de $AC(G_0^\dagger)$ et, pour tout $T_0 \in AC(G_0)$, on note $F(T_0)$ l'ensemble des feuilles de T_0 qui n'ont pas de chemin dirigé vers D.

En combinant les Théorèmes 4.4.19 et 4.4.22, on obtient alors :

Théorème 4.4.24 *Soit G_0 un graphe acyclique connexe qui n'est pas D-orienté. On a :*

$$\kappa(G_0) \geq \max_{T_0 \in AC(G_0)} \left\{ \max_{v \in F(T_0)} \{\gamma(C_0(v)) - 1 - \min_{w \in S(C_0(v))} \delta(C_0(w))\} \right\}.$$

4.4.4 Comparaison avec les résultats de Busch *et al.*

Busch *et al.* [8, 9] ont été les premiers à étudier les complexités en travail et en temps des algorithmes de renversements de liens introduits par Gafni et Bertsekas dans [23]. Ils ont analysé le nombre total de renversements effectués dans une exécution comme une fonction de n, le nombre de nœuds n'ayant pas de chemin dirigé vers la destination dans le graphe initial (appelés *mauvais* nœuds). Dans cette sous-section, nous comparons nos résultats aux leurs.

Complexité en travail

En ce qui concerne l'implémentation de FR avec des hauteurs représentées par des paires, ils ont déterminé le nombre exact de renversements effectués par chacun des nœuds. Pour cela, ils ont partitionné l'ensemble des mauvais nœuds en *couches* ("layers") $L_1, L_2, \ldots, L_m$ définies de la manière suivante (cf. Figure 4.15). Un mauvais nœud v est dans la couche L_1 si l'une des deux conditions suivantes est satisfaite :

- Il existe un *bon* nœud w tel qu'il existe un lien dirigé de w vers v, ou
- il existe un mauvais nœud z tel qu'il existe un lien dirigé de v vers z.

Un mauvais nœud v est dans la couche L_j, $j > 1$, si j est le plus petit entier tel que l'une de deux conditions suivantes est satisfaite :

- Il existe un nœud $w \in L_{j-1}$ tel qu'il existe un lien dirigé de w vers v, ou
- il existe un nœud $z \in L_j$ tel qu'il existe un lien dirigé de v vers z.

Busch *et al.* ont alors démontré qu'un nœud effectuait exactement j renversements si et seulement si il appartenait à L_j. Or, on remarque qu'un nœud v appartient à la couche L_j si et seulement si

$$\min\{r(C) : \quad C \text{ est une } D\text{-chaîne depuis } v\} = j.$$

Ainsi, v appartient à la couche L_j si et seulement si notre formule pour le nombre de renversements effectués par v (Corollaire 4.4.20) est égale à i. Par

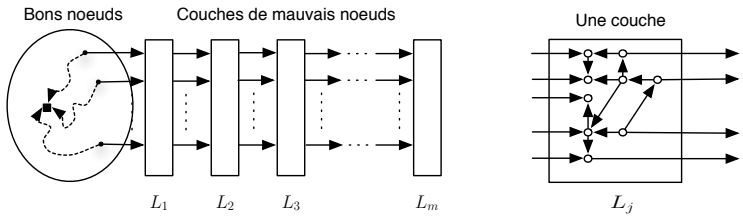

FIG. 4.15 – Partitionnement des mauvais nœuds en couches.

conséquent, nos résultats sont les mêmes que ceux de [8, 9] pour la complexité en travail de *FR*.

Dans ces deux approches, on peut estimer que chacun des n mauvais nœuds effectue au plus n renversements dans une exécution : dans l'approche de Busch *et al.* il y a au plus n couches, et dans la nôtre il y a au plus n liens dans un chemin reliant un mauvais nœud à un bon nœud ou à la destination. Il s'ensuit que la complexité en travail globale est en $O(n^2)$. Comme dans [8, 9], on peut dériver de notre formule un graphe particulier pour lequel toute exécution comporte $\Omega(n^2)$ renversements : le graphe consiste en une chaîne de $n+1$ nœuds telle que D en est l'une des extrémités et tous les liens sont dirigés contrairement à l'orientation naturelle sur cette chaîne ; plus précisément cette chaîne est un chemin dirigé de D vers l'autre extrémité, c'est-à-dire la chaîne supérieure de la Figure 4.14.

En ce qui concerne l'implémentation de *PR* utilisant des triplets comme hauteurs, Busch *et al.* ont calculé une borne supérieure et une borne inférieure sur le nombre de renversements effectués par chacun des nœuds. Leur résultat de borne supérieure utilise un partitionnement de l'ensemble des mauvais nœuds initiaux en *niveaux* (à ne pas confondre avec les couches précédentes). Un mauvais nœud v est au niveau j (on note $Level(v) = j$), si et seulement si le plus court chemin non-dirigé de v à un bon nœud est de longueur i. Busch *et al.* ont alors démontré qu'un nœud v tel que $Level(v) = j$ effectuait au plus j renversements. Comparons cette borne avec notre formule exacte. On peut distinguer trois cas, selon les directions des liens incidents à v dans le graphe initial G :

1. Si $v \in \mathcal{O}$ initialement, alors on a immédiatement

$$\forall C \in \mathcal{C}(v, G) : \quad 2s(C) + Res(C) \leq i(C),$$

où $i(C)$ est le nombre de mauvais nœuds appartenant à C. On en déduit

$$\min_{C \in \mathcal{C}(v,G)} [2s(C) + Res(C)] \leq \min_{C \in \mathcal{C}(v,G)} \{i(C)\},$$

ce qui équivaut à

$$\min_{C \in \mathcal{C}(v,G)} [2s(C) + Res(C)] \leq Level(v).$$

2. Si $v \in \mathcal{S}$ initialement, alors $Res(C) = 1$ pour toute chaîne C de $\mathcal{C}(v, G)$. Par conséquent, on a

$$\forall C \in \mathcal{C}(v, G) : \quad s(C) + Res(C) \leq \left\lfloor \frac{i(C) + 1}{2} \right\rfloor,$$

ce qui implique

$$\min_{C \in \mathcal{C}(v,G)} [s(C) + Res(C)] \leq \left\lfloor \frac{Level(v) + 1}{2} \right\rfloor.$$

3. Si $v \in \mathcal{N}$ initialement, alors $Res(C) = 0$ pour toute chaîne C de $\mathcal{C}(v, G)$. De plus, on a

$$\forall C \in \mathcal{C}(v, G) : \quad s(C) + Res(C) \leq \frac{i(C)}{2}.$$

Par conséquent, on obtient

$$\min_{C \in \mathcal{C}(v,G)} [s(C) + Res(C)] \leq \min_{C \in \mathcal{C}(v,G)} \left\{ \frac{i(C)}{2} \right\},$$

et donc

$$\min_{C \in \mathcal{C}(v,G)} [s(C) + Res(C)] \leq \left\lfloor \frac{Level(v)}{2} \right\rfloor.$$

Grâce à notre approche, on peut, d'une part, exprimer le nombre exact de renversements effectués par chacun des nœuds dans une exécution de PR, mais aussi montrer que la borne obtenue par Busch *et al.* est au moins deux fois plus grande que notre formule exacte pour les nœuds appartenant initialement à $\mathcal{S}$ ou $\mathcal{N}$.

Pour établir leur résultat de borne inférieure, Busch *et al.* utilisent à nouveau des *couches* - différentes de celles qui sont utilisées dans le cas de FR. Ils considèrent des graphes dont l'ensemble des mauvais nœuds peut être partitionné en un nombre *pair* de couches $L_1, L_2, \ldots, L_m$ de la manière suivante.

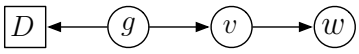

FIG. 4.16 –

- Les couches d'indice pair ne contiennent que des sources, tandis que les couches d'indice impair ne contiennent que des nœuds qui ne sont pas des sources.
- La couche L_1 contient tous les mauvais nœuds adjacents à un bon nœud.
- Un nœud dans la couche L_j, $j > 1$, ne peut être adjacent qu'à des nœuds de L_{j-1}, L_j ou L_{j+1}.
- Tout nœud de L_j est adjacent à au moins un nœud de L_{j-1} et à au moins un nœud de L_{j+1}.

Note:

On remarque qu'il existe des graphes dont l'ensemble des mauvais nœuds ne peut pas être partitionné de cette façon. Considérons, par exemple, la chaîne de la Figure 4.16. Le nœud w n'est pas adjacent à g, le seul bon nœud. Par conséquent w ne peut pas appartenir à L_1. Or w n'est pas une source, et ne peut donc pas appartenir à L_2. Ainsi, w ne peut appartenir à aucune couche.

Ils ont alors démontré que, pour tout indice $j \leq m/2$, si un nœud appartient à L_{2j-1} ou à L_{2j}, alors il effectue au moins $\lfloor i/2 \rfloor$ renversements. Utilisons notre formule exacte pour discuter la pertinence de leur borne. On distingue à nouveau trois cas, selon les directions des liens incidents aux mauvais nœuds initiaux.

1. Si $v \in \mathcal{N}$ initialement, alors v est une source. Par conséquent v appartient à une couche d'indice pair, que l'on note L_{2j}, $1 \leq j \leq m/2$. La définition des couches assure

$$\forall C \in \mathcal{C}(v, G): \quad s(C) \geq i \quad \text{et} \quad Res(C) = 0,$$

et par conséquent

$$\forall C \in \mathcal{C}(v, G): \quad s(C) + Res(C) \geq i.$$

On obtient ainsi

$$\min_{C \in \mathcal{C}(v,G)} [s(C) + Res(C)] \geq i.$$

2. Si $v \in \mathcal{S}$ initialement alors v est un puits et appartient à une couche d'indice impair, que l'on note L_{2j-1}, $1 \leq j \leq m/2$. On a

$$\forall C \in \mathcal{C}(v,G): \quad s(C) \geq i-1 \ \text{ et } \ Res(C) = 1.$$

Par conséquent

$$\forall C \in \mathcal{C}(v,G): \quad s(C) + Res(C) \geq i,$$

et donc

$$\min_{C \in \mathcal{C}(v,G)} [s(C) + Res(C)] \geq i.$$

3. Si $v \in \mathcal{O}$ initialement alors v appartient à une couche d'indice impair, que l'on note L_{2j-1}, $1 \leq j \leq m/2$. On a

$$\forall C \in \mathcal{C}(v,G): \ (s(C) \geq i \wedge Res(C) = 0) \vee (s(C) \geq i-1 \wedge Res(C) = 1).$$

Ainsi

$$\forall C \in \mathcal{C}(v,G): \quad 2s(C) + Res(C) \geq 2i - 1,$$

et donc

$$\min_{C \in \mathcal{C}(v,G)} [2s(C) + Res(C)] \geq 2i - 1.$$

On voit ainsi que leur borne est deux fois plus petite que notre formule exacte dans le cas des puits et sources initiaux, et quatre fois plus petite pour les autres mauvais nœuds initiaux. D'autre part, cette borne a été établie pour une classe très particulière de graphes initiaux. Au contraire, notre approche fournit une formule exacte pour la complexité en travail de toute exécution de *PR* à partir de n'importe quel graphe.

Complexité en temps

Dans [8, 9], Busch *et al.* traitent aussi la question de la complexité en temps des algorithmes de renversements de liens utilisant des hauteurs.

Pour l'implémentation de *FR* avec des paires comme pour celle de *PR* avec des triplets, ils utilisent leurs résultats sur la complexité en travail d'un

nœud pour déduire une borne supérieure sur le nombre de rounds nécessaires à la convergence de toute exécution. Pour cela, ils remarquent, pour les deux implémentations, que la convergence de toute exécution séquentielle nécessite un nombre de rounds au moins égal à celui d'une exécution dans laquelle, à chaque round, plusieurs puits sont susceptibles de prendre un pas de calcul. Ils concluent donc que le nombre de rounds nécessaires à la convergence d'une exécution sur un graphe donné est au moins égal à la somme des complexités en travail des nœuds du graphe. Or, pour tout graphe, l'ensemble des n mauvais nœuds peut être partitionné en au plus n couches ou niveaux.

Dans le premier cas, on a vu qu'un nœud appartenant à la couche j effectuait exactement j renversements. Par conséquent, la somme des complexités en travail de tous les nœuds est au plus égale à n^2. Ils montrent ainsi que toute exécution de *FR* nécessite $O(n^2)$ rounds pour converger. De la même manière, dans l'implémentation de *PR*, chacun des n mauvais nœuds initiaux peut effectuer au plus n renversements. Par conséquent, la convergence de toute exécution de *PR* nécessite au plus $O(n^2)$ rounds.

Ce raisonnement ne dépendant pas de l'approche utilisée, on peut trivialement déduire ces bornes de nos formules exactes.

En utilisant leurs résultats sur la complexité en travail des nœuds et en appliquant le même type de raisonnement que ci-dessus, Busch *et al.* déterminent, pour chacune des deux implémentations, un graphe tel que toute exécution complète - dans laquelle, à chaque round, tous les puits prennent un pas de calcul - requiert $\Omega(n^2)$ rounds.

Pour *FR*, ils considèrent le graphe donné en Figure 4.17, contenant $n = 8$ mauvais nœuds. Les quatre nœuds de L_5 effectuent chacun 5 renversements. De plus, tous les renversements de ces nœuds sont effectués séquentiellement puisqu'ils sont tous voisins deux à deux. Ainsi, une exécution de *FR* sur ce graphe requiert au moins $20 = \frac{n}{2}(\frac{n}{2} + 1) = \Omega(n^2)$ rounds pour converger.

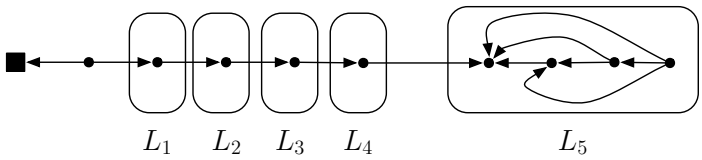

FIG. 4.17 – Un graphe "pire cas" pour la complexité en temps de *FR*.

Pour PR, ils considèrent le graphe donné en Figure 4.18, contenant $n = 8$ mauvais nœuds. Les mauvais nœuds sont partitionnés en $m_1 = n/2 + 2$ couches, et la couche L_{m_1-1} contient $m_2 = n/2$ mauvais nœuds. Leur borne inférieure sur le travail des nœuds implique que chacun des m_2 nœuds de L_{m_1-1} effectue au moins $k_1 = \lfloor \lceil (m_1 - 1)/2 \rceil / 2 \rfloor$ renversements. De plus, tous les renversements de ces nœuds sont effectués séquentiellement puisqu'ils sont tous voisins deux à deux. Ainsi, une exécution de PR sur ce graphe requiert au moins $(\frac{n}{2} - 1) \cdot \left\lfloor \lceil (\frac{n}{2} + 1)/2 \rceil / 2 \right\rfloor = \Omega(n^2)$ rounds pour converger.

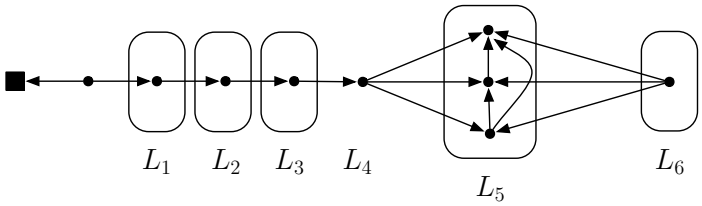

FIG. 4.18 – Un graphe "pire cas" pour la complexité en temps de PR.

Comme c'était le cas pour la borne supérieure, cette démarche ne dépend pas de l'approche utilisée. Ces résultats peuvent de le même manière être déduits de nos formules exactes. D'autre part, alors que la borne inférieure de Busch *et al.* n'est valable que pour certains graphes, notre approche nous permet de donner une formule exacte pour la complexité en temps de FR sur des chaînes, et d'en déduire une borne inférieure sur la complexité en temps de *toute* exécution de FR.

Pour conclure, il apparaît que notre approche est plus efficace - tous les résultats obtenus par Busch *et al.* peuvent être obtenus grâce à notre approche, et cette dernière nous permet d'améliorer certains des résultats en donnant des formules exactes et non des bornes. Notre approche est aussi plus générale, en ce sens que tous nos résultats sont valables pour tous les graphes, alors que certains de ceux de Busch *et al.* ne s'appliquent qu'à des graphes particuliers.

Conclusion

La possibilité de collaboration entre les processus d'un système distribué est étroitement liée à la capacité qu'ont ces processus de communiquer. Nous nous sommes, dans la présente thèse, intéressé aux problèmes de communication dans les systèmes distribués, qu'il s'agisse d'impossibilité de transmission ou d'altération de l'information transmise. Nous avons abordé cette étude sous deux angles distincts.

Généralisation et analyse du modèle HO

Dans un premier temps, nous avons pris le parti de supposer que la coopération entre les processus devait se faire *en dépit* d'éventuels problèmes de communication pouvant survenir, approche communément adoptée par la communauté de la Tolérance aux pannes. Nous nous sommes concentré sur l'analyse du modèle HO, défini par Charron-Bost et Schiper [17] pour l'étude des systèmes distribués sujets à des pannes *bénignes*, qui comprennent les *crashs* - arrêt prématuré et définif d'un processus - ainsi que les *omissions* d'envoi ou de réception ou encore les pertes de messages. Ce modèle fournit un cadre d'étude unifié des systèmes sujets à ces types de pannes en capturant le modèle de pannes et le degré de synchronisme par une unique entité abstraite, le *prédicat de communication*, qui caractérise les propriétés des communications du système.

Contributions

La première contribution du travail que nous avons présenté ici consiste en une généralisation du modèle HO, qui permet de décrire des systèmes distribués sujets à un autre type de pannes, les *erreurs de transmission par valeurs* [13]. Alors que les pannes bénignes ont pour seul effet une absence

de communication entre certains processus, les erreurs de transmission par valeurs, qui couvrent les pannes *byzantines* [30] ainsi que les *corruptions dynamiques* [42], se caractérisent par une altération de l'information échangée. Nous avons, pour cela, enrichi le modèle HO - initialement fondé sur la notion d'*ensemble d'écoute* permettant de caractériser, pour un processus p, l'ensemble des processus dont p reçoit un message à un round donné - en introduisant la notion d'*ensemble d'écoute sûr* qui caractérise la sûreté des communications, en ce sens que cet ensemble représente, à chaque round et pour chaque processus p, l'ensemble des processus dont p reçoit un message dont le contenu est identique à celui qui aurait dû lui être envoyé à ce round.

Le modèle HO se situe à un haut niveau d'abstraction. Nous avons estimé qu'il était crucial d'en analyser l'expressivité ou, en d'autres termes, la possibilité de décrire les différents systèmes distribués qu'il est possible de rencontrer dans la littérature classique. Un premier pas dans ce sens a été fait dans [17] où sont présentés de nombreux systèmes ainsi que des prédicats de communication qui *correspondent* à chacun d'entre eux, en ce sens qu'ils *capturent* les propriétés de leurs communications. Bien que nombreux, seuls des systèmes à échanges de messages sont considérés dans [17]. Nous avons, dans le Chapitre 2, enrichi la *bibliothèque* des correspondances systèmes/prédicats.

Nous nous sommes tout d'abord penché sur une vaste gamme de systèmes à mémoire partagée sujets à des pannes bénignes. Nous avons présenté des prédicats de communication correspondant à des systèmes décrits par Herlihy [25] et Kruskal *et al.* [26], et avons porté un intérêt particulier aux systèmes SWMR et aux systèmes Atomic-Snapshot, qui ont fait l'objet d'études poussées. Dans [22], Gafni a défini le modèle RRFD, un modèle en rounds dans lequel, à chaque round, un *module* RRFD fournit à chaque processus p l'ensemble des processus dont p ne reçoit pas de message. Comme nous l'avons rappelé, le modèle RRFD et le modèle HO sont similaires sur plusieurs aspects. En particulier, tout module RRFD fournit, à chaque round r et pour chaque processus p, le complémentaire dans Π de l'ensemble d'écoute de p au round r. Cependant le médium de communication, qui n'est spécifié dans aucun des deux modèles, est supposé fiable dans le modèle RRFD, ce qui n'est pas le cas dans le modèle HO. Ainsi, tout module RRFD fournit, à chaque round, un ensemble de processus *suspects* alors que les prédicats de communication du modèle HO ne caractérisent que des propriétés vérifiées par les communications. Gafni a, lui aussi, analysé l'expressivité de son modèle et a, en particulier, décrit deux modules RRFD, dont les versions HO

sont les prédicats $\mathcal{P}_{nekrounds}$ et $\mathcal{P}^*_{\mathcal{RD}}$, qu'il prétend *correspondre naturellement* aux systèmes SWMR et aux systèmes Atomic-Snapshot, respectivement. La première partie du Chapitre 2 est en partie consacrée à la discussion de ces résultats, dont une version antérieure est présentée dans [24]. Alors que nous avons approuvé le choix de $\mathcal{P}^*_{\mathcal{RD}}$ pour les systèmes Atomic-Snapshot, nous avons démontré que $\mathcal{P}_{nekrounds}$ est trop faible pour correspondre aux systèmes SWMR, en ce sens qu'il existe un prédicat de communication strictement plus fort, selon l'ordre sur les prédicats défini dans le Chapitre 1, que nous avons appelé $\mathcal{P}_{sym}$, qui est la version HO d'un module lui aussi introduit par Gafni et qui peut être garanti dans de tels systèmes. Nous avons décrit une translation en 2 rounds de $\mathcal{P}_{sym}$ en $\mathcal{P}^*_{nekrounds}$, dont l'existence est évoquée dans [22] sans que la preuve en soit donnée. Enfin, nous avons démontré que $\mathcal{P}_{sym}$ et $\mathcal{P}^*_{\mathcal{RD}}$ sont équivalents et avons donc fourni une version HO de l'équivalence entre systèmes SWMR et systèmes Atomic-Snapshot sujets à des pannes bénignes [4, 20, 3].

La deuxième partie du Chapitre 2 est consacrée à la description de prédicats de communication correspondant à un grand nombre de systèmes à échanges de messages, sujets à des pannes bénignes aussi bien qu'à des erreurs de transmission par valeurs. Nous nous sommes, en particulier, intéressé à quatre des trente-deux systèmes étudiés par Dolev *et al.* [18] qui, selon la terminologie utilisée dans [18], sont les "cas minimaux" en ce qui concerne la résolubilité du problème du Consensus. Deux d'entre eux ont été décrits dans [17]. Pour chacun des deux autres, nous avons présenté un prédicat de communication ainsi qu'un algorithme, censé s'exécuter dans le-dit système, qui simule des collections d'ensembles d'écoute qui satisfont le prédicat en question.

La dernière partie de ce deuxième chapitre revient sur la caractérisation des prédicats de communication grâce auxquels il est possible de résoudre le Consensus [17]. En mettant en relation ce résultat avec certains des prédicats de communication présentés, nous avons été en mesure de retrouver des résultats relatifs à la résolubilité du Consensus rencontrés dans la littérature.

L'analyse et les résultats présentés dans les chapitres 1 et 2 montrent que le modèle HO est adapté pour l'étude des systèmes distribués sujets à des pannes bénignes, comme annoncé dans [17], aussi bien qu'à des erreurs de transmission par valeurs, mais aussi qu'il fournit un cadre d'étude unifié pour les deux grands types de médiums de communication considérés par la communauté de la Tolérance aux pannes, la mémoire partagée et les canaux de communication. La forte expressivité du modèle HO une fois établie, nous

nous sommes attaché, dans le Chapitre 3, à décrire des machines HO qui résolvent le Consensus en présence d'erreurs de transmission par valeurs. La démarche que nous avons adoptée à été de considérer des machines HO résolvant le Consensus dans un contexte de pannes bénignes et de les adapter de manière à tolérer des erreurs de transmission par valeurs.

Nous avons, tout d'abord, présenté trois algorithmes non-coordonnés. Deux d'entre eux, $\mathcal{A}_{T,E}$ et $\mathcal{U}_{T,M,E}$ [13], ont été obtenus en paramétrant les seuils de réception des algorithmes *OneThirdRule* et *UniformVoting* [17]. En faisant varier les valeurs des paramètres, on obtient différents algorithmes, plus ou moins résilients. Ainsi, une base unique paramétrée permet de décrire des algorithmes destinés à être exécutés dans différents systèmes. Il est, d'autre part, intéressant de remarquer que ces deux algorithmes permettent de contourner les bornes inférieures établies dans [42].

Nous avons ensuite adapté les algorithmes coordonnés *LastVoting* [17] et *DLS*. L'analyse qui a suivi a montré que la correction des deux algorithmes obtenus était fondée, en partie, sur l'hypothèse que le coordinateur d'une phase n'appartient pas à la partie altérée de cette phase, ce qui est capturé, par exemple, par le prédicat de communication $\mathcal{P}_{cc}$. Il apparaît alors que l'on ne peut se passer de supposer que le coordinateur d'une phase ne peut pas être à l'origine d'une transmission corrompue au cours de cette phase. Il est intéressant de remarquer que, pour les algorithmes coordonnés présents dans la littérature, les conditions assurant la sûreté des exécutions permettent toutes d'éviter qu'un message corrompu provenant d'un coordinateur soit pris en compte. Il semble donc que les schémas coordonnés ne soient pas réellement adaptés à la résolution du problème du Consensus en présence d'erreurs de transmission par valeurs.

Perspectives

Comme nous l'avons vu, le modèle HO permet de décrire tous les types de systèmes distribués et d'établir des résultats de résolubilité élégants. Cependant, la nature informelle de la notion de *correspondance* entre systèmes et prédicats de communication empêche l'établissement de résultats de non-résolubilité de tel problème dans tel type de systèmes ou de mettre en place une hiérarchie des systèmes classiques fondée sur l'ordre sur les prédicats. Il pourrait donc être intéressant de définir explicitement et rigoureusement ce qu'est le prédicat de communication correspondant à un système donné afin de hiérarchiser les différents types de systèmes classiques.

Dans [17], Charron-Bost et Schiper se sont attachés à caractériser les prédicats de communication grâce auxquels il est possible de simuler des rounds dont le noyau est non-vide et qui permettent de résoudre le problème du Consensus en présence de pannes bénignes. Il serait intéressant de voir s'il est possible d'étendre cette caractérisation à l'ensemble des prédicats de communication, ou encore de s'intéresser à d'autres problèmes célèbres comme le *k-Set Agreement*, ou le *Renommage*. Nous avons établi des résultats préliminaires concernant les deux premiers, qui doivent cependant être complétés et ne sont pas présentés ici.

Enfin, comme on a pu le voir, l'ordre sur les prédicats de communication est fondé sur la notion de *translation* qui n'a été définie que dans le cas des pannes bénignes. De nombreuses tentatives pour trouver une définition pertinente adaptée au cas des pannes par valeurs se sont révélées infructueuses et l'extension de l'ordre sur les prédicats de communication au cas général des prédicats faisant intervenir les ensembles d'écoute sûrs reste donc une question ouverte.

Algorithme $\mathcal{LR}$

Dans le Chapitre 4, nous nous sommes penché sur deux algorithmes de renversements de liens introduits par Gafni et Bertsekas [23], appelés *FR* et *PR*, pour le routage des messages dans des résaux mobiles sujets à de fréquents changements de topologie. Dans les implémentations de ces algorithmes proposées par Gafni et Bertsekas, les nœuds du réseau mettent à jour des variables appelées *hauteurs* à valeurs dans des ensembles non-bornés et totalement ordonnés.

Contributions

Nous avons présenté une nouvelle formalisation de ces deux algorithmes, qui conduit à des implémentations simples et bornées de *PR* et *FR* [14]. Nous avons décrit un algorithme unique, que nous avons appelé $\mathcal{LR}$ (pour *Link Reversal*), destiné à être exécuté sur des graphes acycliques munis d'un *étiquetage binaire des liens*, et avons déterminé des conditions sur l'étiquetage initial suffisantes pour garantir sa correction.

Cette formalisation nouvelle, qui unifie de manière simple et élégante *FR*

et PR et qui permet de décrire des algorithmes qui ne sont pas couverts par l'approche fondée sur des hauteurs, nous a permis de mener une analyse précise de la complexité en travail - le nombre de pas de calcul effectués par chacun des nœuds au cours d'une exécution - et d'améliorer de façon significative les résultats de Busch *et al.* [8, 9] en exhibant une formule exacte dépendant uniquement du graphe initial. En particulier, ce travail permet de comparer l'efficacité de FR et de PR selon la topologie du graphe d'entrée, et montre que, contrairement à l'intuition que l'on peut avoir, ni l'un ni l'autre n'est plus efficace, en ce sens qu'il existe des graphes pour lesquels FR est plus efficace que PR, et vice versa. Un autre point intéressant est que notre approche nous a permis d'étudier précisément quelle peut être l'influence de l'étiquetage initial sur la complexité en travail globale et de montrer que la minimisation de celle-ci nécessite une connaissance globale du graphe, et donc ne peut être aisément obtenue par une solution distribuée.

Nous avons, de plus, étudié la *complexité en temps* de cet algorithme dans le cas des graphes initiaux dont tous les liens sont marqués, c'est-à-dire le nombre d'itérations, au cours desquels chacun des puits effectue un pas de calcul, nécessaire pour qu'une exécution donnée converge. Nous avons établi un résultat exact pour la complexité en temps des exécutions sur des chaînes, dont nous avons déduit une borne inférieure sur la complexité en temps des exécutions sur des graphes quelconques.

Perspectives

Notre formalisation est fondée sur un étiquetage binaire des liens des graphes sur lesquels est exécuté notre algorithme. Il pourrait être intéressant de faire varier la taille k de l'espace des valeurs prises par les étiquettes des liens - un résultat préliminaire a été établi dans ce sens - et d'étudier dans quelle mesure cela permettrait de réduire le nombre de *renversements de liens*, à distinguer des *pas de calcul*, au cours d'une exécution donnée.

Une autre question pourrait consister à étendre le résultat concernant la complexité en temps de $\mathcal{LR}$ sur des chaînes initialement uniformément marquées, et donc de FR sur des chaînes, au cas de graphes plus complexes, ou encore d'établir un résultat similaire pour la complexité en temps de $\mathcal{LR}$ sur des graphes initialement uniformément non-marqués, et par conséquent de PR. On pourrait aussi s'intéresser à la détermination du round d'une

exécution de $\mathcal{LR}$ dans laquelle, à chaque round, tous les puits effectuent un pas de calcul, au cours duquel un nœud effectue son i-ième renversement.

Enfin, notre approche pourrait permettre de démontrer la correction d'autres algorithmes de renversements de liens comme l'algorithme de routage *TORA* [37] ou l'algorithme d'Élection de leader présenté dans [35], et de mener une analyse fine de leurs complexités respectives.

Bibliographie

[1] Ittai Abraham, Gregory V. Chockler, Idit Keidar, and Dahlia Malkhi. Byzantine disk paxos : optimal resilience with byzantine shared memory. In *PODC '04 : Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 226–235, New York, NY, USA, 2004. ACM.

[2] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Consensus with byzantine failures and little system synchrony. In *DSN '06 : Proceedings of the International Conference on Dependable Systems and Networks*, pages 147–155, Washington, DC, USA, 2006. IEEE Computer Society.

[3] Hagit Attiya and Ophir Rachman. Atomic snapshots in o (n log n) operations. *SIAM J. Comput.*, 27(2) :319–340, 1998.

[4] Hagit Attiya and Jennifer Welch. *Distributed Computing*. Wiley, 2004.

[5] V. C. Barbosa and E. Gafni. Concurrency in heavily loaded neighborhood constrained systems. *ACM Transactions on Programming Languages and Systems*, 11(4) :562–584, 1989.

[6] Michael Ben-Or. Another advantage of free choice (extended abstract) : Completely asynchronous agreement protocols. In *PODC '83 : Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, New York, NY, USA, 1983. ACM.

[7] Francisco V. Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal. Consensus in one communication step. In *PaCT '01 : Proceedings of the 6th International Conference on Parallel Computing Technologies*, pages 42–50, London, UK, 2001. Springer-Verlag.

[8] C. Busch, S. Surapaneni, and S. Tirthapura. Analysis of link reversal routing algorithms for mobile ad hoc networks. In *Proceedings of the*

15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pages 210–219, 2003.

[9] C. Busch and S. Tirthapura. Analysis of link reversal routing algorithms. *SIAM Journal on Computing*, 35(2) :305–326, 2005.

[10] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4) :398–461, 2002.

[11] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2) :225–267, 1996.

[12] K. M. Chandy and J. Misra. The drinking philosophers problem. *ACM Transactions on Programming Languages and Systems*, 6(4) :632–646, 1984.

[13] Bernadette Charron-Bost, Antoine Gaillard, Martin Biely, Martin Hutle, André Schiper, and Josef Widder. Tolerating corrupted communication. In *PODC '07 : Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 244–253, New York, NY, USA, 2007. ACM.

[14] Bernadette Charron-Bost, Antoine Gaillard, Jennifer L. Welch, and Josef Widder. Routing without ordering. (under submission), 2008.

[15] Bernadette Charron-Bost, Rachid Guerraoui, and André Schiper. Synchronous system and perfect failure detector : Solvability and efficiency issue. In *DSN*, pages 523–532, 2000.

[16] Bernadette Charron-Bost and André Schiper. Improving fast paxos : being optimistic with no overhead. In *PRDC*, pages 287–295, 2006.

[17] Bernadette Charron-Bost and André Schiper. The Heard-Of Model : Computing in Distributed Systems with Benign Failures. Technical report, 2007. Replaces TR-2006 : The Heard-Of Model : Unifying all Benign Failures.

[18] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1) :77–97, January 1987.

[19] C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2) :288–323, April 1988.

[20] Faith Fich. How hard is it to take a snapshot ? In M. Bielikova et al., editor, *Proceedings of 31st Annual Conference on Current Trends in*

Theory and Practice of Informatics (SOFSEM), volume 3381 of *Lecture Notes on Computer Science*, pages 28–37, 2005.

[21] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2) :374–382, April 1985.

[22] E. Gafni. Rounds-by-rounds fault detectors : unifying synchrony and asynchrony. In *Proceedings of the Seventeenth ACM Symposium on Principles of Distributed Computing*, pages 143–152, August 1998.

[23] E. Gafni and D. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, C-29(1) :11–18, 1981.

[24] Antoine Gaillard. Communication predicates for atomic registers shared-memory. (under submission), 2008.

[25] M. P. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1) :123–149, January 1991.

[26] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. Efficient synchronization of multiprocessors with shared memory. *ACM Trans. Program. Lang. Syst.*, 10(4) :579–601, 1988.

[27] Leslie Lamport. On interprocess communication. part ii : Algorithms. *Distributed Computing*, 1(2) :86–101, 1986.

[28] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2) :133–169, 1998.

[29] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2) :79–103, 2006.

[30] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3) :382–401, 1982.

[31] 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.

[32] Daniel Lehmann and Michael O. Rabin. On the advantages of free choice : a symmetric and fully distributed solution to the dining philosophers problem. In *POPL '81 : Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 133–138, New York, NY, USA, 1981. ACM.

[33] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[34] Y. Malka, S. Moran, and S. Zaks. A lower bound on the period length of a distributed scheduler. *Algorithmica*, 10 :383–398, 1993.

[35] N. Malpani, J. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL M)*, pages 966–103, 2000.

[36] Jean-Philippe Martin. Fast byzantine consensus. *IEEE Trans. Dependable Secur. Comput.*, 3(3) :202–215, 2006. Senior Member-Lorenzo Alvisi.

[37] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of the 16th IEEE Conference on Computer Communications (INFOCOM)*, pages 1405–1413, 1997.

[38] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2) :228–234, 1980.

[39] Fernando Pedone, André Schiper, Péter Urbán, and David Cavin. Solving agreement problems with weak ordering oracles. In *EDCC-4 : Proceedings of the 4th European Dependable Computing Conference on Dependable Computing*, pages 44–61, London, UK, 2002. Springer-Verlag.

[40] Michael O. Rabin. Theoretical impediments to artificial intelligence. In *IFIP Congress*, pages 615–619, 1974.

[41] K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1) :61–77, 1989.

[42] N. Santoro and P. Widmayer. Time is not a healer. In *Proceedings of the 6th Symposium on Theor. Aspects of Computer Science*, pages 304–313, Paderborn, Germany, 1989.

[43] U. Schmid, B. Weiss, and J. Rushby. Formally verified byzantine agreement in presence of link faults. In *ICDCS '02 : Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 608, Washington, DC, USA, 2002. IEEE Computer Society.

[44] T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2) :80–94, 1987.

[45] J. Walter, G. Cao, and M. Mohanty. A k-mutual exclusion algorithm for wireless ad hoc networks. In *Proceedings of the ACM Workshop on Principles of Mobile Computing (POMC)*, 2001.

[46] J. Walter, J. L. Welch, and N. Vaidya. A mutual exclusion algorithm for ad hoc mobile networks. *Wireless Networks*, 7(6) :585–600, 2001.

[47] Piotr Zieliński. Paxos at war. Technical Report UCAM-CL-TR-593, University of Cambridge, June 2004.

Table des matières

1 Le modèle Heard-Of (HO) 9
1.1 Ensembles d'écoute et prédicats 10
1.1.1 Ensembles d'écoute 10
1.1.2 Prédicats de communication 13
1.2 Machines HO et problèmes 14
1.2.1 Machines HO 14
1.2.2 Problème. Résolubilité d'un problème. 15
1.2.3 Machines HO coordonnées 18
1.3 Translations dans le cas bénin 20
1.3.1 Translations uniformes 21
1.3.2 Translations générales 22

2 Systèmes et prédicats 23
2.1 Mémoire partagée dans le cas bénin 24
2.1.1 Registres SWMR et objets Atomic-Snapshot 24
2.1.2 Autres objets partagés 54
2.2 Systèmes à échanges de messages 63
2.2.1 Pannes bénignes 63
2.2.2 Pannes par valeurs 81
2.3 Prédicats pour le Consensus : cas bénin 86
2.3.1 Caractérisation des machines HO qui résolvent Consensus 87
2.3.2 Consensus et systèmes à mémoire partagée 88
2.3.3 Consensus et systèmes à échanges de messages 91

3 Algorithmes de Consensus 93
3.1 Communications plus vivaces 94
3.1.1 Algorithme *UniformVoting* 94

3.1.2 Algorithme $\mathcal{U}_{T,M,E}$ 96
3.1.3 Algorithme $\mathcal{UV}_{vf}$ 107
3.2 Communications plus sûres 114
3.2.1 Algorithme *OneThirdRule* 114
3.2.2 Algorithme $\mathcal{A}_{T,E}$ 115
3.3 Algorithmes uniformément coordonnés 123
3.3.1 Algorithme *DLS* 124
3.3.2 Algorithme DLS_{vf} 125
3.4 Algorithmes coordonnés 139
3.4.1 Algorithme *LastVoting* 140
3.4.2 Algorithme $\mathcal{LV}_{vf}$ 141
3.5 Discussion 150
3.5.1 Correspondance avec les bornes inférieures 151
3.5.2 Sur le recours à des coordinateurs 152

4 Algorithme $\mathcal{LR}$ 155
4.1 Motivations 155
4.2 Orientation vers la destination 160
4.2.1 Notations 160
4.2.2 L'approche de Gafni et Bertsekas 161
4.3 L'algorithme $\mathcal{LR}$ 165
4.3.1 Convergence et insensibilité au délai 168
4.3.2 La question de l'acyclicité 170
4.4 Complexité 173
4.4.1 Complexité en travail 173
4.4.2 Complexité en temps : résultat partiel 184
4.4.3 Cas particuliers de *FR* et *PR* 196
4.4.4 Comparaison avec les résultats de Busch *et al.* 201