ÉCOLE POLYTECHNIQUE

Laboratoire d'Informatique
de l'École Polytechnique

École Doctorale de l'École Polytechnique

THÈSE DE DOCTORAT

par **Jonas LEFÈVRE**

soutenue le **9 décembre 2014**

Pour obtenir le grade de **Docteur de l'École Polytechnique**
Discipline : **Informatique**

Protocoles de Population : une hiérarchie et des variantes. Calcul de couplages autostabilisants.

Président du Jury

M. Beauquier Joffroy	*Professeur, Université Paris-Sud*

Directeur de thèse

M. Bournez Olivier	*Professeur, École Polytechnique*

Rapporteurs

M. Durand-Lose Jérôme	*Professeur, Université d'Orléans*
Mme Potop-Butucaru Maria	*Professeur, Université de Paris VI*

Examinateurs

M. Cervelle Julien	*Professeur, Université de Paris-Est*
M. Chalopin Jérémie	*Chargé de Recherche CNRS, LIF Marseille.*
M. Turau Volker	*Professeur, Hamburg Universtity of Technology .*

Invitée

Mme Pilard Laurence	*Maître de Conférence, Université de Versailles*

Résumé

Le modèle des protocoles de population a été introduit par Angluin *et al.* en 2004 comme un modèle abstrait de réseaux de capteurs avec des interactions par paire. Le modèle, dans sa version initiale, a été caractérisé précisément et ne peut reconnaître que des ensembles semi-linéaires. Cependant, si le graphe de communication est assez restreint (*i.e.* si le degré des noeuds est borné par une constante), il est alors possible de simuler des machines de Turing non-déterministes utilisant un espace linéaire.

Dans cette thèse, j'ai travaillé avec Olivier Bournez et nous avons introduit un nouveau formalisme pour définir des protocoles avec environnement. Ce formalisme nous a permis de porposer une nouvelle construction pour la simulation d'une Machine de Turing dans le cas où le graphe de communication est à degré borné. En utilisant ce formalisme, nous avons pu étendre la construction aux cas où un sous-graphe est à degré borné. De cette manière, nous avons obtenu la simulation de Machines de Turing non-déterministes utilisant un espace sous-linéaire. Nous avons ainsi étendu certaines hiérarchies existantes. Par exemple, Chatzigiannakis *et al.*. ont démontré que, si l'espace de calcul alloué à chaque agent est en $o(\log \log n)$ (où n est la taille de la population), et si le graphe de communication est complet, le modèle obtenu n'est pas plus puissant que le modèle classique. Cependant notre construction permet d'utiliser ce petit espace disponible sur chaque agent si le graphe est assez restreint, mais pas nécessairement à degré borné. Nous avons également étudié des variantes du modèle initial et caractérisé leur puissance. Par exemple, avec Olivier Bournez et Mikaël Rabie, nous avons étudié la variante où deux agents ne peuvent communiquer que s'ils ne sont pas d'accord sur la sortie, ou plus généralement s'ils n'ont pas la même opinion, plusieurs opinions étant possibles pour une même sortie. Nous avons caractérisé l'ensemble des prédicats reconnus par un tel modèle : c'est l'ensemble des prédicats semilinéaires qui peuvent s'écrire comme ne dépendant pas du nombre d'occurrence de chaque symbole mais de leur fréquence (le nombre d'occurrence divisé par la taille de la population). Ainsi un prédicat de type modulo (par exemple $x = y \mod 5$) ou un prédicat de type seuil avec une valeur de seuil non-nulle (par exemple $x + y > 7$) ne peuvent être reconnus par un protocole de

notre modèle.

Parallèlement, j'ai travaillé avec Johanne Cohen (LRI), Laurence Pilard (PRISM) et Khaled Maâmra sur un autre aspect du calcul distribué : les algorithmes autostabilisants. Nous avons proposé des solutions au du problème couplage distribué. À partir d'un algorithme de Manne *et al.*, nous avons construit un algorithme probabiliste autostabilisant qui calcule un couplage maximal dans un réseau distribué anonyme. Un couplage maximal est $\frac{1}{2}$-approximation d'un couplage maximum. Sous le démon adversaire, notre algorithme se stabilise en $O(n^3)$ pas de calcul avec probabilité très proche de 1, où n est le nombre de noeuds du réseau. L'algorithme original de Manne *et al.* brisait la symétrie entre les noeuds en utilisant les identifiants ; pour que l'algorithme fonctionne sur un réseau anonyme, nous avons utilisé un comportement probabiliste pour briser la symétrie : les noeuds choisissent un partenaire au hasard et ils peuvent aléatoirement décider de ne rien faire. De cette façon, nous pouvons garantir la correction de l'algorithme et un temps de convergence qui reste faible. Dans un second temps, nous nous sommes intéressés à un algorithme de Manne *et al.* qui calcule une $\frac{2}{3}$-approximation d'un couplage maximum. En exhibant une exécution de taille $2^{\sqrt{n}}$, où n est la taille de la population, nous avons prouvé que la complexité de cet algorithme était au moins subexponentielle.

Abstract

The computational model of population protocols was introduced by Angluin *et al.* in 2004 as an abstract model of sensors network with pairwise interactions. The initial version of this model was fully characterized : the computable languages correspond exactly to the semi-linear sets. However, by changing the communication graph (namely if the degree of the vertices is bounded by some constant), it becomes possible to model a linear-space non-deterministic Turing machine. I worked with Olivier Bournez and we introduced a new formalism to define population protocols with environment. This formalism allowed us to propose a new construction for the simulation of a Turing machine by a population protocol working on a restricted communication graph. Using this formalism, we could model Turing machines using sublinear space and then we could extend existing hierarchies. For instance, Chatzigiannakis *et al.* showed that if each agents has a memory space of size $o(\log \log n)$ (where n denotes the size of the population) and if the graph is complete, then the resulting model is not more powerful than the classical population protocols. However our construction can use this small available space on the agents if the communication graph is restricted enough, but not necessarily degree-bounded. We also have studied some variants of the original model and characterized their computational power. As an example, with Olivier bournez and Mikael Rabie, we studied the variant where two agents can interact only if they disagree on the output or, more generally, do not have the same opinion, several opinions being possible for one output. We characterized the set of predicates computed by this model : it is exactly the semi-linear predicates over the frequency of each input letter, and not the number of occurrences as usual. Thus the modulo-based predicates (as $x = y \mod 5$) or the threshold predicates with non-zero constant (such as $3x - 2y > 7$) can not be computed by our model.

In the same time, I worked with Johanne Cohen (LRI), Laurence Pilard (PRISM) and Khaled Maâmra on an other topic related to the distributed computation : self-stabilizing algorithms for distributed matching. From an algorithm of Manne *et al.*, we build a randomized self-stabilizing algorithm computing a maximal matching

(that is a $\frac{1}{2}$-approximation of a maximum matching) in an anonymous network. Under the adversarial daemon, our algorithm stabilizes in $O(n^3)$ moves with high probability (as close to 1 as wished). The alglorithm of Manne *et al.* breaks the symmetry between the nodes by using their identifier. As our algorithm works in an anonymous networks, there is no identifier. Our algorithm use a probabilistic behavior to break the symmetry : a node randomly choose an other node, but it can also choose nobody with probability $\frac{1}{2}$. By this way, we can guarantee the correctness of the algorithm and a good convergence time. In a second time, we were interested in an algorithm of Manne *et al.* that computes a $\frac{2}{3}$-approximation of a maximum matching. We construct an execution of size $2^{\sqrt{n}}$, with n being the size of the graph. We thus proved that the complexity of this algorithm is not polynomial, but at least sub-exponential.

Remerciements

Ce manuscrit est le résultat le plus évident de mes années de thèses. Cependant ces années que j'ai passées en tant que thésard m'ont apporté bien plus.

Je voudrais tout d'abord remercier Olivier pour m'avoir proposé un sujet de thèse qui ne portait au début que sur les protocoles de populations. Commencée lors de mon stage de M2, notre collaboration s'est étendue sur 4 ans et demi. J'ai beaucoup appris à ses cotés, d'un point de vue scientifique bien sûr mais aussi sur la gestion d'emplois du temps et les contraintes administratives associées aux responsabilités. Mon sujet de thèse a évolué pour s'étendre à l'étude d'algorithmes calculant des couplages de manière autostabilisante et distribuée. Je remercie donc Johanne, Laurence et Khaled avec qui j'ai travaillé sur cette thématique. J'ai pu ainsi élargir mon champs d'investigation, explorer d'autres techniques et m'entraîner à d'autres outils. Et travailler avec différentes personnes, c'est travailler avec autant de méthodes et d'approches différentes. J'ai également collaboré avec Mikaël. Nous avons travaillé en toute confiance ensemble sur une variante des protocoles de population.

Tout ces travaux scientifiques ont été enrichis par de nombreux échanges. À ce titre, je voudrais remerciés tout ceux qui se sont penchés un jour sur un article que j'écrivais (et qui m'ont souligné mes fautes), tout ceux qui se sont penchés un jour sur une question que je me posais, quitte à perdre une après-midi devant un tableau blanc, tout ceux avec qui j'ai échangé dans un bureau, dans un couloir ou devant un café sur mon sujet d'étude, sur le leur ou sur un autre.

La vie d'un laboratoire n'est pas composée que de recherche et tout ces heures passées sur le plateau n'auraient pas été les mêmes sans la bonne compagnie qu'on y trouve. Je remercie donc tout ceux avec qui j'ai partagé mon bureau, dans l'aile 0 ou dans le "TuTu", tout ceux avec qui j'ai partagé des parties de cartes, coinche, wizard ou autres, tout ceux avec qui j'ai pu discuter sur tout et surtout n'importe quoi, tout ceux qui m'ont tenu compagnie pendant ces 4 ans et demi.

Ces années de thèse ont parfois été un peu longues et même un peu difficiles. Tout ne marche pas tout de suite du premier coup. Mais j'ai toujours pu trouver quelqu'un autour de moi pour prendre du recul et penser à autre chose. Je remercie

donc tout mes amis et mes proches, ceux qui m'ont permis de penser à autre chose et de m'aérer l'esprit, ceux qui se sont intéresser à mes problèmes et à leurs solutions. Notamment merci à tout ceux du Raton-Laveur pour les quelques dizaines de milliers de mails.

Tout au long de mes études, et donc aussi durant ma thèse, j'ai toujours pu compter sur le soutien de ma famille. Même s'ils ne comprenaient pas vraiment ce que je faisais, ce sur quoi je travaillais, mes parents, mes soeurs, mes grands-parents et toute le reste de ma famille m'ont toujours soutenu et je leur suis reconnaissant.

Et finalement, je voudrais remercier la personne qui m'a soutenu tout au long de cette aventure, chaque jour. Grâce à elle, la grisaille parisienne était un peu moins grise, et la banlieue plus supportable. Pour beaucoup de raisons et plus encore, merci à elle.

Table des matières

Chapitre 1

Introduction

Cette thèse porte sur deux aspects du calcul distribué. D'un côté, je présente une étude liée aux protocoles de population, qui sont un modèle de calcul distribué. De l'autre, je présente des résultats sur des algorithmes autostabilisants calculant des approximations de couplage maximum dans un réseau.

Contexte

En 1936, Turing introduit les *Machines de Turing* [Tur36] afin de définir formellement ce qu'est un algorithme. En utilisant ce modèle de calcul, il put définir les *nombres calculables* et apporter une réponse, négative, au Problème de Décision. D'autres modèles de calculs ont été introduits à cette époque et plus tard, comme le λ-calcul ou les grammaires formelles. Les machines de Turing sont un modèle de calcul pouvant être qualifié de *centralisé*, c'est-à-dire qu'il existe un unique centre de calcul où toutes les opérations sont exécutées linéairement, l'une après l'autre.

Avec les premiers grands réseaux d'ordinateurs, la notion de calcul distribué émerge dans les années 1970. Le développement de réseaux à l'échelle mondiale, où des ordinateurs peuvent échanger des messages les uns avec les autres, nourrissent le développement de modèles de calcul distribué où une multitude de centres de calcul échangent des messages selon un programme interne. Dans un cadre distribué, il n'est en général pas possible de rassembler toute l'information en un point pour effectuer l'ensemble du calcul. Cette contrainte peut venir d'un coût élevé en temps de communication et de calcul, mais aussi de contraintes physiques, comme la taille de la mémoire d'un centre de calcul.

Les développements technologiques récents renforcent l'intérêt de l'étude du calcul distribué. Les réseaux comportent de plus en plus d'unités de calcul, de plus en plus

d'appareils sont équipés d'unités de calcul performantes (téléphone, tablette, voiture, lunette,...), les énormes calculateurs cèdent le pas aux grappes de calcul composées de centaines voire de milliers d'ordinateurs, et les ordinateurs eux-mêmes comportent maintenant de plus en plus de processeurs.

Les modèles de calculs distribués possèdent une diversité reflétant la diversité des systèmes distribués rencontrés en pratique. Selon les cas, les agents sont mobiles ou non, ce qui peut générer un réseau dynamique, où des liens de communication apparaissent ou disparaissent au cours du temps. Selon les cas, le graphe de communication sous-jacent au réseau peut être fixé et avoir certaines propriétés spécifiques. Les communications entre les unités de calculs peuvent être synchronisées ou non, être à sens unique ou non, et il peut être possible de négliger le temps pour un message de partir et d'arriver. De plus, les unités de calculs du réseau peuvent avoir aussi une grande variété de propriétés différentes.

À cause de cette diversité, il n'est pas possible de construire un modèle générique de calcul distribué, un modèle qui jouerait le rôle des machines de Turing pour le calcul centralisé. Les modèles de calcul distribué sont souvent incomparables entre eux.

Au sein de cette diversité, nous nous sommes intéressés à deux cas particuliers reflétant chacun un aspect différent de cette thématique. Nous avons étudié des modèles de calcul distribué avec les protocoles de population et certaines de ses variantes. Et nous avons étudié des algorithmes distribués calculant des approximations de couplage maximum dans un réseau.

Contributions

La première partie de ma thèse est consacrée aux protocoles de population. Le travail présenté dans cette partie a été réalisé en collaboration avec Olivier Bournez, et Mikaël Rabie pour un chapitre.

Dans le **chapitre 2** , nous définissons le modèle des protocoles de population tel qu'il a été introduit par Angluin *et al.* en 2004. Ce chapitre est consacré à une étude bibliographique. Nous présentons les principaux résultats sur les protocoles de population. Nous décrivons également les variantes les plus étudiées, ainsi que les résultats sur la puissance de calcul de ces variantes. Finalement, nous présentons différents modèles de calcul distribué qui peuvent être reliés, d'une manière ou d'une autre, aux protocoles de population ou à l'une de leurs variantes.

Le **chapitre 3** est consacré à la preuve de l'équivalence entre les protocoles de population avec un graphe d'interaction restreint et la classe des machines de Turing non-déterministes utilisant un espace linéaire et reconnaissant uniquement

des langages symétriques. Nous présentons la preuve de ce résultat. Le résultat est présent dans la bibliographie, cependant la construction est répartie entre plusieurs articles, et certaines parties de la preuve ne semblent pas complètes. Après avoir exposé l'approche utilisée dans la littérature, nous présentons notre approche de ce problème. Nous avons défini un formalisme permettant de facilement composer les protocoles entre eux. De cette manière, nous avons obtenu une construction d'une simulation de machine de Turing sur un protocole de population calculant sur un graphe à degré borné, construction qui nous semble plus modulable.

Dans le **chapitre 4**, nous construisons une hiérarchie de protocoles de population. Nous construisons toute une hiérarchie entre les ensembles semi-linéaires, reconnus par les protocoles de population classiques, et les classes de la complexité en espace des machines de Turing. Cette hiérarchie est obtenue en considérant des familles de graphes intermédiaires entre la famille des chaînes et celle des cliques. Il existe dans la littérature un certain nombre de hiérarchies s'appuyant sur telle ou telle variante. Pour construire la nôtre, nous avons réutilisé notre construction du chapitre 2 et mis à profit sa modularité. Finalement, nous obtenons une hiérarchie qui contient les hiérarchies existantes et qui est de plus capable de distinguer plus de niveaux de puissance de calcul.

Les variantes montrées et étudiées dans les premiers chapitres sont au moins aussi puissantes que le modèle initial, et elles permettent de construire des hiérarchies entre le modèle initial et certaines classes de machines de Turing. Dans les chapitres 5 et 6, nous étudions différentes variantes qui sont plus faibles que le modèle initial.

Dans le **chapitre 5**, nous étudions une série de variantes qui porte sur la forme des règles de transition et le mode d'acceptation. Nous comparons les transitions classiques (il y a deux agents avant et après l'interaction) à des transitions génératrices (il y a deux agents avant l'interaction et un nombre quelconque, mais fixé pour chaque règle, après) et à des transitions de type temps-réel (il y a deux agents avant l'interaction et un seul après). Les classes de prédicats reconnus par les variantes étudiées dans ce chapitre ne sont en général pas stables par les opérations booléennes. Elles ont comme point commun de toutes contennir l'ensemble des prédicats modulo. Ces classes contiennent également certains prédicats de seuil selon la variante précise qui est considérée.

Dans le **chapitre 6**, nous étudions des variantes où en plus de leur état et de leur sortie, les agents ont des opinions, qui ne dépendent que de l'état. Nous avons étudié les cas où l'opinion d'un agent correspond à sa sortie. Dans les variantes considérées, les agents prennent en compte cette opinion pour interagir : dans une des variantes, par exemple, deux agents avec la même opinion ne peuvent pas interagir. Les deux variantes considérées se trouvent avoir la même expressivité. La classe des

langages reconnus par ces variantes est un sous-ensemble des langages semi-linéaires ne comprenant pas les prédicats modulo mais seulement les prédicats de seuil nul. Nous décrivons de différentes manières les langages reconnus par ces variantes.

La seconde partie traite du problème du couplage distribué dans un contexte autostabilisant.

Cette partie relate le travail réalisé en collaboration avec Johanne Cohen, Laurence Pilard et Khaled Maâmra.

Dans le **chapitre 7**, qui est un chapitre bibliographique, nous introduisons la problématique du couplage en tant que problème de théorie des graphes. Nous rappelons notamment le résultat selon lequel un couplage maximal est une $\frac{1}{2}$-approximation d'un couplage maximum. Nous donnons également une condition nécessaire bien connue dans la littérature pour qu'un couplage soit une $\frac{2}{3}$-approximation. Dans la suite du chapitre, nous introduisons les notions d'algorithmique distribuée que nous utiliserons dans les chapitres suivants. Nous définissons deux modèles de communication et plusieurs démons. Nous discutons aussi de la mesure de la complexité d'un algorithme distribué.

Au **chapitre 8**, nous construisons un algorithme autostabilisant anonyme en nous basant sur un algorithme existant de Manne *et al.* [MMPT09]. Notre algorithme, comme l'original, calcule un couplage maximal, c'est-à-dire une $\frac{1}{2}$-approximation d'un couplage maximum. Cependant notre algorithme n'utilise plus les identifiants afin de briser la symétrie entre les nœuds du réseau mais plutôt des bits aléatoires. Nous obtenons de cette manière un algorithme distribué qui se stabilise en $O(n^3)$ pas de calcul avec forte probabilité. Nous complétons l'étude de la complexité de notre algorithme en calculant une meilleure borne, en $O(n^2)$ pas de calcul avec forte probabilité, sur la complexité de notre algorithme dans un cas particulier. L'**appendice A** complète ce chapitre en explicitant la manière que nous utilisons pour passer du modèle à numérotation des ports au modèle à état, et ce afin de bien respecter l'hypothèse d'anonymat.

Au **chapitre 9**, nous étudions la complexité d'un autre algorithme de Manne *et al.* [MMPT08]. Cet algorithme calcule une $\frac{2}{3}$-approximation d'un couplage maximum. La complexité de cet algorithme avait déjà été étudiée [MMPT08] mais seule une borne supérieure exponentielle avait été démontrée. Nous exhibons une exécution non polynomiale et prouvons ainsi que l'on ne peut espérer une meilleure borne sur la complexité de cette algorithme qu'une borne sous-exponentielle.

Première partie

Protocoles de Population

Chapitre 2

Protocoles de Population

Nous introduisons dans ce chapitre le modèle des *Protocoles de Population*, ainsi que certaines variantes. Nous donnons les principaux résultats disponibles sur les protocoles de population et sur les principales variantes. Nous faisons également le lien entre ces modèles et d'autres modèles de calculs de la littérature.

2.1 Définition du modèle classique

Les *systèmes distribués* ou *systèmes répartis* sont étudiés sous de nombreux aspects et il existe une grande quantité de différents modèles pour les étudier. Un système distribué est décrit via les agents qui le composent. L'évolution du système est le résultat de ses agents et de leurs interactions. Mais les propriétés du système sont généralement plus que la simple somme des propriétés de ses agents : toutes les propriétés du système sont héritées des propriétés des agents et de leurs interactions, mais il peut y avoir émergence de propriétés non présentes au niveau des agents qui découlent d'une manière ou d'une autre des propriétés fondamentales du système sur les agents et les interactions. En étudiant comment varie un système distribué lorsque certaines propriétés des agents et des communications varient, il est possible de comprendre l'origine des propriétés émergentes.

Les *Protocoles de Population* sont un modèle de calcul distribué utilisant des agents anonymes passivement mobiles et avec une puissance de calcul faible . Il a été introduit par Angluin, Aspnes, Diamadi, Fischer et Peralta en 2004 [AAD$^+$04].

Dans ce modèle, les agents sont mobiles au sein d'une population — une population est un ensemble d'au moins deux agents — mais leurs mouvements sont supposés imprévisibles. Deux agents communiquent ensemble lorsque, suite à leur mouvement, ils passent assez proche l'un de l'autre. La communication est asymétrique : il y a

un agent initiateur de la communication et un agent récepteur. Comme il n'y a pas de contrôle sur leur mouvement, il n'est pas possible de prédire quel agent communiquera avec quel agent et à quel moment. Par ailleurs, les agents ont une mémoire bornée par une constante qui ne dépend que du programme. En particulier, cette constante est indépendante de la taille de la population : chaque agent de la population est un automate fini, et tout les agents ont le même programme à exécuter. Cette contrainte empêche par exemple la construction d'un identifiant unique pour chacun des agents. Deux agents dans le même état (ayant la même configuration de mémoire) sont donc totalement indistinguables par un tiers.

Pour les définir formellement, il faut : un alphabet d'entrée Σ et un alphabet de sortie Y, un ensemble fini d'états Q pour les agents, des fonctions qui traduisent un symbole d'entrée en état et un état en symbole de sortie, et une relation de transition.

Définition 1 (Protocole de Population, d'après [AAD+04]) *Un* protocole de population

$$(Q, \Sigma, in, out, \delta)$$

(parfois abrégé en (Q, δ) dans la suite) est formellement définit par :

- *Q, un ensemble fini d'*états*, qui correspond aux états possibles pour un agent ;*
- *Σ, un alphabet fini d'entrée et Y, un alphabet fini de sortie ;*
- *in, une fonction d'entrée allant de Σ dans Q, où $in(\sigma)$ représente l'état initial d'un agent dont l'entrée est σ ;*
- *out, une fonction de sortie de Q dans Y, où $out(q)$ représente la valeur de sortie d'un agent dans l'état q ;*
- *$\delta \subseteq Q^2 \times Q^2$, une relation transition qui décrit comment des paires d'agents peuvent interagir. Nous écrirons souvent $(s_1, s_2) \to (r_1, r_2)$ ou $s_1\ s_2 \to r_1\ r_2$ au lieu de $((s_1, s_2), (r_1, r_2)) \in \delta$.*

Dans l'article originel [AAD+04] et les suivants [AAC+05, AAD+06, AAER07], les transition possibles sont représentées par une fonction, et non une relation. Pour plus de généralité, nous utilisons une relation de transition. Nous préciserons dans la section 2.2.3 ce que ce changement implique plus précisément.

Pour une paire d'agents qui interagit, il n'y a *a priori* pas unicité de la règle qui peut être appliquée. Si, pour chaque paire d'agents, il y a unicité, alors le protocole de population est dit *déterministe*. Cependant, cela ne modifie pas fondamentalement le modèle de calcul en terme d'expressivité ou de puissance [AR07].

Les communications sont supposées instantanées, simultanées et asymétrique. L'asymétrie est codée par la distinction qui est faite par définition entre le premier et le second élément d'un n-uplet de la relation.

Sauf mention contraire, nous supposerons que l'alphabet de sortie est réduit à $Y = \{true, false\}$. Nous nous limitons ainsi aux protocoles de population vus comme des reconnaisseurs de langages sur l'alphabet Σ et aux problèmes de décisions du type "est-ce que le mot ω appartient au langage $\mathcal{L}$?" ; et nous ne regarderons pas les problèmes avec des sorties multi-valuées comme par exemple "Quelle est le nombre modulo 3 de lettre σ dans le mot ω ?". Il est possible de passer d'une catégorie de problème à l'autre. Nous ne perdons ainsi aucune généralité, tout en gagnant en simplicité.

À chaque instant, la *configuration* d'un système distribué décrit les états des agents et des liens de communications. Dans le modèle des protocoles de population, les communications sont immédiates, il n'y a donc pas de messages en attente dans les liens de communications. Une *configuration* C d'un protocole de population est la donnée pour chaque agent de son état. Cependant comme deux agents ayant le même état sont indistinguables par un tiers, il est possible de représenter une configuration par un multi-ensemble d'états. La configuration de la population peut aussi être vue comme une fonction qui compte les occurrences de chaque état dans la population. Plus précisément, une configuration C sera indifféremment vue comme un multi-ensemble de Q ou une fonction de Q dans $\mathbb{N}$: pour tout état $q \in Q$, il y a $C(q)$ agents dans l'état q. Nous utiliserons aussi une configuration comme une fonction de Q dans $\mathbb{N}$ comme un vecteur de $\mathbb{N}^Q$. L'ensemble des configurations sera noté $\mathcal{C}$. Il sera muni de l'ordre partiel composante par composante canonique noté $\leq$.

Nous étendons la définition de la fonction d'entrée aux configurations de la manière suivante. Soit $C_{in} \in \Sigma^*$ et $C_0 \in \mathcal{C}$. Nous avons $in(C_{in}) = C_0$ si pour toute lettre $\sigma \in \Sigma$ nous avons $C_0(in(\sigma))$ occurrences de la lettre σ dans C_{in}. Les entrées sont donc des mots écrits sur l'alphabet Σ. Nous définissons ici la convention pour la lecture des entrées entières : un d-uplet d'entiers naturels $(x_1, \ldots, x_d)$ sera encodé, sur un alphabet Σ contenant au moins les d lettres $a_1, \ldots, a_d$, par le mot d'entrée $a_1^{x_1}, \ldots, a_d^{x_d}$. Cette convention sera celle utilisée à chaque fois que nous considérerons des protocoles de population travaillant sur des entiers en entrée.

Une *exécution* d'un système distribué est une suite alternée finie ou infinie de configurations et de transition. Les *transitions* sont une ou plusieurs actions ou interactions d'un ou plusieurs agents.

Pour un protocole de population, une transition est l'application d'une règle par une paire d'agents pouvant appliquer une règle. Formellement, la transition C de à C' peut avoir lieu si il existe deux agents respectivement dans l'état q_1 et q_2 dans C et dans l'état q_1' et q_2' dans C', et s'il existe une règle $((q_1, q_2), (q_1', q_2')) \in \delta$, et si tous les autres agents sont dans le même état dans C et dans C'. Dans ce cas nous noterons $C \to C'$. Nous noterons $\cdot \to^* \cdot$ la fermeture transitive de $\to$, c'est-à-dire

que $C \to^* C'$ s'il existe une suite de k configurations $C_1, \ldots, C_k$ telle que $C_1 = C$, $C_k = C'$ et, pour tout $1 \leq i < k$, $C_i \to C_{i+1}$.

Une *exécution* d'un protocole de population est donc une suite finie ou infinie de configurations $\mathcal{E} = C_0 C_1 \ldots C_k C_{k+1} \ldots$ telle que, pour tout k, on ait $C_k \to C_{k+1}$.

Considérer toutes les exécutions possibles d'un système distribué n'est en général pas souhaitable, cela revient souvent à considérer des exécutions jugées non acceptables. Il existe plusieurs manières de palier à ce problème : se limiter à certaines exécutions qui vérifient une certaine propriété d'équité par exemple, ou bien adopter un comportement probabiliste qui augmentera la probabilité de certaines exécutions et diminuera (voire annulera) la probabilité d'autres exécutions.

Pour le modèle classique des protocoles de populations, le choix qui a été fait est de considérer que l'ensemble des exécutions acceptables est contraint par une propriété d'équité. Il y existe des variantes où cette propriété d'équité est remplacée par un comportement probabiliste, nous en présentons certaines à la section 2.4.3.

Définition 2 (Équité) *Une exécution $\mathcal{E} = C_0 C_1 \ldots C_k \ldots$ est* équitable *si pour toute configuration C apparaissant une infinité de fois (i.e. il existe une infinité d'indices k tel que $C = C_k$) et pour tout configuration C' telle que $C \to C'$ alors la configuration C' apparaît infiniment souvent.*

De manière équivalente, une exécution est équitable si toute configuration infiniment souvent accessible (en une ou plusieurs application de règle) est infiniment souvent présente dans l'exécution. La propriété d'équité signifie que ce qui est infiniment souvent possible arrivera infiniment souvent.

Cette propriété d'équité a été définie de manière à mimer d'une certaine manière un comportement probabiliste de la population, cependant l'évolution d'une population n'est pas probabiliste mais bien non-déterministe : un ordonnanceur externe au système choisit à chaque pas quelle paire d'agents interagit, cet ordonnanceur est totalement libre dans son choix, la suite infinie de ses choix doit seulement être telle que l'exécution soit équitable. Il peut être montré que les configurations atteintes infiniment souvent dans une exécution équitable sont les configurations qui seraient atteinte avec probabilité non nulle (et même avec probabilité 1) si le choix de la prochaine paire d'agents à interagir se faisait selon une loi de probabilité uniforme.

Il est important de noter que cette propriété d'équité n'implique rien sur le comportement de la population durant une exécution finie. L'équité ne contraint l'évolution de la population que sur son comportement "limite". Par ailleurs cette propriété d'équité peut ressembler à la propriété "si deux agents peuvent infiniment souvent interagir ensemble alors ils interagiront ensemble infiniment souvent", or il n'en est

rien. Il n'y a même aucune implication entre les deux propriétés. La propriété d'équité est une condition sur les configurations et ne dit rien sur les interactions entre agents.

Considérons l'exemple donné dans la figure 2.1. En ne suivant que les flèches en trait plein et en tournant une infinité de fois sur le cycle, nous obtenons une exécution infinie dans laquelle tout les couples d'agents interagissent une infinité de fois (à chaque étape le couple qui interagit est marqué par des cercles en gras). Cependant cette exécution n'est pas équitable, en effet la configuration composée d'un agent hachuré et de deux agents "1" est infiniment souvent accessible en une application de règle (via la flèche en pointillés). Inversement l'exécution qui consiste à aller de la configuration initiale à cette configuration en 3 pas de calcul et ensuite à boucler dessus en activant toujours la même paire d'agent est une exécution équitable puisque toute configuration infiniment souvent accessible est atteinte infiniment souvent, par contre une seule paire d'agent interagit infiniment souvent.

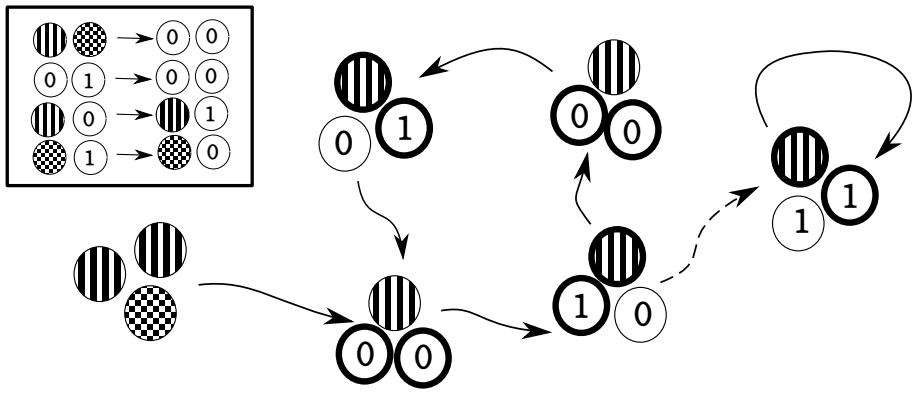


FIGURE 2.1 – Exemple de possibilités d'exécution pour un protocole ;
la relation de transition est composée des règles représentées en haut et complétée par l'identité dans les autres cas

Nous ne considérerons que cette notion d'équité. Dans la littérature, il existe d'autres définition d'équités, des équités globales, des équités locales, des équités fortes et des équités faibles. Des études plus détaillées sont disponibles dans [FJ06].

La fonction de sortie *out* associe chaque état une valeur de sortie. Nous étendons cette fonction aux configurations de la façon suivante :

— si tous les états présent q dans la configuration C ont la même valeur de sortie $out(q) = y$, alors $out(C) = y$;
— sinon, nous dirons que $out(C)$ n'est pas défini.

Une configuration C sera dite *stable en sortie* si il existe $b \in Y$ une sortie possible

telle que $out(C) = b$ et que toute configuration C' accessible depuis C (*i.e.* $C \to^* C'$) vérifie $out(C') = b$. Une configuration stable est une configuration où tout les agents sont d'accord sur la sortie et où aucun ne changera jamais d'avis.

Soit ω un mot sur l'alphabet Σ. Nous dirons qu'un protocole de population $\mathcal{P}$:

— *accepte* le mot ω ou *reconnaît* ω si pour tout exécution équitable $(C_i)_{i\geq 0}$ telle que $C_0 = in(\omega)$, il existe t_0 tel que C_{t_0} est stable en sortie et sa sortie est $true$ — c'est-à-dire que, pour tout $t \geq t_0$, $out(C_t) = true$;

— *rejette* le mot ω ou *ne reconnaît pas* ω si pour tout exécution équitable $(C_i)_{i\geq 0}$ telle que $C_0 = in(\omega)$, il existe t_0 tel que C_{t_0} est stable en sortie et sa sortie est $false$ — c'est-à-dire que, pour tout $t \geq t_0$, $out(C_t) = false$.

Un langage $\mathcal{L} \subseteq \Sigma^*$ est *reconnu* par un protocole de population $\mathcal{P}$ si pour tout mot $\omega \in \Sigma^*$, ω est reconnu ou rejeté par $\mathcal{P}$ et ω appartient à $\mathcal{L}$ si et seulement si ω est reconnu par $\mathcal{P}$.

Si $\omega = \omega_1 \dots \omega_n$ est un mot et si $\pi \in \mathfrak{S}_n$ est une permutation, on notera ω^π le mot $\omega_{\pi(1)} \dots \omega_{\pi(n)}$

Définition 3 (Langage symétrique) *Un langage L sera dit* symétrique *si pour tout $\omega \in L$ et toute permutation $\pi \in \mathfrak{S}_{|\omega|}$, le mot ω^π appartient au langage L.*

Du fait des caractéristiques intrinsèques du modèle, comme l'anonymat des agents, si $\omega = \omega_1 \dots \omega_n$ est reconnu par un protocole de population P, alors pour toute permutation $\pi \in \mathfrak{S}_n$ le mot ω^π est reconnu par P.

Les langages reconnus par protocoles de populations sont donc symétriques.

2.2 Des résultats fondamentaux sur les protocoles de population classiques

2.2.1 L'arithmétique de Presburger et les ensembles semi-linéaires

Afin de caractériser précisément les langages reconnus par les protocoles de population classiques, nous avons besoin d'introduire l'arithmétique de Presburger et les ensembles semi-linéaires.

L'arithmétique de Presburger a été introduite par Presburger en 1930 [Pre30].

Définition 4 (Arithmétique de Presburger) *Un prédicat de* l'arithmétique de Presburger *est un prédicat du premier ordre sur la signature* $(+, \leq, =, 1, 0)$.

Autrement dit, un prédicat de l'arithmétique de Presburger est un prédicat qu'il est possible d'écrire avec uniquement les symboles suivants : $\forall x, \exists x, \vee, \wedge, \neg, +, \leq, =, 0, 1$. Les prédicats de l'arithmétique de Presburger sont interprétés sur l'ensemble des entiers $\mathbb{N}$ comme on s'y attend.

Autrement dit, l'arithmétique de Presburger correspond à l'arithmétique de Peano (l'arithmétique classique) mais sans la multiplication. Il est possible de multiplier –ou diviser– par une constante puisque cela revient à additionner un nombre connu de fois, mais il n'est pas possible de décrire la relation de divisibilité. Par exemple, les formules suivantes appartiennent à l'arithmétique de Presburger :

$$\begin{aligned} P(x) &= 3 \cdot x + 1 \geq 5 \\ Q(x,y) &= (x - y - 3 \equiv 2x \mod 5) \\ R(x,y,z) &= (x + y < z + 13) \vee (x - 2 \cdot y + z \equiv 2 \mod 5) \end{aligned}$$

Ils peuvent en effet être réécrits de la manière suivante :

$$\begin{aligned} P(x) &= 1+1+1+1+1 \leq x+x+x+1 \\ Q(x,y) &= \exists z, x = 1+1+1-y+x+x+z+z+z+z+z \\ R(x,y,z) &= (x+y \leq z+1+1+1+1+1+1+1+1+1+1+1+1+1) \\ &\quad \vee (\exists w, x+z = 1+1+y+y+w+w+w+w+w) \end{aligned}$$

Cependant, les formules ci-après n'y appartiennent pas :

$$\begin{aligned} M(x,y,z) &= x \cdot y = z \\ D(x,y) &= x \equiv 0 \mod y \end{aligned}$$

Il est possible de réécrire les prédicats de l'arithmétique de Presburger en prédicats équivalents ayant une forme plus pratique, où les quantificateurs ont été éliminés.

Proposition 2.2.1 (À partir de [Pre30] et [AAD+04]) *Soit $P(x_1, \ldots, x_d)$ un prédicat de l'arithmétique de Presburger.*

Il existe $P_1(x_1, \ldots, x_d), \ldots, P_k(x_1, \ldots, x_d)$ tels que $P(x_1, \ldots, x_d)$ est une combinaison booléenne des $P_1(x_1, \ldots, x_d), \ldots, P_k(x_1, \ldots, x_d)$ et tels que, pour $1 \leq i \leq k$, $P_i(x_1, \ldots, x_d)$ est de l'une des formes suivantes :

— $\Sigma_{i=1}^{d} a_i x_i \leq b$

— $\Sigma_{i=1}^{d} a_i x_i \equiv b \mod c$

pour $a_1, \ldots, a_d, b, c \in \mathbb{Z}$.

Cette proposition découle de méthodes utilisées par Presburger afin de démontrer la décidabilité des prédicats de l'arithmétique appelée aujourd'hui de Presburger. Le résultat utilisé par Presburger n'est pas formulé de cette manière : il n'utilise pas la notation "$\cdot \equiv \cdot \mod \cdot$". Cependant ce résultat est bien connu et mentionné dans un certain nombre de papier sous diverses formes. L'énoncé que nous utilisons est très proche de celui utilisé par Angluin *et al.* [AAD+04].

Nous introduisons maintenant les *ensembles linéaires* et *semi-linéaires*.

Définition 5 (Ensemble linéaire et semilinéaire)

— *Un* ensemble linéaire *de* $\mathbb{N}^d$ *est un ensemble de la forme*

$$\left\{ u + \sum_{i=1}^{p} a_i v_i \;\middle|\; a_i \in \mathbb{N} \right\}$$

où $u, v_1, \ldots, v_p$ *sont* $p+1$ *éléments de* $\mathbb{N}^d$.

— *Un* ensemble semi-linéaire *est un union finie d'ensembles linéaires.*

Un exemple simple d'ensemble linéaire est $\mathbb{N}$ puisque tout entier naturel n peut s'exprimer sous la forme $0+n\cdot 1$, 0 et 1 étant deux éléments de $\mathbb{N}^1$. Le nom d'ensemble linéaire est justifié par la propriété suivante que possède tous ces ensembles : si a et b, des éléments de $\mathbb{N}^d$, sont tels que a et $a+b$ appartiennent à un ensemble linéaire donné S avec alors pour tout $k \in \mathbb{N}$, l'élément $a+kb$ appartient aussi à l'ensemble linéaire S.

Un ensemble semi-linéaire est une union finie d'ensembles linéaires. À ce titre, un ensemble linéaire est donc un ensemble semi-linéaire particulier. Mais on peut construire des exemples plus généraux comme l'ensemble X des points de $\mathbb{N}^2$ tels que leur première coordonnée est paire et inférieure à la seconde strictement :

$$X = \left\{ (x,y) \in \mathbb{N}^2 \;\middle|\; (x \equiv 0 \mod 2) \wedge (x < y) \right\}$$

Les ensembles semi-linéaires sont stables par intersection, union et complément à $\mathbb{N}^d$. Cela veut dire que l'intersection ou l'union de deux ensembles semi-linéaires est un ensemble semi-linéaire, et si X est un ensemble semi-linéaire, alors $\mathbb{N}^d \setminus X$ en est un aussi.

Ginsburg *et al.* ont démontré en 1966 que les ensembles semi-linéaires sont exactement ceux définissables à l'aide des prédicats de l'arithmétique de Presburger.

Proposition 2.2.2 ([GS66]) *Soit* $X \subseteq \mathbb{N}^d$.
Il y a équivalence entre :

1. *X est un ensemble semi-linéaire*

2. il existe un prédicat $P(x_1, \ldots, x_d)$ de l'arithmétique de Presburger tel que

$$X = \{(x_1, \ldots, x_d) | P(x_1, \ldots, x_d)\}.$$

Il existe une troisième manière de construire ces ensembles. Pour cela nous redéfinissons d'abord la *fonction de Parikh*.

Définition 6 (Fonction de Parikh [Par61, Par66]) *Soit $\Sigma = \{\sigma_1, \ldots, \sigma_d\}$ un alphabet fini.*

La fonction de Parikh Φ_{Parikh} *est la fonction de Σ^* dans $\mathbb{N}^d$ qui, à un mot $\omega \in \Sigma^*$, associe le d-uplet $(x_1, \ldots, x_d)$ tel que pour tout $1 \leq k \leq d$, x_k est le nombre d'occurrence de la lettre σ_k dans le mot ω.*

Autrement dit, la fonction de Parikh renvoie le nombre d'occurrences de chaque lettre dans un mot, son résultat ne dépend pas de la place des lettres dans le mot. En utilisant la fonction de Parikh, il est possible d'obtenir de nouvelles caractérisations des ensembles semi-linéaires et de les relier aux langages réguliers et aux langages générés par grammaire hors-contexte.

Proposition 2.2.3 ([Par61, Par66]) *Soit $X \subseteq \mathbb{N}^d$.*

Il y a équivalence entre les propositions suivantes :

1. *X est ensemble semi-linéaire*
2. *il existe un langage $\mathcal{L}_1$ généré par grammaire hors-contexte tel que*

$$X = \Phi_{Parikh}(\mathcal{L}_1) \ ;$$

$$i.e.\ X = \left\{ x \in \mathbb{N}^d \;\middle|\; \exists \omega \in \mathcal{L}_1, x = \Phi_{Parikh}(\omega) \right\} \ ;$$

3. *il existe un langage régulier $\mathcal{L}_2$ tel que*

$$X = \Phi_{Parikh}(\mathcal{L}_2)$$

$$i.e.\ X = \left\{ x \in \mathbb{N}^d \;\middle|\; \exists \omega \in \mathcal{L}_2, x = \Phi_{Parikh}(\omega) \right\}.$$

Comme corollaire de cette proposition, Parikh a montré que les langages issus de grammaire hors-contexte et les langages réguliers sont indiscernables lorsqu'on les regarde à travers le prisme de la fonction de Parikh. Si $\mathcal{L}$ est un langage régulier (resp. généré par une grammaire hors contexte), alors son image par la fonction de Parikh est un ensemble semi-linéaire.

Nous utiliserons selon les besoins courants l'une ou l'autre manière de voir ces ensembles, résumées dans la proposition suivante.

Proposition 2.2.4 *Soit $X \subseteq \mathbb{N}^d$.*
Il y a équivalence entre :

1. *X est un ensemble semi-linéaire ;*
2. *il existe un prédicat $P(x_1, \ldots, x_d)$ de l'arithmétique de Presburger tel que*

$$X = \{(x_1, \ldots, x_d) \mid P(x_1, \ldots, x_d)\} \ ;$$

3. *il existe un langage $\mathcal{L}_1$ généré par grammaire hors-contexte tel que*

$$X = \Phi_{Parikh}(\mathcal{L}_1) \ ;$$

4. *il existe un langage régulier $\mathcal{L}_2$ tel que*

$$X = \Phi_{Parikh}(\mathcal{L}_2) \, .$$

Par exemple, considérons l'ensemble semi-linéaire $X = \{(n, n) \mid n \in \mathbb{N}\}$, le langage $\mathcal{L}_1$ généré par la grammaire hors-contexte $\{S \to T, T \to \varepsilon, T \to TT, T \to aTb\}$ qui génère le langage des mots bien parenthésés et le langage régulier $\mathcal{L}_2 = (ab)^*$. En identifiant $\mathbb{N}^2$ à $\mathbb{N}^{\{a,b\}}$, nous pouvons vérifier que

$$X = \Phi_{Parikh}(\mathcal{L}_1) = \Phi_{Parikh}(\mathcal{L}_2).$$

Dans la suite, pour tout ensemble fini Σ, nous identifierons les espaces $\mathbb{N}^\Sigma$ et $\mathbb{N}^{|\Sigma|}$.

2.2.2 Les langages calculables par protocoles de population

Dans [AAD+04], Angluin *et al.* montrent que les langages reconnus par protocoles de population sont stables par union, intersection et négation. Par ailleurs, ils montrent comment construire un protocole de population qui calcule un ensemble linéaire donné. Ils obtiennent donc la proposition suivante.

Proposition 2.2.5 ([AAD+04]) *Soit X un sous-ensemble semi-linéaire de $\mathbb{N}^\Sigma$. Il existe un protocole de population reconnaissant X.*

Une preuve détaillée et exhaustive de cette proposition est disponible dans [AAD+04], ci-après nous présentons quelques aspects de la preuve.

Pour calculer le complémentaire d'un langage, il suffit d'inverser la fonction de sortie. Pour calculer l'union ou l'intersection de deux langages, il faut construire un protocole qui calcul en parallèle l'appartenance à chacun des langages. La valeur de

sortie est alors le "OU" logique (le "ET" logique pour l'intersection) des valeurs de sorties calculées.

Du fait de la stabilité par les opérations logiques $\neg$ (qui correspond à l'opération ensembliste complémentaire), $\vee$ et $\wedge$ (qui correspondent respectivement aux opérations ensemblistes d'union et d'intersection), pour prouver la proposition 2.2.5, nous n'avons besoin de construire des protocoles de population calculant uniquement les formules de bases : $\Sigma_{i=1}^{d} a_i x_i \leq b$ et $\Sigma_{i=1}^{d} a_i x_i \equiv b \mod c$ pour $a_1, \ldots, a_d, b, c \in \mathbb{Z}$.

Nous ne donnerons ici la construction d'un protocole $P = (Q, \Sigma, in, out, \delta)$ que pour le cas $\Sigma_{i=1}^{d} a_i x_i \leq b$.

On pose $M = \max\{|a_1|, \ldots, |a_d|, 2b-1\}$.

L'ensemble des états Q est $\{-M, \ldots, M\} \cup \{0^+, 0^-\}$. La fonction d'entrée est telle que, pour tout $1 \leq i \leq d$, $in(\sigma_i) = a_i$. La fonction de sortie est telle que, pour tout $q \in Q$, $out(q) = false$ si $q > b$ ou $q = 0^-$ et $out(q) = true$ sinon. La relation de transition est construite comme suit. La paire d'agents dans l'état (q_1, q_2) prend la valeur :

— $(q_1+q_2, 0^*)$ si q_1 et q_2 sont non-nuls et si $|q_1+q_2| \leq M$; 0^* vaut 0^+ si $q_1+q_2 \leq b$ et 0^- sinon.

— $(\lfloor \frac{q_1+q_2}{2} \rfloor, \lceil \frac{q_1+q_2}{2} \rceil)$ si $|q_1 + q_2| > M$.

— $(q_1, 0^*)$ si q_1 est non nul et si $q_2 \in \{0^+, 0^-\}$; 0^* vaut 0^+ si $q_1 \leq b$ et 0^- sinon.

La somme des états sur toute la population est constante au cours du calcul. Si cette somme est comprise dans l'intervalle $[-M, M]$, alors la population évoluera vers une configuration où un agent porte un état représentant cette valeur et où tout les autres agents porteront un état nul renvoyant en sortie la valeur correcte. Si la somme dépasse M (respectivement $-M$), alors la population évoluera vers une configuration où la somme sera répartie entre plusieurs agents en utilisant uniquement des valeurs comprises entre b et M (respectivement $-b$ et $-M$), les autres agents portent la même valeur nulle qui renvoie la sortie correcte. La population évoluera de ces façons sous réserve d'équité : de telles configurations sont accessibles à chaque instant, donc la propriété d'équité assure qu'elles seront atteintes à un certain moment.

La proposition précédente, présentée dans [AAD+04], a ensuite été étendue en une caractérisation exacte de la puissance de calcul : il n'est pas possible de faire mieux.

Théoreme 2.2.1 ([AAE06b]) *Soit $X \subseteq \mathbb{N}^{\Sigma}$.*
Il y a équivalence entre :

1. *X est un ensemble semi-linéaire*
2. *X est reconnu par un protocole de population P.*

Nous ne donnerons pas ici la preuve complète de la semi-linéarité des langages reconnus par les protocoles de population, celle-ci est disponible dans [AAE06b]. Nous présentons cependant quelques étapes majeures de la preuve.

La preuve utilise les *fonctions de troncatures* $\tau_k : \mathcal{C} \to \mathcal{C}$ pour $k \geq 1$:

$$\forall c \in \mathcal{C},\ \forall q \in Q,\ \tau_k(c)(q) = \min(k, c(q)).$$

Les fonctions de troncatures vérifient les propriétés suivantes :

— $\forall c, d \in \mathcal{C},\ c \leq d \Rightarrow \tau_k(c) \leq \tau_k(d)$

— $\forall c, c', d \in \mathcal{C},\ \tau_k(c) = \tau_k(c') \Rightarrow \tau_k(c + d) = \tau_k(c' + d)$

Ensuite, l'ensemble des configurations $\mathcal{C}$ est partitionné entre l'ensemble des configurations stables $\mathcal{S}$ — c'est-à-dire les configurations c telles que $out(c)$ soit bien défini et que pour toute configuration c' accessible depuis c nous avons $out(c') = out(c)$ — et les configurations instables $\mathcal{U} = \mathcal{C} \setminus \mathcal{S}$.

L'ensemble $\mathcal{U}$ est clos vers le haut par inclusion, c'est-à-dire que :

$$\forall c \leq d,\ c \in \mathcal{U} \Rightarrow d \in \mathcal{U}.$$

Nous rappelons ici un corollaire utile du lemme d'Higman.

Lemme 2.2.1 (À partir de [Hig52]) *Tout sous-ensemble de $\mathbb{N}^d$ admet un nombre fini d'éléments minimaux.*

L'ensemble $\mathcal{U}$ a donc un nombre fini d'éléments minimaux et il est clos par inclusion vers le haut. Nous pouvons donc en déduire que l'ensemble des configurations $\mathcal{U}$ est donc entièrement caractérisé par un nombre finis de ses éléments. Nous obtenons alors le lemme suivant :

Lemme 2.2.2

— *Il existe $k \geq 1$ tel que $c \in \mathcal{U}$ si et seulement si $\tau_k(c) \in \mathcal{U}$.*

— *Il existe $k \geq 1$ tel que $c \in \mathcal{S}$ si et seulement si $\tau_k(c) \in \mathcal{S}$.*

Ce lemme dit qu'une sous-configuration bornée d'une configuration c donnée suffit à déterminer si la configuration c est stable et quelle est sa sortie.

Nous définissons maintenant *les extensions* d'une configuration c comme le sous-ensemble de $\mathbb{N}^\Sigma$ suivant :

$$X(c) = \left\{ x \in \mathbb{N}^\Sigma \;\middle|\; \exists d \geq c, c + x \to^* d \wedge \tau_k(c) = \tau_k(d) \right\}.$$

Cet ensemble correspond aux entrées que l'on peut ajouter à la configuration c et telles qu'après stabilisation la configuration d obtenue ait la même troncature que c.

En particulier, si x est une entrée telle que $x \to^* c$ où $c \in \mathcal{S}$ et y une entrée appartenant à $x + X(c)$, alors nous avons l'équivalence entre :

(1) x est accepté par P (2) y est accepté par P.

Il est ensuite démontré qu'un ensemble $Y \subseteq \mathbb{N}^\Sigma$ reconnu par le protocole de population P peut être recouvert par un nombre finis d'ensembles de la forme $x_i + M_i$ où x_i est une entrée et M_i une extension d'une configuration stable accessible depuis x_i.

Le fait que, pour toute configuration c, $X(c)$ soit un monoïde — c'est-à-dire qu'il contient l'entrée vide $\mathbf{0}$ et est stable par addition — et que de plus c'est un monoïde finiment engendré permet de conclure que $x_i + M_i$ est un ensemble linéaire. Par suite, y est un ensemble semi-linéaire.

2.2.3 Les protocoles de population déterministes et les protocoles non-déterministes

Dans le modèle classique [AAD+04, AAC+05, AAD+06, AAER07], les transitions sont représentées généralement par une fonction, et non une relation.

Pour plus de généralité, comme nous l'avons déjà écrit plus haut, nous utiliserons une relation de transition. De ce fait pour une paire d'agents qui interagit, il n'y a *a priori* pas unicité de la règle qui peut être appliquée.

Si, pour chaque paire d'agents, il y a unicité, alors le protocole de population est dit *déterministe* et la relation de transition est alors en fait une fonction. Sinon le protocole de population est dit *non-déterministe*. Le choix des paires qui interagissent reste cependant lui toujours non-déterministe, et soumis à la propriété d'équité. Le *degré de déterminisme* d'un protocole est mesuré par

$$d = \max_{(p,q)\in Q} \Big\{ |\{(r,s) \mid p\ q \to\ r\ s\}| \Big\}.$$

Beauquier *et al.* ont effectué une étude sur la différence entre ces deux variantes du modèle classique dans [BBRR12] afin de compléter les observations initiales de Angluin *et al.*.

Ils démontrent que cela ne modifie pas fondamentalement le modèle de calcul en terme d'expressivité ou de puissance.

Proposition 2.2.6 ([AR07]) *Les protocoles de population déterministes et non-déterministes reconnaissent les ensembles semi-linéaires.*

Si les protocoles de population déterministes et non-déterministes sont équivalents en terme de langages calculés, il est possible de les distinguer en terme de *langages de sortie* générés.

À une configuration $C \in \mathcal{C} = \mathbb{N}^Q$ d'un protocole de population, nous pouvons associer une *configuration de sortie* $O \in \mathbb{N}^Y$ de la manière suivante :

$$\forall y \in Y, \quad O(y) = \sum_{i \in Q,\ out(i)=y} C(i)$$

À une exécution $\mathcal{E} = C_0, C_1, \ldots, C_k, \ldots$, il est alors possible d'associer un mot — infini si l'exécution $\mathcal{E}$ est infinie — ω sur l'alphabet de sortie Y appelé le *mot de sortie* : $\omega = O_0, O_1, \ldots, O_k, \ldots$ où, pour tout k, O_k est la configuration de sortie associée à la configuration C_k. Nous associons alors à un protocole de population P le *langage de sortie* $L(P)$ qui est l'ensemble des mots de sortie qu'il peut générer.

Proposition 2.2.7 ([BBRR12]) *L'ensemble des langages de sortie générés par des protocoles de population déterministes est strictement inclus dans l'ensemble des langages de sortie générés par des protocoles de population non-déterministes.*

Le non-déterminisme dans le choix des interactions du modèle déterministe ne suffit pas pour simuler le non-déterminisme des règles du modèle non-déterministe. Cependant, en modifiant légèrement le modèle, il est possible de renverser ce résultat. Considérons en effet le modèle des *Protocoles de Population par triplet* qui ne diffère du modèle classique que par le fait que les agents interagissent par triplet et non plus par paire. Afin d'éviter toute confusion, nous parlerons ici, et uniquement ici, de *Protocoles de Population par paire* pour parler du modèle classique.

Proposition 2.2.8 ([BBRR12]) *Pour tout protocole de population par paire non-déterministe ayant un degré de non-déterminisme de d, il existe un protocole de population par triplet déterministe utilisant au moins $d + 2$ états initiaux différents qui génère le même langage de sortie.*

2.3 Des machines de Turing

Le modèle des machines de Turing est aujourd'hui encore le modèle de calcul de référence. C'est un modèle de calcul centralisé introduit en 1936 par Turing [Tur36]. Ce modèle de calcul est généralement décrit comme un automate de contrôle avec un nombre fini d'états couplé à un ou plusieurs rubans sur lequel l'automate lit et écrit via une tête de lecture (par ruban) en fonction de son programme interne.

Définition 7 (Machine de Turing non-déterministe) *Une* machine de Turing non-déterministe $(\Sigma, Q, \Gamma, q_0, F, \delta)$ *est formellement définie par :*

— Σ, *l'alphabet d'entrée*
— Q, *l'ensemble d'états de l'automate*
— Γ, *l'ensemble des symboles de travail pour les rubans qui contient un symbole spécial blanc* B
— $q_0 \in Q$, *l'état initial de l'automate*
— $F \subseteq Q$ *l'ensemble des états finaux acceptants*
— $\delta \subseteq (Q \times \Gamma^k) \times (Q \times \Gamma^k \times \{-1, 0, +1\}^k)$, *la relation de transition.*

La machine de Turing a k rubans : un ruban d'entrée, où elle ne peut que lire, un ruban de sortie, où elle ne peut qu'écrire (sa sortie) et $k-2$ rubans de travail, où elle peut lire et écrire.

De nombreuses classes de complexités sont définies grâce aux machines de Turing ; en fonction du temps ou de l'espace sur les rubans utilisé par la machine de Turing, et en fonction de son comportement (déterministe, non déterministe ou probabiliste par exemple) il est possible de hiérarchiser les problèmes. Nous définissons ci-après quelques classes qui nous seront utiles plus en avant.

Pour définir formellement les classes de machines de Turing avec des contraintes sur l'espace mémoire utilisé, nous avons besoin de la notion de fonction constructible.

Définition 8 (Fonction constructible) *Soit s une fonction de $\mathbb{N}$ dans $\mathbb{N}$.*

La fonction s est dite constructible *s'il existe une machine de Turing qui, pour tout $n \in \mathbb{N}$, utilise exactement $s(n)$ cases sur son ruban de calcul sur l'entrée 1^n.*

Par exemple, les fonctions $Id : n \mapsto n$ et $\log : n \mapsto \lfloor \log_2 n \rfloor$ sont constructibles. Plus généralement, toutes les fonctions usuelles sont constructibles.

2.3.1 Quelques machines non déterministes

Sur une même entrée et pour une même machine de Turing, il est possible qu'il y ait plusieurs exécutions. Si il existe une exécution acceptante parmi toutes les exécutions possibles d'une machine non-déterministe M sur une entrée donnée ω, alors la machine de Turing non-déterministe M reconnaît, ou accepte, le mot d'entrée ω.

Nous nous intéresserons principalement aux classes NSPACE (s) où s est une fonction constructible.

Définition 9 (NSPACE (s) **et** co-NSPACE (s)**)** *Soit s une fonction constructible.*

La classe NSPACE (s) *est l'ensemble des langages qui sont reconnus par une machine de Turing non-déterministe utilisant* $s(n)$ *cases sur ses rubans de travail pour une entrée de taille* n.

La classe co-NSPACE (s) *est l'ensemble des langages tels que leur complémentaire est dans* NSPACE (s).

Nous rappelons le résultat suivant dont nous ferons usage :

Proposition 2.3.1 ([Imm88]) *Soit* s *une fonction constructible telle que* $s = \Omega(\log)$.

$$\text{NSPACE}\,(s) = \text{co-NSPACE}\,(s)\,.$$

2.3.2 Quelques machines probabilistes

Une machine de Turing probabiliste est une machine de Turing qui, à chaque instant durant une exécution, peut avoir le choix entre différentes transitions possibles. La transition suivante est choisie en utilisant des bits aléatoires.

Une machine de Turing probabiliste aura pour chaque entrée une certaine probabilité d'accepter, cette probabilité pouvant être 0 ou 1.

Définition 10 (RSPACE (s) **et** co-RSPACE (s)**)** *Soit* s *une fonction constructible.*

La classe RSPACE (s) *est l'ensemble des langages tels qu'il existe une machine de Turing probabiliste* M *utilisant* $s(n)$ *cases sur ses rubans pour une entrée de taille* n *vérifiant les conditions suivantes :*

- *la machine* M *ne possède pas d'exécution infinie.*
- *si le mot en entrée appartient au langage, alors la machine* M *accepte avec une probabilité supérieure à* $\frac{1}{2}$.
- *si le mot en entrée n'appartient pas au langage, alors la machine* M *accepte avec probabilité 0.*

La classe co-RSPACE (s) *est l'ensemble des langages* L *tels que leur complémentaire* $\bar{L}$ *est dans* RSPACE (s).

Définition 11 (ZPSPACE (s)**)** *Soit* s *une fonction constructible.*

Nous définissons la classe $\text{ZPSPACE}\,(s) = \text{RSPACE}\,(n) \cap \text{co-RSPACE}\,(n)$.

La classe ZPSPACE (s) peut se voir comme l'ensemble des langages reconnus par les machines de Turing probabilistes avec erreur Zéro utilisant un espace mémoire borné par s.

De manière équivalente (et ce sera la caractérisation qui sera utilisée dans la toute suite), il est possible de caractériser la classe ZPSPACE (s) de la façon suivante.

Proposition 2.3.2 (Voir par exemple [Sak96]) *La classe* $\mathrm{ZPSPACE}(s)$ *correspond également à la classe des langages acceptés par une machine de Turing non-déterministe utilisant un espace mémoire borné par* s*, ayant trois états finaux Oui, Non et Peut-etre et vérifiant la propriété suivante :*

si $\omega \in L$ *(resp.* $\omega \notin L$*) alors toute exécution de la machine de Turing avec* ω *en entrée peut seulement retourner en sortie Oui ou Peut-etre (resp. Non or Peut-etre). Et pour chaque entrée, il existe des exécutions qui renvoient une sortie sûre (Oui ou Non).*

Dans la suite nous utiliserons beaucoup ces classes de complexités probabilistes ainsi que les classes non-déterministes définies ci-avant. Le résultat suivant permet de passer de l'une à l'autre : les langages reconnus par une classe de machines de Turing d'un type sont exactement les langages reconnus par la classe correspondante de l'autre type.

Proposition 2.3.3 (Voir par exemple [Sak96]) *Soit* s *une fonction constructible telle que* $s = \Omega(\log)$*.*

$$\mathrm{NSPACE}(s) = \mathrm{RSPACE}(s) = \mathrm{ZPSPACE}(s).$$

2.3.3 Les langages symétriques reconnus par machine de Turing

Le modèle des protocoles de population traite naturellement de langages symétriques. C'est pourquoi nous définissons ici les classes de langages symétriques reconnus par machines de Turing correspondantes aux classes définies plus haut.

Définition 12 *Pour* $X \in \{\mathrm{N}, \text{co-N}, \mathrm{R}, \text{co-R}, \mathrm{ZP}\}$ *et pour toute fonction constructible* s*, la classe des langages symétriques appartenant à* $\mathrm{XSPACE}(s)$ *est notée* $\mathrm{XSPACE}_{\mathrm{sym}}(s)$*.*

2.4 Quelques variantes

Nous discutons ici de quelques variantes du modèle des protocoles de populations. Les variantes considérées sont parmi le plus classiques de la littérature. Nous discuterons d'autres variantes dans les chapitres 5 et 6.

2.4.1 Où il est question de restreindre le graphe communication

Une première variante consiste à réduire le graphe d'interaction. Les agents sont placés sur les sommets d'un graphe. Deux agents peuvent interagir seulement s'il existe une arête entre les deux sommets où dont placés les agents en question. Les notions de configurations, d'exécution et d'équité sont conservées. Cependant les configurations ne peuvent *a priori* plus être encodées par un multi-ensemble car les agents sont sur des sommets qui ne sont plus indiscernables. Un vecteur d'état est donc utilisé à la place d'un multi-ensemble : il associe à chaque sommet du graphe l'état de l'agent qui y est placé. Dans certains cas nous pourrons utiliser des solutions intermédiaires, par exemple s'il y a une clique dans le graphe, la configuration de la sous-population constituée des agents sur la clique pourra être encodée par un multi-ensemble comme dans le cas classique. Le modèle classique peut être vu comme une instance de la variante protocole de population sur graphes où la famille de graphes considérée est celle des cliques.

Définition 13 (Protocole de Population sur un graphe) *Soit $G = (V, E)$ un graphe avec $|V| = n$.*

Un protocole de population sur G est un protocole de population dont l'ensemble des agents est identifié à V et où deux agents u et v peuvent interagir seulement si (u, v) est une arrête du graphe G.

Formellement : $C \to C'$ si C contient deux états q_1 et q_2 respectivement sur les sommets v_1 and v_2, et C' est obtenu de C en remplaçant q_1 et q_2 par q'_1 et q'_2, où $((q_1, q_2), (q'_1, q'_2)) \in \delta$, et $(v_1, v_2) \in E$.

Le calcul d'un langage se place maintenant sur une famille de graphe. On suppose que la famille de graphe associe pour chaque taille de population un graphe.

Cette variante a été étudiée dans [AAD+04] et [AAC+05], entre autres. Les auteurs démontrent que si la famille de graphes sous-jacents utilisée vérifie de bonnes propriétés alors il est possible de modéliser un ruban de machine de Turing sur la population.

Proposition 2.4.1 ([AAC+05]) *Pour tout entier naturel d, il existe un protocole de population qui travaille sur tout graphe de degré borné d qui construit un arbre couvrant du graphe sous-jacent.*

Il est possible d'utiliser l'arbre couvrant pour modéliser n cases d'un ruban de machine de Turing, où n est la taille de la population.

Nous reviendrons plus en détail dans le chapitre 3 sur cette construction. Schématiquement, le fait que les graphes utilisés aient un degré borné par une constante d peut être utilisé pour construire des identifiants locaux pour les agents de la population. Chaque agent aura un identifiant de telle façon que, pour un agent donné, tous ses voisins ont un identifiant différent. En utilisant ces identifiants locaux, le protocole construit un arbre couvrant sur la population. Et en utilisant cet arbre couvrant, le protocole peut organiser la population comme si les agents étaient disposés sur une chaîne. Chaque agent peut alors modéliser une case d'un ruban de machine de Turing, et la population peut alors modéliser un ruban avec autant de cases que d'agents.

La construction de ces identifiants permet aux agents d'utiliser des pointeurs vers leurs voisins, le nombre de pointeurs utilisés doit être borné par une constante indépendante de la taille de la population. Il est alors possible de récupérer les constructions et les résultats d'un modèle similaire proposé par Itkis et Levin [IL94]. Dans le modèle de Itkis et Levin, chaque agent a un espace mémoire de taille s et a en outre un nombre borné de pointeurs vers ses voisins, et ce indépendamment du graphe de communication. Leurs résultats se traduit par la proposition suivante.

Proposition 2.4.2 (À partir de [IL94] et [AAC$^+$05]) *Soit d un entier naturel. Soit L un langage sur Σ.*

Il existe un protocole de population qui travaille sur une famille de graphes de degré borné d et qui reconnaît le langage L si et seulement si $L \in \mathrm{RSPACE}_{\mathrm{sym}}(n)$.

Dans [AAC$^+$05], ils s'intéressent également à la question du calcul de propriété de la famille de graphe. Le problème n'est plus de calculer un prédicat en fonction d'une entrée distribuée sur les agents, mais de savoir si un prédicat donné est vrai pour le graphe sur lequel sont placés les agents. Par exemple, il existe un protocole de population qui détermine si le graphe sous-jacent a un degré borné par une constante d fixée.

2.4.2 Où il est question de donner plus de puissance aux agents

Une autre manière de rendre le modèle plus puissant est d'augmenter la puissance de calcul des agents. Dans ces variantes, les agents ne sont plus de simples automates finis mais des machines de Turing avec une contrainte sur la taille des rubans de calcul. Ce modèle est appelé *Passively Mobile Machines* et est plus longuement décrit dans [CMN$^+$11] et [CMN$^+$10].

Définition 14 (Passively Mobile Machines Protocol,[CMN$^+$11, CMN$^+$10])
Un Passively Mobile Machines Protocol est un 6-uplet $(X, \Gamma, Q, \delta, \gamma, q_0)$ *où :*

- X *est l'alphabet fini d'entrée,*
- Γ *est l'alphabet des rubans,*
- Q *est un ensemble fini d'états,*
- $\delta : Q \times \Gamma^4 \to Q \times \Gamma^4 \times \{L, R\}^4 \times \{0, 1\}$ *est la fonction de transition interne,*
- $\gamma : Q \times Q \to Q \times Q$ *est la fonction de transition externe*
- q_0 *est l'état initial*

On suppose que l'ensemble X *est un sous-ensemble de* Γ *et* Γ *contient un symbole spécial* $\square$ *qui n'est pas dans* X*.*

Un agent est équipé de :

- un émetteur-récepteur lui permettant d'envoyer et recevoir des messages avec les autres agents qui passent à portée,
- quatre rubans de machine de Turing sur lesquels sont écrits des mots sur l'alphabet Γ : un pour les messages entrant, un pour les messages sortants, un pour l'entrée et un pour les calculs internes, le reste des rubans est rempli par le symbole $\square$.
- un automate de contrôle dont l'ensemble des états est Q et qui commence dans l'état q_0 .
- un drapeau binaire : si le drapeau est à 1, cela signifie que l'agent effectue des calculs internes, et si le drapeau est à 0 que l'agent est prêt à interagir.

Un agent peut faire un pas interne selon sa fonction de transition interne et son état si son drapeau est à 1, ou bien faire un pas externe en interagissant avec un autre agent si son drapeau est à 0. Chaque agent envoie à l'autre le contenu de son ruban de sortie, inscrit ce qu'il reçoit dans son ruban d'entrée. Puis ils commutent leurs drapeaux et les états des deux agents changent selon la fonction de transition externe.

Comme un automate fini est équivalent à une machine de Turing n'utilisant qu'un nombre constant de cases en fonction de la taille de l'entrée, ce modèle est au moins aussi puissant que le modèle classique. Si chaque agent dispose d'un espace au moins logarithmique en la taille de la population, il est possible de calculer le prédicat $x_a = x_b \cdot x_c$ où x_i désigne le nombre d'occurrences de la lettre i dans le mot. La démonstration de ce résultat est disponible dans [CMN$^+$10]. Ce prédicat n'est pas semi-linéaire, ce qui prouve que ce modèle est strictement plus puissant que le modèle des protocoles de population classique.

Si aucune contrainte n'est imposée sur les machines de Turing utilisées, c'est-à-dire si les agents peuvent calculer aussi longtemps qu'elles le désire en utilisant autant de cases que désiré, alors ce modèle peut calculer n'importe quelle fonction symétrique

calculable. La variante proposée ici permet d'étudier les cas intermédiaires, ceux où les agents sont des machines de Turing disposant d'un espace bornée par une certaine fonction de la taille de l'entrée. Il est possible de déterminer la puissance du modèle obtenu en faisant varier la contrainte sur l'espace alloué aux agents.

Nous noterons $\mathrm{PMSPACE}\,(f(n))$ la classe des langages reconnus par des Passively Mobile Machines Protocols où chaque agent a une contrainte d'espace de $f(n)$ cases.

Proposition 2.4.3 ([CMN+11]) *Pour tout $f = \Omega(\log)$,*

$$\mathrm{PMSPACE}\,(f) = \mathrm{NSPACE}_{\mathrm{sym}}\,(n \cdot f)\,.$$

Une hiérarchie est construite par Chatzigiannakis *et al.* dans [CMN+11]. Ils utilisent la proposition précédente et obtiennent la suivante.

Proposition 2.4.4 ([CMN+11]) *Pour tout f et g deux fonctions constructibles telles que $f = \Omega(\log)$ et $g = o(f)$, nous avons*

$$\mathrm{PMSPACE}\,(g) \subsetneq \mathrm{PMSPACE}\,(f)\,.$$

Cette proposition définit une hiérarchie pour les protocoles de ce modèle utilisant un espace au moins logarithmique. Cette hiérarchie est directement héritée de celle des machines de Turing non-déterministes en utilisant la proposition précédente. Par ailleurs, toujours dans [CMN+11], ils montrent que si chaque agent a un espace en $o(\log\log n)$, où n est la taille de la population, alors le modèle a exactement la même puissance de calcul que le modèle classique des protocoles de population : il n'est possible de calculer que les ensembles semi-linéaires avec.

Proposition 2.4.5 *Si $f = o(\log\log)$, alors*

$$\mathrm{PMSPACE}\,(f) = \mathrm{PMSPACE}\,(1)\,.$$

Autrement dit, le modèle ne reconnaît que les ensembles semi-linéaires.

2.4.3 Où il est question de probabilité et de complexité temporelle

Le choix de la paire d'agents qui interagit ou de la règle à appliquer est un choix non-déterministe dans le cas classique. Il y a une contrainte d'équité, qui ne s'applique qu'aux exécutions infinies, mais à chaque instant tout pas de calcul possible peut avoir lieu. En changeant cette manière de faire on obtient d'autres variantes

ayant des puissances de calcul différentes. Une variante courante est d'utiliser des tirages aléatoires. L'utilisation de probabilité est naturelle dans le sens où l'équité utilisée a été construite pour modéliser un fonctionnement avec probabilité 1. Il y a cependant beaucoup de manières d'utiliser les probabilités. On peut tirer aléatoirement uniformément une paire d'agent, ou bien non-uniformément. On peut aussi avoir des probabilités sur quelle règle sera appliquée au prochain pas, avec des probabilités uniformes ou non entre les règles. On peut également faire dépendre les probabilités des proportions des différents états dans la population. Cette dernière variante tend à se rapprocher d'un modèle inspiré de la chimie, où une réaction a d'autant plus de chance de se produire que ses réactifs sont présents dans la solution.

La variante probabiliste la plus naturelle et la plus étudiée est la suivante :

Définition 15 (Protocole de Population Probabiliste [AAD$^+$04]) *Un* Protocole de Population Probabiliste *est un protocole de population où, à chaque pas, la paire d'agents qui interagit est choisie aléatoirement, indépendemment et uniformément parmi toutes les paires possibles.*

Cette variante, bien que différente de la variante classique sur un mécanisme fondamentale, reconnaît avec forte probabilité, les mêmes langages que la version classique.

Proposition 2.4.6 ([AAD$^+$04]) *Soit L un langage. Il y a équivalence entre les propositions suivantes :*

1. *L est semi-linéaire*
2. *Il existe un protocole de population probabiliste qui reconnaît L avec une probabilité de succès 1. De plus, une configuration stable est atteinte après $O(k_L n^2 \log n)$ interactions, où k_L est une constante dépendante de L.*

Idée de preuve :
Pour prouver cette proposition, il faut démontrer que les exécutions équitables du modèle classique correspondent exactement aux exécutions avec probabilité 1 du modèle probabiliste.

Une exécution non équitable est une exécution où infiniment souvent un évènement de probabilité non nulle n'est pas choisi. L'ensemble des exécutions refusant infiniment souvent certaines transitions est alors de probabilité nulle.

Le calcul de la durée d'une exécution nécessaire pour atteindre une configuration stable se fait en utilisant un protocole élisant un leader en $\Theta(n^2 \log n)$ interactions en moyenne. Ensuite, le leader centralise le calcul et il ne reste plus qu'à calculer le nombre moyen d'interactions nécessaires pour que le leader ait rencontré assez

de fois chaque agent. Plus de détails pour cette preuve peuvent être trouvés dans [AAD+04].

Cette preuve utilise un protocole élisant un leader. Beaucoup de protocoles utilisent un leader afin d'organiser le calcul. Un tel protocole part avec une population où chaque agent est candidat. À chaque fois que deux candidats interagissent, l'un d'eux perd son statut de candidat. À partir d'un certain moment il ne reste alors qu'un candidat qui est alors le leader. Plus de détails sur les protocoles élisant un leader et leurs utilisations sont disponibles dans [AAE06a].

Cependant, le remplacement d'une évolution non-déterministe soumise à une propriété d'équité par une évolution probabiliste permet de faire deux choses qui n'étaient pas possibles avec le modèle initial : simuler un ruban de machine de Turing avec un nombre conséquent de cases (puisque les seuls prédicats calculables sont semi-linéaires) et parler de complexité temporelle (puisque la propriété d'équité ne contraint pas les morceaux finis d'exécution). Dans le modèle classique, il était possible de construire des exécutions équitables où une configuration stable en sortie apparaît arbitrairement tard : il est généralement possible d'ajouter des sous-exécutions finies arbitrairement longues. Dans le modèle probabiliste, cette manipulation est toujours possible mais il est possible de mesurer la probabilité de ne pas avoir atteint une configuration stable en sortie en un certain nombre d'interaction.

Proposition 2.4.7 ([AAD+04]) *Soit $L \in \mathrm{NSPACE}(\log n)$. Soit d un entier positif et $T(n) = O(n^d)$ une borne sur la durée d'une exécution d'une machine de Turing reconnaissant le langage L.*

Pour tout entier $c > 0$, il existe un protocole de population probabiliste qui reconnaît L avec une probabilité d'erreur en $O(\frac{1}{n^c} \log n)$ et une configuration stable est atteinte en $O(n^{d+2} \log n + n^{2d+c+1})$ interactions.

Idée de preuve :
La construction se fait de la manière suivante. Nous ne donnerons ici que le squelette principal de la preuve, les détails sont entièrement disponibles dans le papier original [AAD+04].

Premièrement, un leader est élu dans la population afin de centraliser le calcul. Deuxièmement, la population est utilisée pour simuler des compteurs : k agents dans un état i représentent le compteur i avec la valeur k. L'objectif est de simuler une machine à compteur. Pour cela le leader doit pouvoir effectuer les opérations suivantes : tester la nullité d'un compteur et incrémenter ou décrémenter un compteur. Le leader peut tester si un compteur a la valeur 0 avec une probabilité d'erreur de l'ordre de $\Theta(n^{-c})$ pour une constante $c > 0$. Par ailleurs, l'incrémentation ou la décrémentation d'un compteur i peut se faire en ajoutant ou en retirant des agents

dans l'état i. De cette façon, il est possible de simuler avec forte probabilité le comportement d'une machine à compteurs dont la somme des compteurs est linéaire en la taille de la population.

Il est bien connu que les machines à compteurs dont la somme des compteurs est linéaire en la taille de la population peuvent simuler des machines de Turing utilisant un espace logarithmique [Min67]. Pour passer d'une machine de Turing avec rubans à une machine à compteurs, il fait représenter les rubans comme deux piles (la partie avant la tête de lecture et la partie après celle-ci) et encoder chaque pile dans un compteur.

Il est possible de contenir l'erreur afin d'obtenir une erreur finale bornée par $O(\frac{1}{n^c}\log n)$ et, en calculant le temps nécessaire pour élire un leader et simuler chaque pas de calcul, nous obtenons une exécution de $O(n^{d+2}\log n + n^{2d+c+1})$ interactions.

Cette proposition montre comment un protocole de population probabiliste peut simuler une machine de Turing utilisant un espace logarithmique avec probabilité $1-\varepsilon$. Ce résultat peut être vu comme optimal dans le sens où un protocole de population probabiliste ne peut pas reconnaître un langage non semi-linéaire avec probabilité 1 — cela contredirait la proposition 2.4.6. De plus un protocole de population probabiliste ne pourra pas simuler une machine de Turing utilisant un espace d'au moins $s(n)$ cases avec $\log n = o(s(n))$. En effet, une population de n agents anonymes où chaque agent est un automate fini à q états peut se trouver dans $O\left(n^q\right)$ configurations différentes ; une machine de Turing utilisant un espace s sur son ruban, k symboles sur son ruban et q' états pour son automate peut se trouver dans $q'.k^{s(n)}$ configurations différentes. Par suite, si $\log n = o(s(n))$, alors pour tout $q \in \mathbb{N}$ nous avons que $n^q = o\left(q'.k^{s(n)}\right)$. Et ainsi un protocole de population ne peut pas simuler une machine de Turing qui nécessite un espace de travail d'au moins $s(n)$, avec $\log n = o(s(n))$.

Beaucoup de protocoles de population commencent par élire un leader, c'est-à-dire identifier un unique agent dans la population, afin de centraliser tout ou partie du calcul. C'est pourquoi la complexité de l'élection d'un leader dans un protocole de population probabiliste est un problème clef.

En supposant qu'un leader est initialement présent dans la population, il est en effet possible d'améliorer les résultats précédents. Il est possible de reconnaître les langages semi-linéaires plus rapidement au prix d'une faible probabilité d'erreur.

Proposition 2.4.8 ([AAE06a]) *Soit L un langage semi-linéaire. Soit $c > 0$.*

Il existe un protocole de population probabiliste disposant d'un leader initialement qui reconnaît L avec une probabilité d'erreur de $O(\frac{1}{n^c})$ et une configuration stable est atteinte après $O(n\log^5 n)$ interactions.

Plus généralement, si un leader est déjà présent dans la population initiale, alors la simulation d'une machine de Turing est plus rapide et l'erreur peut être réduite.

Proposition 2.4.9 ([AAE06a]) *Soit $L \in \mathrm{NSPACE}(\log n)$. Soit d un entier positif tel que $O(n^d)$ est une borne sur la durée d'une exécution d'une machine de Turing reconnaissant le langage L.*

Pour tout entier $c > 0$, il existe un protocole de population probabiliste disposant d'un leader initialement qui reconnaît L avec une probabilité d'erreur en $O(\frac{1}{n^c})$ et une configuration stable est atteinte en $O(d\, n^d \log^2 n)$ interactions.

2.4.4 Où il est question d'application et de zèbres

ZebraNet est une expérience au Kenya menée par l'Université de Princeton. Cette expérience étudie les populations de zèbres et leurs déplacements. Pour cela des colliers équipés d'émetteur sont attachés aux zèbres et collectent des données, les colliers peuvent échanger des données et ils doivent régulièrement délivrer leurs données à une station de base. Un protocole de collecte de donnée est utilisé : toutes les données présentes dans la population doivent être délivrées à la station de base.

Beauquier *et al.* [BBBD13] ont étudié ce protocole dans une version simplifiée, mais qui est suffisante cependant pour montrer que le protocole utilisé ne garantit pas que toutes les valeurs sont bien délivrées à la station de base, certaines données pouvant circuler en bouclant infiniment entre différents zèbres. Ils utilisent pour leur étude une variante du modèle des protocoles de population où à chaque agent x est associé un *temps de couverture* $\mathbf{cv}_x$: dans toute sous-exécution de longueur $\mathbf{cv}_x$, l'agent x a rencontré chacun des autres agents de la population. Un agent ne connaît pas son temps de couverture, ni celui des autres agents.

Ils proposent alors deux versions modifiée du protocole initial, appelées $MZP1$ et $MZP2$, garantissant que toutes les données sont délivrées à la station de base. Par ailleurs, ils déterminent le temps nécessaire dans le pire cas en fonction de $\mathbf{cv}_{max}$ et $\mathbf{cv}_{min}$, respectivement les temps de couverture maximal et minimal dans la population.

Le protocole $MZP1$ repose sur le principe suivant : tout donnée transmise à un autre zèbre ne peut plus être transmise qu'à la station de base. Cela assure aussi qu'une donnée ne peut pas cycler sur un ensemble de zèbres.

Proposition 2.4.10 ([BBBD13]) *Sur une population de taille n, le protocole $MZP1$ permet la délivrance de toute les données en au plus $(n-1)\mathbf{cv}_{max} - 2(n-2)$ interactions.*

Le protocole $MZP2$ repose lui sur le principe suivant : tout zèbre ayant transmis ses données à un autre zèbre ou à la station de base ne peut plus accepter de données de la part d'un autre zèbre. Cela assure en particulier qu'une donnée ne peut pas cycler sur un ensemble de zèbres.

Proposition 2.4.11 ([BBBD13]) *Sur une population de taille n, le protocole $MZP2$ permet la délivrance de toute les données en au plus $2\mathbf{cv}_{max} - 2$ interactions.*

Pour les deux cas, il n'est pas supposée de borne sur la taille de la mémoire, ce qui empêcherait par exemple qu'un zèbre collecte toutes les données de la population. Les auteurs montrent que l'hypothèse d'une mémoire bornée par une constante en fonction de la taille de la population ne change pas fondamentalement les protocoles (un zèbre peut refuser des données si sa mémoire est pleine) et ne change pas leur complexité.

Ces résultats semblent indiquer que le protocole $MZP2$ a une meilleure complexité que le protocole $MZP1$. Il faut cependant garder à l'esprit que ce sont des complexité dans les pires cas. En affinant la modélisation d'une population de zèbre — en la divisant en une sous-population en bonne santé et rapide et une sous-population vieille ou malade et lente — le protocole $MZP1$ semble être mieux adapté.

2.4.5 Où il est question de communication

D'autres variantes ont été étudiées dans [AAER07]. Les communications dans le modèle classique sont bidirectionnelles et instantanées. Ces deux hypothèses peuvent être modifiées pour obtenir des modèles de communications différents.

Les communications peuvent être *unidirectionnelles*, c'est-à-dire qu'une interaction entre une paire d'agents se déroule de la manière suivante : un agent envoie un message contenant son état à un second qui reçoit le message et change son état en conséquent. Si l'agent qui envoie le message est autorisé à changer son état suite à son envoi, nous parlerons de *modèle avec transmission*, sinon nous parlerons de *modèle avec observation*.

La réception d'un message peut avoir lieu immédiatement après son envoi — nous parlerons alors de *modèle avec livraison immédiate* — ou bien la réception peut être retardée. Si la réception est retardée, les messages entrants peuvent s'accumuler dans le buffer d'un agent. Si un agent peut choisir de retarder la lecture d'un message entrant en attendant d'avoir la chance d'envoyer un message, nous parlerons de *modèle avec livraison en file d'attente*, sinon nous parlerons de *modèle avec livraison retardée*.

Plus formellement, les différents modèles obtenus peuvent être décrits via leurs différences avec la définition du modèle classique.

Transmission immédiate : la relation de transition δ est modifiée comme suit : il existe des relations $\delta_1 \subseteq Q \times Q$ et $\delta_2 \subseteq Q^2 \times Q$ telles que $p\ q \to r\ s \in \delta$ si et seulement si $(p, r) \in \delta_1$ et $((p, q), s) \in \delta_2$.

Observation immédiate : dans ce modèle, la relation de transition est modifiée de la même manière que pour le modèle avec transmission immédiate mais en plus la relation δ_1 est réduite à l'identité.

Transmission en file d'attente : nous ajoutons à la définition du modèle un ensemble fini M de messages et nous transformons la relation de transition δ en deux relations d'actions : $\delta_s \in Q \times (M \times Q)$ qui est la relation d'émission et $\delta_r \in (Q \times M) \times Q$ qui est la relation de réception. Un agent dans un état q peut envoyer le message m en passant dans l'état r si $(q, (m, r)) \in \delta_s$ et un agent dans l'état p peut passer dans l'état s en recevant le message m si $((q, m), r) \in \delta_r$. La description d'une configuration doit prendre en compte les message en attente de réception et l'ensemble des configurations est alors $\mathcal{C} = \mathbb{N}^{Q \cup M}$.

Transmission retardée : ce modèle est le cas spécial du modèle avec transmission en file d'attente où la relation de réception est totale, c'est-à-dire qu'elle vérifie : $\forall q \in Q,\ \forall m \in M,\ \exists p,\ ((q, m), p) \in \delta_r$.

Observation retardée : ce modèle est le cas spécial du modèle de transmission retardée où la relation d'émission ne change pas l'état de l'agent émetteur, c'est-à-dire qu'elle doit vérifier : $\forall q, p \in Q,\ \forall m \in M,\ (q, (m, p)) \in \delta_s \Rightarrow q = p$.

Afin de caractériser précisément les prédicats reconnus par ces variantes, nous avons besoin d'introduire les familles de prédicats et les notions suivantes.

Nous construisons une notion de similarité entre deux prédicats selon des sous-alphabets. Soit $\Sigma' \subseteq \Sigma$ un sous-alphabet de l'alphabet d'entrée. Une entrée $x \in \mathbb{N}^\Sigma$ est dite *k-riche selon* Σ' si $\forall \sigma' \in \Sigma',\ x(\sigma') \geq k$ et $\forall \sigma' \in \Sigma \setminus \Sigma',\ x(\sigma) = 0$. Deux prédicats ψ et ψ' sur $\mathbb{N}^\Sigma$ sont *k-similaires selon* Σ' si, pour toute entrée x qui est k-riche selon Σ', nous avons $\psi(x) = \psi'(x)$.

Nous définissons maintenant les classes suivantes telles qu'elles sont définies dans [AAER07] :

MOD est la classe des combinaisons booléennes des prédicats modulo.

coreMOD est la classe des prédicats ψ' tels qu'il existe $\psi \in MOD$, $k \in \mathbb{N}$ et $\Sigma' \subseteq \Sigma$ tels que ψ' est k-similaire à ψ selon Σ'.

$\mathbf{COUNT_k}$ est la classe des combinaisons booléennes de prédicats qui définissent des ensembles de la forme $\{x \mid x(\sigma) \geq r\}$ où $\sigma \in \Sigma$ et $r \leq k$.

COUNT$_*$ est l'union des classes $COUNT_1, COUNT_2, COUNT_3, \ldots$.

Ces classes correspondent à des ensembles de prédicats qu'il est possible de reconnaître sous certaines conditions.

Proposition 2.4.12 ([AAER07]) *Nous donnons ici pour chaque modèle la classe des prédicats reconnus.*

- *En utilisant le modèle avec transmission en file d'attente, l'ensemble des prédicats reconnus est l'ensemble des prédicats semi-linéaires.*
- *En utilisant le modèle avec transmission immédiate, l'ensemble des prédicats reconnus est l'ensemble des prédicats semi-linéaires et qui sont dans $coreMOD$.*
- *En utilisant le modèle avec transmission retardée, l'ensemble des prédicats reconnus est l'ensemble des prédicats semi-linéaires et qui sont dans $coreMOD$.*
- *En utilisant le modèle avec observation immédiate, l'ensemble des prédicats reconnus est l'ensemble des prédicats $COUNT_*$.*
- *En utilisant le modèle avec observation retardée, l'ensemble des prédicats reconnus est l'ensemble des prédicats $COUNT_1$.*

D'autres variations du modèle original existent. Par exemple, dans [BCC$^+$13], les auteurs considèrent la variante du modèle classique où les interactions sont symétriques. Cette variante est appelée *les Protocoles de Population Symétriques*. La relation de transition doit vérifier les deux conditions suivantes :

- Pour toute règle $p\ p \to r\ s$, nous avons $r = s$.
- Si $p \neq q$ et $p\ q \to r\ s$, alors $q\ p \to s\ r$.

Cependant les auteurs montrent ensuite que la symétrie ainsi ajoutée au modèle peut être brisée. Ils obtiennent alors la proposition suivante.

Proposition 2.4.13 ([BCC$^+$13]) *Les prédicats reconnus par protocoles de population symétriques sont exactement les prédicats semi-linéaires.*

Dans le chapitre 5, nous explorons d'autres variantes basées, entre autre, sur la forme de la relation de transition.

2.5 Modèles de calcul proches des protocoles de population

Le modèle de calcul des protocoles de population peut-être rapproché d'autres modèles de calculs comme les *Réseaux Stochastiques de Réaction Chimique* — que

nous appellerons SCRN dans la suite — ou les *Systèmes d'Addition de Vecteurs* — que nous appellerons VAS dans la suite.

Le modèle des SCRN est un modèle de calcul s'inspirant d'une modélisation stochastique des réactions chimiques. Ce modèle a été étudié en profondeur par Cook et al dans [CSWB09]. Un ensemble de molécules baigne en solution. Selon les concentrations relatives des différentes espèces chimiques, des molécules d'espèces données ont une probabilité de s'entrechoquer plus ou moins grande. Lorsque des molécules s'entrechoquent, elles peuvent éventuellement réagir selon une équation de réaction qui détruira les réactifs et créera les produits. Dans ce modèle, chaque réaction a *a priori* une vitesse de réaction propre, cela est traduit par une augmentation (pour les réactions rapides) ou une diminution (pour les réactions lentes) de la probabilité d'interaction des réactifs.

Contrairement aux véritables réactions chimiques, on ne se restreint pas avec l'hypothèse de la conservation de la matière. On s'autorise à avoir des réactions chimiques telles que :

$$A + B \to A + B + C.$$

Une réaction chimique est habituellement supposée réversible et un équilibre chimique qui dépend des concentrations des espèces présentes et des conditions expérimentales est atteint ; cependant dans le modèle des SCRN on ne considérera pas cette contrainte : il peut y avoir une réaction $A + B \to C + D$ sans la réaction $C + D \to A + B$.

Un SCRN est défini par la donnée :

— d'un ensemble fini de m espèces chimiques,

— d'un ensemble fini de d réactions.

L'état du système est décrit par un vecteur $z \in \mathbb{N}^m$. Pour $i < m$, $z(i)$ donne le nombre de molécules de l'espèce i dans la solution.

Une réaction α est un triplet de $\mathbb{N}^m \times \mathbb{N}^m \times \mathbb{R}^+$. Une réaction $\alpha = (l_\alpha, r_\alpha, k_\alpha)$ est définie par la donnée d'un vecteur $l_\alpha \in \mathbb{N}^m$, qui donne la quantité de réactifs nécessaire, d'un vecteur $r_\alpha \in \mathbb{N}^m$, qui donne la quantité de produits créés, et un réel positif k_α qui est la constante de réaction. La réaction $\alpha = (l_\alpha, r_\alpha, k_\alpha)$ peut avoir lieu dans le système dans l'état z si $z \geq l_\alpha$ — si pour tout espèce chimique $i < m$, $z(i) \geq l_\alpha(i)$. À l'issue de la réaction, le nouvel état du système est $z - l_\alpha + r_\alpha$.

Dans l'état z, le temps d'attente avant qu'une réaction α se produise est une variable aléatoire exponentiellement distribuée ; cela signifie en particulier que la dynamique d'un SCRN est un processus markovien à temps continu. La probabilité que la prochaine règle appliquée soit la règle α est proportionnelle à k_α et à $\frac{1}{V^{r_\alpha(i)}} \frac{z(i)!}{(z(i)-r_\alpha(i))!}$ pour tout $i < m$ (où V est le volume de la solution), ce dernier terme

correspond approximativement à $q_i^{r_\alpha(i)}$ où q_i est la concentration de l'espèce i dans la solution $q_i = \frac{z(i)}{V}$.

Le système est dit stabilisé lorsque il n'y a plus de réaction possible. Le résultat du calcul est alors lu en fonction de la présence ou de l'absence d'une espèce donnée.

Le modèle des SCRN et celui des protocoles de population peuvent être rapprochés de la manière suivante : les agents des protocoles de population correspondent aux molécules des SCRN, les règles de la relation de transitions correspondent aux réactions chimiques. Les modèles diffèrent sur d'autres points : le nombre de molécules est *a priori* variable, les interactions des agents dans les protocoles de population se font uniquement par paire et l'évolution du système dans un SCRN est stochastique alors que, pour un protocole de population, elle est non-déterministe et soumise à une propriété d'équité.

Cependant ces différences peuvent être partiellement gommées et ainsi il est possible de traduire certains résultat des SCRN vers les protocoles de population. Nous pouvons par exemple considérer la variantes des protocoles de population où le nombre d'agents après interactions est variable — comme nous le faisons au chapitre 5 par exemple. Le nombre d'agents à interagir peut aussi être supposé variable selon les règles. Cette modification, comme la précédente, ne modifient pas la puissance du modèle. Des variantes probabilistes des protocoles de populations ont été considérées — comme à la section 2.4.3 — mais l'évolution probabiliste est différente : d'un côté le choix est uniforme, de l'autre il dépend de la configuration.

Le premier résultat étudié par les auteurs de [CSWB09] est le suivant :

Proposition 2.5.1 ([CSWB09]) *Le modèle des SCRN est Turing-universel.*

Cette proposition signifie que toute machine de Turing peut-être simulée par un SCRN. La démonstration de ce résultat est l'occasion, pour les auteurs, de montrer diverses constructions.

Les auteurs montrent d'abord que, pour tout circuit logique, il est possible de construire un SCRN qui calcule la même fonction. Chaque porte logique peut être remplacée par un nombre fini de règles, et chaque variable est remplacée par deux espèces chimiques, une espèce représente la variable dans l'état vrai et l'autre dans l'état faux.

Ensuite, ils montrent que leur modèle de SCRN est équivalent avec d'autres modèles tels que les *Réseaux de Petri*, les VAS (Systèmes d'Addition de Vecteurs) ou les *Programmes Fractran*.

Les Réseaux de Petri sont un modèle de calcul introduit par Petri en 1962 [Car62]. Un réseau de Petri est représenté par un graphe biparti orienté ; l'un des ensembles de sommets représente les places et l'autre les transitions. Un certains nombres de

jetons sont répartis sur les places. S'il y a des jetons sur chaque place ayant une flèche qui entre dans une transition, il est possible d'effectuer cette transition. La réalisation de la transition détruit les jetons à l'origine des flèches entrantes et en crée à la sortie des flèches sortantes de la transition. La correspondance formelle entre les modèles des Réseaux de Petri et des SCRN se fait en associant une place et une espèce chimique, un jeton dans une place et une molécule d'une espèce chimique, une transition et une équation.

Le modèle VAS a été introduit par Karp et Miller dans [KM69]. Un calcul dans ce modèle correspond à une marche dans $\mathbb{N}^d$ pour un d fixé. La configuration du système est le point de l'espace courant et le programme est l'ensemble des pas possibles, c'est-à-dire un ensemble de vecteurs de $\mathbb{Z}^d$. La correspondance formelle entre le modèle des VAS et celui des SCRN se fait en associant à chaque configuration du système chimique un vecteur d'entier de $\mathbb{N}^d$ où d est le nombre d'espèces chimiques entrant possiblement en jeu dans le système, chaque équation chimique correspond alors à un vecteur de $\mathbb{Z}^d$ (les réactifs sont comptés négativement et les produits positivement). La seule difficulté du passage d'un modèle à l'autre réside dans la prise en compte des catalyseurs — une espèce chimique C dans une réaction de la forme $A+C \to B+C$ — dans ce cas là il convient d'utiliser un intermédiaire de réaction — une nouvelle espèce chimique I est utilisée et la réaction $A+C \to B+C$ est divisée en $A+C \to I$ et $I \to B+C$.

Pour plus de détails sur les passages d'un modèle à l'autre, les constructions sont développées dans [CSWB09].

La question de l'accessibilité — savoir s'il est possible d'aller d'un point x à un point y — pour les VAS a été résolu et a été prouvée décidable par Mayr en 1981 [May81] ; une solution plus simple et plus élégante à été proposée par Leroux en 2012 [Ler12].

Le modèle des Programmes Fractran est un modèle de calcul où l'état du système est un entier $n \in \mathbb{N}\backslash\{0\}$ et où l'ensemble des transitions est une liste finie de fractions. Si le système est dans l'état n, il passe dans l'état $n\frac{p}{q}$, où $\frac{p}{q}$ est la première fraction dans la liste telle que $n\frac{p}{q}$ est un entier. Plus de détails sont donnés dans [Con87]. La correspondance bien que directe est moins visible pour ce cas, cependant si on réécrit l'entier n en le décomposant comme un produit de nombre premier il est possible de l'interpréter comme un vecteur d'entiers positifs.

Nous rappelons qu'une *Machine à Compteurs* est un automate fini couplé non pas à un ou plusieurs rubans comme une machine de Turing mais à un nombre fini de compteurs. L'automate peut incrémenter ou décrémenter chaque compteur et tester si leur valeur est nulle. Une étude plus complète sur les machines à compteurs est disponible dans [FMR68]. En utilisant un compteur pour stocker le ruban à droite

de la tête de lecture, un compteur pour stocker le ruban à gauche et un compteur pour stocker la tête de lecture, il est possible de simuler une machine de Turing avec un ruban à l'aide d'une machine à compteurs avec trois compteurs.

Proposition 2.5.2 ([FMR68]) *Une machine de Turing avec k rubans fonctionnant en temps linéaire peut être simulée par une machine à compteurs fonctionnant en temps exponentiel.*

De manière plus générale, toute machine de Turing peut être simulée par une machine à compteurs avec un ralentissement exponentiel.

Les auteurs de [CSWB09] utilisent les machines à compteurs comme intermédiaires entre les SCRN — et donc les VAS et les Réseaux de Petri — et les machines de Turing. Un compteur de la machine à compteurs est simulé par une espèce chimique : m molécules d'une espèce chimique donnée dans la solution signifie qu'un compteur donné stocke la valeur m. Tester si un compteur est vide revient à tester la présence d'une espèce chimique dans la solution. Pour réaliser ce test, les auteurs construisent une horloge à l'aide d'un ensemble de réactions qui font osciller une partie du système entre deux configurations. Les autres réactions, qui simulent le comportement de la machine, ont une vitesse de réaction plus rapide mais sont cadencées par l'horloge. De cette façon, si un pas est réalisable dans la configuration courante, il est exécuté avec très forte probabilité avant que l'horloge ne fasse un battement. La construction complète ainsi que l'intégralité des calculs et des améliorations sont disponibles dans [CSWB09].

Le fait que le modèle des SCRN soit Turing puissant, c'est-à-dire qu'il puisse simuler des machines de Turing, montre que le problème suivant est indécidable : étant donné un système dans un état x, quelle est la probabilité qu'il atteigne un état y ? Cela ne constitue pas une contradiction avec le problème de l'accessibilité qui est décidable pour les VAS et donc aussi pour les SCRN. Le problème de l'accessibilité pose juste la question de la possibilité et non de la probabilité.

2.6 Résumé et conclusion

Dans ce chapitre bibliographique, nous avons introduit un certain nombre de protocoles de population et de variantes dont la puissance est déterminée. Les variantes décrites ont été comparées à différentes classes de machines de Turing probabilistes ou non-déterministes. Nous avons également fait le lien entre le modèle des protocoles de population et d'autres modèles de calculs moins courants que les machines de Turing, comme les réseaux stochastiques de réactions chimiques ou les systèmes d'addition de vecteurs.

Dans le chapitre 3, nous complétons ce chapitre bibliographique en comparant la construction d'une machine de Turing sur une variante particulière des protocoles de population telle qu'elle apparaît dans la littérature avec notre propre version de cette construction.

Dans les chapitres 5 et 6, nous présentons d'autres variantes de protocoles de population. Les résultats présentés dans ces chapitres sont des contributions originales.

Chapitre 3

Des Protocoles de Population pour faire des Machines de Turing

Dans ce chapitre, nous présentons des résultats de simulation de machines de Turing par des protocoles de population travaillant sur des graphes d'interaction réduits — qui ne sont pas des cliques. Angluin *et al.* décrivent la simulation de machines de Turing par des protocoles de population travaillant sur des graphes d'interactions ayant un degré borné dans [AAC+05]. Nous présentons leurs résultats dans la section 3.1. Dans les sections suivantes, nous introduisons un formalisme que nous avons développé afin de composer les protocoles de population dans un carde formel. En utilisant, ce formalisme nous avons construit notre version d'un protocole simulant une machine de Turing. Nous obtenons ainsi une preuve et une construction qui nous semble plus directe, et plus modulable, du résultat. Les constructions et résultats de ces dernières sections ont été publiées dans [BL13a, BL13b]. Nous utiliserons ces constructions dans le chapitre suivant. Tout les graphes considérés dans ce chapitre, et les suivants, seront supposés connexes.

Autrement dit, l'objectif de ce chapitre est de prouver le théorème suivant :

Théoreme 1 *Un langage L est calculable par un protocole de population travaillant sur les chaînes si et seulement si L est un langage symétrique de* $\mathrm{NSPACE}(n)$.

et de son corollaire :

Théoreme 2 *Un langage L est calculable par un protocole de population travaillant sur les graphes à degré borné si et seulement si L est un langage symétrique de* $\mathrm{NSPACE}(n)$.

L'objectif est aussi d'introduire un formalisme permettant la composition et la modularité dans la conception de protocoles de population.

3.1 Simulation d'une machine de Turing sur un graphe à degré borné

L'article [AAC+05] décrit une manière de modéliser une machine de Turing avec un protocole de population travaillant sur une famille de graphes dont le degré des nœuds est borné par une constante d. Nous expliquons ici les principales étapes de cette construction.

L'algorithme de [AAC+05] peut être décomposé en trois sous-algorithmes :

— construire un coloriage à distance 2 sur le graphe ;
— utiliser cet étiquetages des sommets pour construire un arbre couvrant du graphe ;
— utiliser cet arbre couvrant pour simuler un ruban de machine de Turing.

Un *coloriage à distance 2* d'un graphe G est un coloriage des sommets de G de telle sorte que deux sommets adjacents ou qui ont un voisin en commun sont coloriés de deux couleurs différentes.

Une famille de graphes est dite *coloriable à distance 2* s'il existe un protocole de population qui, quelque soit la configuration initiale, finira par atteindre une configuration (sous réserve d'équité) qui correspond à un coloriage à distance 2. Le même protocole doit marcher sur tout les graphes de la famille.

Proposition 3.1.1 ([AAC+05]) *Une famille de graphes est coloriable à distance 2 si et seulement si la famille de graphes est à degré borné : il existe une constante d qui borne le degré de tout les graphes de la famille.*

Idée de preuve :
La preuve du "si" de la proposition utilise un jeton mobile de leader qui mémorise des informations sur les couleurs des agents qu'il visite et de leurs voisins. Après un éventuel déplacement, le jeton finit par interagir avec deux voisins. S'ils ont la même couleur, l'un change de couleur, si l'un a la même couleur que l'agent où est le jeton, il change de couleur. Ensuite le jeton refait ultimement un déplacement. Grâce à la propriété d'équité, la population finira par atteindre une configuration où deux voisins d'un même agent n'ont pas la même couleur. Si deux jetons mobiles se rencontre un est détruit et l'autre continue son oeuvre. Une preuve plus détaillée peut être trouvée dans [AAC+05].

Réciproquement, puisque les agents ont un nombre fini d'états, le degré est borné par le nombre de couleurs (qui est plus petit que le nombre d'états) dans un coloriage à distance 2.

■

Notre définition de famille de graphes coloriables à distance 2 est fortement dépendante du modèle des protocoles de population : en utilisant un modèle plus fort — des agents avec plus de mémoire allouée par exemple — il est possible d'augmenter le nombre de famille de graphes coloriable à distance 2.

La proposition suivante montre comment un protocole de population peut utiliser un tel coloriage afin d'organiser la population.

Proposition 3.1.2 ([AAC+05]) *Soit $(G_n)_{n\in\mathbb{N}}$ une famille de graphes coloriable à distance 2. Il existe un protocole de population qui construit un arbre couvrant de G_n pour tout $n \in \mathbb{N}$.*

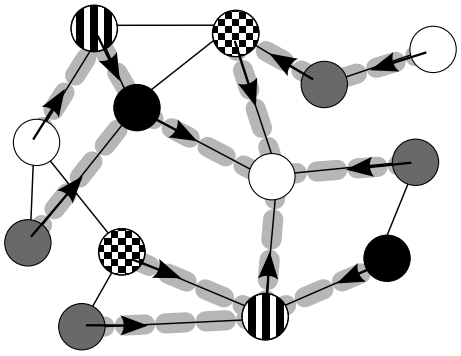


FIGURE 3.1 – Arbre couvrant

Idée de preuve :
La preuve repose sur le fait qu'un coloriage à distance 2 d'un graphe permet de simuler un nombre fini de pointeurs sur chaque sommet du graphe. En effet, chaque agent peut stocker dans son état la couleur d'un de ses voisins (il y a un nombre borné de couleurs possibles) et, par définition du coloriage utilisé, un sommet a au plus un voisin portant une couleur donnée. En d'autres termes la couleur d'un agent peut être utilisée comme identifiant local et permet à ses voisins de pointer vers lui sans ambiguïté. Cela permet de construire une forêt où les liens de parenté dans l'arbre sont les pointeurs et la forêt peut finir par être connexe en utilisant des jetons leader et sous réserve d'équité ; nous obtenons alors un arbre couvrant du graphe.

Plus formellement, la construction de la forêt utilise des jetons leader. Un jeton commence la construction d'un arbre à partir d'un sommet arbitrairement choisi comme racine. Ensuite, il explore le graphe par un parcours en profondeur. Tant qu'il trouve un sommet qui n'appartient pas un arbre, il se déplace sur cet agent et l'ajoute à l'arbre — il devient le fils du noeud précédent. S'il n'est pas possible

d'ajouter de nouveau fils, le jeton leader recule au père de l'agent où il est et cherche pour cet agent un fils supplémentaire. Si un jeton en rencontre un autre, l'un est détruit et des jetons d'effacements sont envoyés pour effacer l'arbre en construction du jetons leader détruit. Le nombre de leaders décroît ainsi jusqu'à ce qu'il n'en reste plus qu'un. Lorsqu'il ne reste qu'un seul jeton leader — la propriété d'équité nous assure que cela arrivera — ce leader réussira à construire correctement un arbre couvrant du graphe.

Le coloriage à distance 2 et la construction de l'arbre couvrant s'effectuant en parallèle, si un agent dans un arbre est recolorié, un jeton *reset* est produit, il efface l'arbre et réinitialise le jeton *leader*.

Au bout d'un certain temps, le coloriage sera stable, il ne restera qu'un seul jeton *leader* construisant l'arbre couvrant, et donc un arbre couvrant sera effectivement construit.

■

Proposition 3.1.3 ([AAC+05], en utilisant [IL94]) *Soit $(G_n)_{n\in\mathbb{N}}$ une famille de graphes.*

S'il existe un protocole de population sur $(G_n)_{n\in\mathbb{N}}$ qui organise les agents en arbre couvrant à degré borné, alors il existe un protocole de population qui simule un ruban de machine de Turing de RSPACE(n) *sur G_n, pour tout $n \in \mathbb{N}$.*

Idée de preuve :
Une fois le coloriage à distance 2 effectué et l'arbre couvrant construit, la population d'agents peut être utilisée comme ruban de machine de Turing. Les auteurs de [AAC+05] se ramènent à un cas particulier d'un modèle proposé par Itkis et Levin dans [IL94] qui est, dans ce cas là, équivalent aux machines de Turing de RSPACE(n).

■

Dans le reste du chapitre, nous présentons une preuve alternative de ces résultats.

3.2 Des protocoles de population composables

3.2.1 Composition

En utilisant la définition formelle classique des protocoles de population il est compliqué de parler de composition de protocoles et d'expliquer simplement comment plusieurs protocoles peuvent calculer ensemble sur une même population. Nous

proposons un formalisme alternatif qui permet de parler de composition et qui permet de donner des conditions sous lesquelles la composition est faisable. Ensuite, nous utiliserons cette composition pour construire un protocole de population qui simule une machine de Turing sur un graphe ligne en composant quelques protocoles simples. Nous caractériserons la classe des machines de Turing simulables. Avec cette construction, nous redémontrons donc la caractérisation de la puissance de la variante considérée, celle des protocoles de population avec graphe d'interaction restreint.

Tout d'abord, nous définissons donc les protocoles de population travaillant avec environnement. Cet environnement représente une vue sur l'espace de calcul d'autres protocoles travaillant sur la même population et dont on veut utiliser le résultat. Ce modèle formel peut être rapprocher des *protocoles de population avec entrée stabilisante* proposés par Angluin *et al.* dans [AAC+05].

Définition 16 (Protocole de Population avec Environement) *Un* protocole de population avec environnement $P = (Q, Q_{env}, \Sigma, in, out, \delta)$ *(qui sera souvent écrit* (Q, Q_{env}, δ)*) est formellement défini par :*

- Q*, un ensemble fini d'états de travail ;*
- Q_{env}*, un ensemble fini d'états d'environnement ;*
- Σ*, un alphabet fini d'entrée ;*
- in*, une fonction d'entrée de* Σ *dans* Q *;*
- out*, une fonction de sortie de* Q *dans* $\{true, false\}$ *;*
- $\delta \subseteq (Q \times Q_{env})^2 \times Q^2$*, une relation de transition.*
  Nous écrirons $\begin{pmatrix} e_1 \\ a_1 \end{pmatrix} \begin{pmatrix} e_2 \\ a_2 \end{pmatrix} \to b_1\, b_2$ *au lieu de* $(((a_1, e_1), (a_2, e_2)), (b_1, b_2)) \in \delta$*.*

L'état de chaque agent est divisé en deux :

- une composante d'environnement que le protocole suppose constante et qu'il utilise afin de mener son calcul ;
- une composante de travail sur laquelle le protocole effectue son calcul de la même façon que pour un protocole classique mais en tenant compte de l'environnement.

Nous conservons les notions de configurations, d'exécutions et d'équités du modèle classique. Cependant la notion de calcul est légèrement modifiée. Avec les protocoles classiques, nous nous intéressons aux exécutions qui commencent avec une configuration correspondant à la traduction d'un mot sur l'alphabet d'entrée par la fonction d'entrée. Ici nous nous intéressons aux exécutions commençant sur des configurations vérifiant une contrainte initiale sur son espace de travail, telles qu'à tout instant l'environnement vérifie une contrainte environnementale, et qu'à partir

d'un certain moment, toute les configurations aient un espace de travail vérifiant une contrainte finale. Plus formellement, nous définissons les *contraintes de progression*.

Définition 17 (Contraintes de progression) *Soit Q un ensemble d'états de travail et Q_{env} un ensemble d'états d'environnement. Soit δ une relation transition appartenant à $(Q \times Q_{env})^2 \times Q^2$. Soit θ un prédicat sur les mots sur Q, σ un prédicat sur les mots sur Q_{env} et ψ un prédicat sur les mots sur $Q \times Q_{env}$.*

Un protocole de population (Q, Q_{env}, δ) avec environnement vérifie les contraintes de progression *(θ, σ, ψ) si pour tout exécution équitable $(C_0, C_{env,0})(C_1, C_{env,1}) \dots (C_i, C_{env,i})\ (C_{i+1}, C_{env,i+1}) \dots$, commençant avec une configuration initiale $(C_0, C_{env,0})$ vérifiant $\theta(C_0)$, et telle que, pour tout $t \geq 0$, $\sigma(C_{env,t})$ soit vérifié, alors il existe t_0 tel que, pour tout $t \geq t_0$, la configuration $(C_t, C_{env,t})$ est telle que $\psi(C_t, C_{env,t})$ soit vérifié.*

Les contraintes de progression permettent par exemple d'exprimer le fait qu'un unique leader a été élu dans la population — la contrainte finale étant alors $\exists! x,\ x : L$ — et qu'ainsi notre protocole d'élection de leader a terminé. Il faut bien noter cependant que les agents ne connaissent généralement pas l'évaluation de la contrainte finale sur la population. Ces contraintes de progression permettent aussi de parler d'un protocole qui démarre sur une population où un leader est déjà élu — la contrainte initiale impliquant alors $\exists! x,\ x : L$.

Afin de simplifier les écritures des contraintes de progressions, nous introduisons l'opération $\cdot\|\cdot$ sur les prédicats : si ψ_1 est un prédicat sur un mot écrit sur Q_1 et ψ_2 est un prédicat sur un mot écrit sur Q_2, alors $\psi_1\|\psi_2$ est un prédicat sur un mot écrit sur $Q_1 \times Q_2$ tel que pour tout mot $C = (C_1, C_2)$ écrit sur $Q_1 \times Q_2$

$$\psi_1\|\psi_2(C) = true \text{ si et seulement si } \psi_1(C_1) = true \text{ et } \psi_2(C_2) = true,$$

où C_i représente la projection de C sur Q_i pour $i = 1, 2$.

Si deux protocoles de populations calculent sur la même population en parallèle, l'un deux peut voir l'autre comme faisant partie de son environnement et utiliser ainsi les résultats de celui-ci pour aider son calcul. Cette façon de combiner deux protocoles de population avec environnement sera appelée *composition parallèle*. L'idée est de construire à partir d'un protocole P_1 et d'un protocole P_2 un protocole $P_3 = P_1\|P_2$. La proposition 3.2.1 permet alors de garantir que sous certaines hypothèses le protocole P_3 se comporte "bien".

Avant d'introduire la composition parallèle, dans le soucis de simplifier les notations, nous introduisons l'opération $\cdot\|\cdot$ sur les relations de transition : si $\delta_1 \in (Q_1 \times Q_e)^2 \times Q_1^2$ est une relation de transition et si $\delta_2 \in (Q_2 \times (Q_1 \times Q_e))^2 \times Q_2^2$ est

une relation de transition, alors $\delta_1 \| \delta_2 \in ((Q_2 \times Q_1) \times Q_e))^2 \times (Q_1 \times Q_2)^2$ est une relation de transition telle que :

$$\begin{pmatrix} e \\ (p,q) \end{pmatrix} \begin{pmatrix} e' \\ (p',q') \end{pmatrix} \to (r,s)\,(r',s') \in \delta_1 \| \delta_2$$

si et seulement si

$$\begin{pmatrix} e \\ p \end{pmatrix} \begin{pmatrix} e' \\ p' \end{pmatrix} \to r\,r' \in \delta_1 \text{ et } \begin{pmatrix} (p,e) \\ q \end{pmatrix} \begin{pmatrix} (p',e') \\ q' \end{pmatrix} \to s\,s' \in \delta_2.$$

Proposition 3.2.1 (Composition Parallèle) *Soit $P_1 = (Q_1, Q_{env}, \delta_1)$ un protocole de population avec environnement qui vérifie les contraintes de progression $(\theta_1, \sigma_1, \psi_1)$. Soit $P_2 = (Q_2, Q'_{env}, \delta_2)$ un protocole de population avec environnement qui vérifie les contraintes de progression $(\theta_2, \sigma_2, \psi_2)$. Supposons que :*

1. *$Q_1 \times Q_{env} \subseteq Q'_{env}$;*
2. *ψ_1 implique σ_2 ;*
3. *pour toute exécution $(C_t, C_{env,t})_{t \geq 0}$ de P_2, $\theta_2(C_0)$ implique $(\forall t.\theta_2(C_t))$.*

Soit ψ_3 un prédicat sur $(Q_1 \times Q_2) \times Q_{env}$ tel que $\psi_3((C_1, C_2), C_e)$ soit vérifié si et seulement si $\psi_2(C_2, (C_1, C_e)) \wedge \psi_1(C_1, C_e)$ soit vérifié.

Alors le protocole de population $(Q_1 \times Q_2, Q_{env}, \delta_1 \| \delta_2)$, noté $P_3 = P_1 \| P_2$, satisfait les contraintes de progression $(\theta_1 \| \theta_2, \sigma_1, \psi_3)$.

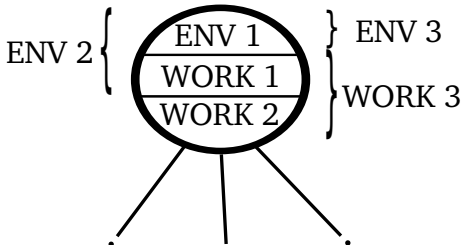


FIGURE 3.2 – Schéma de la composition.

Le leader est un agent distingué dans la population. Il est très souvent utilisé pour mener à bien tout ou partie d'un calcul. Un *protocole avec leader* est protocole de population (avec environnement) tel que

— il existe $Q_{leader} \subset Q$ un ensemble d'états possibles pour un unique leader : si il existe initialement un unique agent ayant un état dans Q_{leader}, alors, pour toute configuration accessible, il y aura toujours exactement un agent avec un états dans Q_{leader}.
— chaque pas de calcul non trivial implique nécessairement le leader : pour toute règle $(r, s) \to (p, q)$, si ni r ni s ne sont dans Q_{leader}, alors $(r, s) = (p, q)$. En d'autres termes, une règle est l'identité ou implique le leader.
— Q_{leader} a trois états spéciaux : L_0 appelé *l'état initial*, L_f^+ *l'état final acceptant* et L_f^- *l'état final rejetant*.

Pour simplifier les écritures, nous utiliserons la syntaxe suivante (inspirée par `C` et Java). Si ϕ est un prédicat sur Q^2 et si $\delta_1, \delta_2 \subset Q^2 \times Q^2$, alors $\phi?\delta_1 : \delta_2$ est le sous-ensemble de $Q^2 \times Q^2$ qui vérifie

$$((p, q), (r, s)) \in (\phi?\delta_1 : \delta_2)$$
$$\text{si et seulement si}$$
$$(\phi(p, q) \wedge ((p, q), (r, s)) \in \delta_1) \text{ ou } ((\neg\phi(p, q)) \wedge ((p, q), (r, s)) \in \delta_2).$$

Pour les protocoles avec leader, il est possible de définir une autre composition : la composition séquentielle. Généralement, il n'est pas possible pour un agent de déterminer si un calcul a fini ou non. Un protocole avec leader, cependant, peut savoir si le calcul est terminé ou non. Il est alors possible d'implémenter l'instruction séquentielle "*puis*". C'est ce que fait la *composition séquentielle*, formellement définie ci-après.

Proposition 3.2.2 (Composition Séquentielle) *Soit $P_1 = (Q_1, Q_{env,1}, \delta_1)$ un protocole de population avec leader et environnement qui vérifie les contraintes de progression $(\theta_1, \sigma_1, \psi_1)$. Soit $P_2 = (Q_2, Q_{env,2}, \delta_2)$ un protocole de population avec leader et environnement qui vérifie les contraintes de progression $(\theta_2, \sigma_2, \psi_2)$. Supposons :*

1. $Q_{env} = Q_{env,1} = Q_{env,2}$
2. $\sigma_1 = \sigma_2$
3. σ_1 *implique l'existence d'un leader*
4. *Pour tout $t \geq 0$, $\psi_1(C_t, C_{env,t})$ implique $\theta_2(C_t)$*

La dernière condition implique que les états finaux du leader de P_1 sont des états initiaux pour le leader de P_2. Soit $\varphi_1(x, y)$ (resp. $\varphi_2(x, y)$) un prédicat tel qu'il est vrai si et seulement si x ou y est un leader dans un état non final pour P_1 (resp. P_2).

Alors le protocole de population $(Q_1 \cup Q_2, Q_{env}, (\varphi_1?\delta_1 : (\varphi_2?\delta_2 : Id)))$, noté $P_3 = P_1; P_2$, vérifie les contraintes de progression $(\theta_1, \sigma_1, \psi_2)$.

De même qu'il est possible de définir "*puis*", en utilisant un protocole avec leader, on peut implémenter l'instruction "*tant que*".

Définition 18 *Soit $P = (Q, Q_{env}, \delta)$ un protocole de population avec environnement et leader qui satisfait les contraintes de progressions (θ, σ, ψ).*

Soit φ un prédicat sur $Q \times Q_{env}$.

Nous supposons que ψ implique θ sur la composante de travail.

Nous noterons $[P]_\varphi = (\theta \| true, \sigma, true \| (\forall x.x = 1), Q \times \{0,1\}, Q_{env}, \delta \| \tau)$ le protocole de population qui, à la fin de son calcul, sauve le résultat du calcul sur une nouvelle composante et, si le leader vérifie φ, relance un calcul du début, grâce aux règles τ.

La condition "ψ implique, sur la composante de travail, θ" sera appelle *condition de persistance.*

3.2.2 Des graphes à degré borné aux chaînes

Grâce au formalisme défini ci-dessus, il est possible de donner un sens à la simulation du calcul d'un protocole travaillant sur une famille de graphes par un protocole travaillant sur une autre famille de graphe.

Définition 19 (Plongement) *Soit $(G_n)_{n\in\mathbb{N}}$ et $(H_n)_{n\in\mathbb{N}}$ deux familles de graphes. Pour tout $n \in \mathbb{N}$, nous posons $G_n = (V_n, E_n)$ et $H_n = (W_n, F_n)$.*

Un plongement *de $(G_n)_{n\in\mathbb{N}}$ dans $(H_n)_{n\in\mathbb{N}}$ est une fonction f de $\bigcup_{n\in\mathbb{N}} V_n$ dans $\bigcup_{n\in\mathbb{N}} W_n$ telle que :*

— $\forall n \in \mathbb{N}, \exists m \in \mathbb{N},\ f(V_n) = W_m$

— $(u, v) \in E_n$ *implique que* $\exists m \in \mathbb{N},\ ((f(u), f(v)) \in F_m) \vee (f(u) = f(v))$

Par exemple, la famille des cycles

$$\Big((\{i \mid i \in [\![0, n-1]\!]\}, \{(i, i+1 \mod n) \mid 0 \leq i < n\}) \Big)_{n\in\mathbb{N}}$$

peut se plonger dans la famille des chaînes

$$\Big((\{i \mid i \in [\![0, n-1]\!]\}, \{(i, i+1) \mid 0 \leq i < n-1\}) \Big)_{n\in\mathbb{N}}$$

de la façon suivante : pour tout sommet i du cycle de taille k, nous lui associons le sommet $f(i) = \min(i, k-i)$ de la chaîne de taille $\lceil \frac{k}{2} \rceil$. Inversement la famille des chaînes se plonge dans la famille des cycles en conservant les mêmes sommets et en supprimant juste une unique arête.

Cependant cette notion de plongement est trop vaste pour être utilisable : il est possible de plonger n'importe quel graphe dans le graphe à un sommet. Nous nous intéresserons aux plongements calculables par protocoles de population.

Définition 20 (Plongement calculable) *Soit deux familles de graphes* $(G_n)_{n\in\mathbb{N}} = (V_n, E_n)_{n\in\mathbb{N}}$ *et* $(H_n)_{n\in\mathbb{N}} = (W_n, F_n)_{n\in\mathbb{N}}$.

Un plongement f *de* $(G_n)_{n\in\mathbb{N}}$ *dans* $(H_m)_{m\in\mathbb{N}}$ *est* calculable par protocole de population *si il existe un protocole de population* $P_{G,H}$ *tel que pour tout protocole de population* P_1 *travaillant sur* G_n *il existe un protocole de population* $P_2 = P_1 \| P_{G,H}$ *travaillant sur* H_m *simulant* P_1.

En utilisant cette notion de plongement calculable, nous pouvons reformuler les propositions 3.1.1 et 3.1.2 de la façon suivante.

Proposition 3.2.3 *Soit* $d \in \mathbb{N}$. *Soit* $(G_n)_{n\in\mathbb{N}}$ *une famille de graphes tel que, pour tout* n, G_n *est de degré borné par* d.

Il existe un plongement calculable de la famille des arbres enracinés à degré borné par d *dans* $(G_n)_{n\in\mathbb{N}}$.

Preuve. Le protocole de population avec environnement construit en composant les protocoles de population des propositions 3.1.1 et 3.1.2 construit un arbre couvrant sur un graphe à degré borné. Ce protocole calcule donc le plongement qui renvoie l'arbre couvrant proposé en sortie du protocole dans le graphe. Par suite, ce plongement est un plongement calculable. En composant ce protocole qui construit un arbre couvrant avec un protocole travaillant sur les arbres, il est possible d'obtenir un protocole travaillant sur les graphes à degré borné et renvoyant le même résultat pour la même entrée.

■

Il est aussi possible de plonger de manière calculable les chaînes sur les arbres à degré borné. Nous pouvons reformuler la proposition 3.1.2 de la manière suivante :

Proposition 3.2.4 *Soit* $d \in \mathbb{N}$. *Soit* $(T_n)_{n\in\mathbb{N}}$ *une famille de graphes tel que, pour tout* n, G_n *est un arbre enraciné de degré borné par* d.

Il existe un plongement calculable de la famille des chaînes dans $(T_n)_{n\in\mathbb{N}}$.

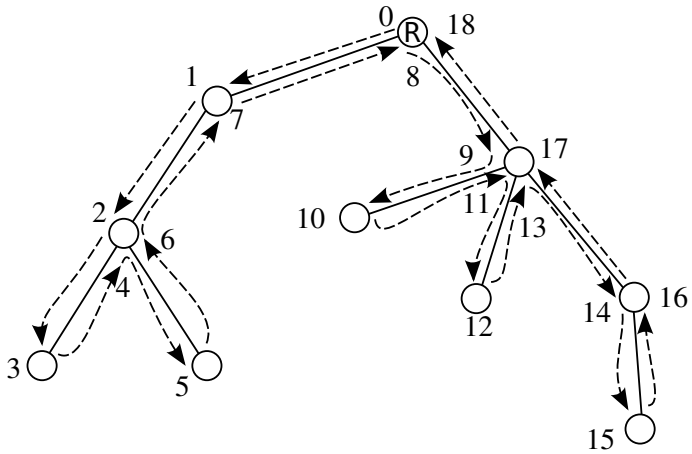


FIGURE 3.3 – Un arbre enraciné en R. Le parcours des flèches en pointillés représente une chaîne plongée sur ce graphe.

Preuve. Nous construisons une suite de sommets sur l'arbre enraciné de la manière suivante. Le premier sommet est la racine. Ensuite nous prenons les sommets dans l'ordre d'un parcours en profondeur en ajoutant les sommets à chaque fois que nous passons dessus (à l'aller et au retour). La figure 3.3 illustre ce parcours via les flèches en pointillés. Comme le graphe est à degré borné, chaque sommet a un nombre borné de voisins et le parcours décrit ci-avant ne passera qu'un nombre borné de fois sur chacun des sommets. Cette condition permet la construction d'un tel parcours par un protocole de population.

Il est important de noter qu'un sommet n'enregistre que les informations suivantes : combien de cases il simule et, pour chacune de ces cases, lequel de ses voisins porte la case précédente et lequel porte la case suivante. Toutes ces informations peuvent être stockées avec un nombre fini d'états. Remarquons qu'un sommet ne connaît en général pas la place précise de la case qu'il simule dans la chaîne.

■

Ce plongement justifie que, par la suite, nous nous intéressons à construire une simulation de machine de Turing non pas sur un arbre à degré borné comme les auteurs de [AAC+05] mais plutôt directement sur une chaîne. En composant notre simulation avec le protocole calculant ce plongement, il est possible d'obtenir une simulation de machine de Turing sur un arbre à degré borné. En composant ensuite avec le protocole calculant le plongement des arbres sur les graphes à degré borné, nous pouvons généraliser facilement le résultat aux graphes à degré borné. Tout cela

est permis par le formalisme des protocoles de population composables. Pour prouver le théorème 2, il suffit donc de prouver le théorème 1. Nous y consacrons le reste de ce chapitre.

3.2.3 Des protocoles de population pour modéliser une machine de Turing sur une chaîne

3.2.3.1 Trouver les deux extrémités d'une chaîne

Dans cette sous-section, nous montrons le lemme suivant :

Lemme 3.2.1 *Il existe un protocole travaillant sur chaînes qui, sous réserve d'équité, marque les agents extrémaux de la chaîne, en distinguant une extrémité de l'autre.*

Nous commençons par construire un protocole pour trouver une extrémité d'une chaîne. Formellement, le protocole $Extremity$ (défini dans la formalisation standard) utilise l'ensemble d'états suivant :

$$Q_{Extremity} = \{E_0, E_1, E'_0, E'_1, M_0, M_1, C_0, C_1, D_0, D_1, S_0, S_1\}.$$

Chacun de ces états stocke deux informations. Une est représentée par la lettre et donne des indications sur la géographie et l'avancement du protocole. L'autre information est stockée dans un nombre en indice (0 ou 1). Elle représente un lien de synchronisation entre deux voisins.

Les états E_0, E_1, E'_0, E'_1 signifient "ici se trouve une extrémité ", ils seront appelés états extrémité. E'_0, E'_1 sont dit marqués.

Les états C_0, C_1 signifient "je connais un état extrémité marqué et j'en cherche un non-marqué", D_0, D_1 signifient "je sais qu'il y a deux états extrémité et je vais détruire le premier état extrémité marqué trouvé", S_0, S'_1 signifient "je ne connais pas d'état extrémité mais j'en cherche" Et les états M_0, M_1 signifie "milieu" (par opposition à extrémité). Les états X_0 et X_1 seront appelés *états* X, pour X appartenant à $\{E, E', D, C, S, M\}$.

Chaque agent commence dans un état extrémité. Lorsque deux états extrémité interagissent, l'un est détruit. Les agents dans un état extrémité cherchent en permanence à avoir le même indice que leur voisin (ils supposent n'en avoir qu'un). Cela n'est effectivement possible que s'ils n'ont qu'un voisin. Les états $C_0, C_1, D_0, D_1, S_0, S_1$ sont des jetons qui essayent de détecter la présence de deux agents dans un état extrémité pour détruire l'un des deux. De cette manière, le protocole trouvera exactement une et une seule extrémité.

Nous définissons la relation d'équivalence $\cdot \cong \cdot$ par les classes d'équivalence : $\{E_0, E_1, E'_0, E'_1\}$ et $\{M_0, M_1, C_0, C_1, D_0, D_1, S_0, S_1\}$. Cette relation est étendue aux configurations sur des chaînes de longueur n de la façon suivante :

$$c \cong d \text{ si et seulement si}$$
$$(\forall m < n,\ c(m) \cong d(m)) \vee (\forall m < n,\ c(m) \cong d(n-m-1)).$$

Deux configurations sont considérées comme équivalentes si elles ont le même nombre d'états extrémité et qu'ils sont aux mêmes endroits (modulo une éventuelle symétrie du graphe).

Une configuration c sera dite *stable* si $\forall d,\ c \to^* d \Rightarrow (c \cong d)$: en d'autres termes, une configuration est stable si aucun état extrémité ne peut disparaître ou se déplacer.

Nous dirons que deux agents sont *synchronisés* si l'un d'eux est dans un état extrémité et si les deux ont le même indice.

Nous utiliserons la notation $|c|_X$ pour représenter le nombre d'agent dans l'état X au sein de la configuration c, et cette notation est généralisée avec la relation de récurrence : $|c|_{X,Y,\ldots} = |c|_X + |c|_{Y,\ldots}$.

Initialement, tout les agents sont dans l'état E_0. Le protocole utilise les règles de transition suivantes pour $i, j, k \in \{0, 1\}$ avec $i \neq k$, $e_i \in \{E_i, E'_i\}$,$x_i \in \{M_i, S_i, D_i, C_i\}$ et $z \in \{S_i, C_i, D_i\}$:

$$\begin{array}{rcll}
E_i, E_j & \to & E'_0, C_0 & (1) \\
E'_i, E_j & \to & M_0, E'_0 & (2) \\
E'_i, E'_j & \to & E'_{1-i}, E'_{1-j} & (3) \\
e_i, M_i & \to & e_{1-i}, M_{1-i} & (4) \\
E_i, C_i & \to & E_0, D_1 & (5) \\
E'_i, C_i & \to & E'_{1-i}, C_{1-i} & (6) \\
E_i, D_i & \to & E_{1-i}, D_{1-i} & (7)
\end{array}
\qquad
\begin{array}{rcll}
E'_i, D_i & \to & E_{1-i}, S_{1-i} & (8) \\
E_i, S_i & \to & E'_{1-i}, C_{1-i} & (9) \\
E'_i, S_i & \to & E'_{1-i}, S_{1-i} & (10) \\
e_i, x_k & \to & x_0, e_0 & (11) \\
M_i, z_j & \to & z_i, M_j & (12) \\
z_i, S_j & \to & z_i, M_j & (13) \\
C_i, C_j & \to & D_i, D_j & (14)
\end{array}$$

Lemme 3.2.2 *Au cours de n'importe quelle exécution commençant avec tout les agents dans l'état E_0, les propriétés suivantes restent vraies :*

1. *Pour toute configuration accessible c,*

$$|c|_{E_0,E_1,E'_0,E'_1} \geq 1.$$

2. *Pour toute configuration accessible c qui n'est pas configuration initiale :*

$$|c|_{C_0,C_1,D_0,D_1,S_0,S_1} \geq 1.$$

3. *Pour toute configuration accessible c,*

$$|c|_{E'_0,E'_1} = |c|_{C_0,C_1} + |c|_{D_0,D_1}.$$

4. *Pour toute configuration accessible c, un état D peut être généré si et seulement si il existe au moins deux états extrémité dans la population.*

Preuve.

1. La propriété est vérifiée initialement. Pour chaque règle avec des états extrémité dans la partie gauche, il y a au moins un état extrémité dans la partie droite. Par suite, il y a toujours au moins un état extrémité dans la population.
2. La première application d'une règle met nécessairement un agent dans un état C_0. Pour effacer un état C, D ou S de la configuration, un autre de ces état est nécessaire (règle (13)) mais le second reste dans la configuration. Par suite, il y aura toujours au moins un d'entre eux.
3. Cette formule est vérifiée pour la configuration initiale. Les règles (1) et (9) ajoute 1 aux valeurs $|c|_{E'_0,E'_1}$ et $|c|_{C_0,C_1}$. La règle (5) ajoute 1 à $|c|_{D_0,D_1}$ et retire 1 de $|c|_{C_0,C_1}$; la règle (14) ajoute 2 à $|c|_{D_0,D_1}$ et soustrait 2 à $|c|_{C_0,C_1}$. La règle (8) soustrait 1 à $|c|_{E'_0,E'_1}$ et $|c|_{D_0,D_1}$. Les quatre autres règles n'affectent pas ces valeurs. Par suite, la propriété est vérifiée à tout instant.
4. Pour passer dans l'état D_i, un agent doit appliquer les règles (5) ou (14).
   — Si cet état est apparu après l'application de la règle (5), alors juste avant l'application la configuration contenait au moins un état E et un état C. Du point 3, nous savons que si il existe un état C alors il existe au moins un état E'. Dans la configuration il y a donc au moins un état E et un état E', c'est-à-dire deux agents porteurs d'états extrémité.
   — Si cet état est apparu après l'application de la règle (14), alors juste avant l'application la configuration contenait au moins deux états C. Du point 3, nous savons que s'il y a deux états C, alors il y a au moins deux états E', c'est-à-dire deux états extrémité, dans la configuration.

■

Lemme 3.2.3

1. *Si un agent dans un état extrémité a deux voisins dans des états non-extrémaux, alors la configuration est instable. Plus formellement, si e_j est un état extrémité et x_i, v_k des états non extrémaux, alors il existe f_p un état extrémité et y_q, w_r des états non-extrémaux tels que $\ldots xev \cdots \to^* \ldots fyw \ldots$.*

2. Si deux états extrémité sont adjacents, alors la configuration est instable.

Preuve.

1. La preuve s'effectue avec une étude de cas.

   1- x_i **et** e_j **ne sont pas synchronisés.** Dans ce cas, une application de la règle (11) sur la paire (e_j, x_i) suffit pour obtenir le résultat avec $v_k = w_r$.

   2- e_j **est synchronisé avec** x_i **et** v_k**.** Si $(e_j, v_k) \in \{(E_0, C_0), (E_1, C_1)\}$, nous sommes en présence de deux cas spéciaux respectivement résolu par les exécutions suivantes :

   $$\begin{array}{ll} \ldots x_0 E_0 C_0 \ldots & \ldots x_1 E_1 C_1 \ldots \\ \ldots x_0 \overline{E_0 D_1} \ldots & \ldots x_1 \overline{E_0 D_1} \ldots \\ \ldots \overline{y_1 g_1} D_1 \ldots & \ldots \overline{E_1 x_1} D_1 \ldots \\ \ldots y_1 \overline{f_0 w_0} \ldots & \\ \ldots \overline{f_0 y_0} w_0 \ldots & \end{array}$$

   où g et f sont des états extrémité et w un état non extrémité. Sinon, l'application d'une règle parmi les règles (4) et de (6) à (10) sur (e_i, v_k) produit une configuration correspondant premier cas.

   3- e_j **est synchronisé avec** x_i **mais pas avec** v_k**.** L'application d'une règle parmi les règles de (4) à (10) transforme la paire (e_j, x_i) en $(g_{i'}, z_{j'})$. Si $(e_j, v_k) \in \{(E_0, C_0), (E_1, C_1)\}$, alors il suffit d'appliquer la règle (11). Sinon, la configuration correspond en fait au cas précédent.

2. Si au moins un des états extrémité n'est pas marqué, alors en appliquant les règles (1) ou (2) l'un des deux est effacé. Par suite, la configuration était instable.

   Si les deux sont dans un état marqué, alors il y a au moins deux agents dans un état C ou D. Cela signifie qu'il y a au moins 4 agents dans la chaîne et donc l'un des agents portant un état extrémité a deux voisins, dont un qui n'est pas dans un état extrémité marqué. En utilisant les règles (3), (4), (6), (8) et (10), l'agent portant cet état extrémité marqué change d'état de façon à ne plus être synchronisé avec son voisin non marqué. La configuration est donc instable.

■

Il peut être déduit du lemme 3.2.3 qu'un état extrémité appartient à une configuration stable uniquement s'il est à l'une des extrémités de la chaîne et qu'il est synchronisé avec son voisin, qui ne doit pas être un état extrémité.

Lemme 3.2.4 *Soit une configuration c accessible.*
Si $|c|_{D_0,D_1} \geq 1$*, alors c n'est pas une configuration stable.*

Preuve. Nous allons prouver que s'il existe un agent dans un état D alors il existe un état extrémité qui peut se déplacer.

Puisque la configuration est accessible et puisque il y a un état D dedans, le lemme 3.2.2.4 nous dit qu'il y a au moins deux états extrémité.

Le lemme 3.2.3 nous dit que les places où un état extrémité peut être stable sont les agents à l'extrémité de la chaîne. Par suite, s'il y a trois états extrémité (ou plus) dans la configuration, elle ne peut pas être stable.

Supposons qu'il y a exactement deux états extrémité et qu'ils sont sur les agents aux extrémité de la chaîne.

Puisqu'il y a un état D, au moins un des états extrémité est marqué. Puisqu'il n'y a que deux états extrémité, il ne peut y avoir au plus qu'un seul autre état C ou D dans la configuration, d'après le lemme 3.2.2.3. Sans perte de généralité, nous pouvons supposer que tout les états S ont été supprimés de la configuration grâce aux règles (12) et (13). Nous avons maintenant 3 cas à étudier.

1- Il n'y a pas d'autre état D ou C Un état extrémité est dans un état E, l'autre est dans un état E. En utilisant la règle (12), l'état D peut se déplacer jusqu'à être à coté de l'état E'. Si ces deux états ne sont pas synchronisés, alors la configuration est instable. Sinon nous pouvons appliqué les règles (8) et (9). Cela crée un état C qui peut se déplacer jusqu'à l'état E, grâce à la règle (12). Ensuite une application des règles (5) puis (11) fait bouger l'état extrémité.

2- Il y a un autre état D Les deux états extrémité ont des états E' Un état D peut se déplacer pour se trouver à coté d'un état E'. Après application de la règle (8), l'état D se transforme en état S qui peut se déplacer jusqu'à se trouver à coté de l'autre état D. Une application de la règle (13) permet d'effacer l'état S et ainsi la configuration obtenue correspond au premier cas.

3- Il y a un état C Les deux états extrémité ont des états E' Un état D peut se déplacer pour se trouver à coté d'un état E'. Après application de la règle (8), l'état D se transforme en état S et l'état E' devient un état E. L'état S créé peut se déplacer jusqu'à se trouver à coté de l'autre état D. Une application de la règle (13) permet d'effacer l'état S. Finalement, l'état C se déplace pour

être adjacent à l'état E et une application des règles (5) puis (11) fait bouger l'état extrémité.

■

Proposition 3.2.5 *Les seules configurations accessibles qui soient stables du protocole Extremity sont les configurations c telle que :*

$$c \cong E_0 M_0 \dots M_0.$$

Preuve. À partir des lemmes précédents, nous pouvons déduire ce qui suit.

Si la configuration est accessible depuis la configuration initiale composée uniquement d'états E_0, alors il y a dans la configuration au moins un état extrémité (lemme 3.2.2). S'il y a au moins deux états extrémité, alors un état D peut être généré et donc la configuration n'est pas stable (lemme 3.2.4). Par suite, une configuration stable et accessible ne peut avoir qu'un unique état extrémité. Par ailleurs, cet état doit se trouver à l'extrémité de la chaîne (lemme 3.2.3).

Si l'état extrémité n'est pas synchronisé avec son voisin, la configuration n'est pas stable (par application de la règle (11)). Par conséquent, une configuration stable et accessible doit nécessairement être d'une des formes suivantes :

$$E'_i M_i w_c \text{ ou } E'_i C_i w_m \text{ or } E_i M_i w_s \text{ or } E_i S_i w_m$$

où $i = 0, 1$, $w_c \in \{M_0, M_1\}^*.(C_0{+}C_1).\{M_0, M_1\}^*$, $w_s \in \{M_0, M_1\}^*.(S_0{+}S_1).\{M_0, M_1\}^*$ et $w_m \in \{M_0, M_1\}^*$.

Tant qu'il y a plusieurs états extrémité, il est possible de faire bouger l'un d'eux, et donc de les mettre en contact. Si un état E interagit avec un état extrémité (E ou E'), il est immédiatement effacé. S'il n'y a que des états E', et au moins deux, alors il y a dans la configuration, ou à défaut il peut apparaître, un état D. L'état D peut se déplacer pour interagir avec un état E' et lui enlever sa marque, le transformant en état E. De cette façon, tant qu'il existe au moins deux états extrémité dans la configuration, il est possible de se ramener à un cas où il est possible d'en effacer un.

Lorsqu'il n'y a qu'un seul état extrémité, il ne peut y avoir qu'au plus un état C ou D. Les états S peuvent supprimés jusqu'à ce qu'il ne reste qu'un seul état S, C ou D.

Finalement, les seules configurations stables et accessibles sont les suivantes :

$$E'_i M_i w_c \text{ ou } E'_i C_i w_m \text{ ou } E_i M_i w_s \text{ ou } E_i S_i w_m.$$

■

Nous avons donc construit le protocole $Extremity = (Q_{Extremity}, \delta_{Extremity})$.

Soit $\psi_{Ext}(\omega)$ le prédicat valant vrai pour une configuration ω si et seulement si la configuration ω contient exactement un état extrémité et si c'est l'état d'un agent situé à l'extrémité de la chaîne. De même, nous posons $\psi_{Ext2}(\omega)$ le prédicat valant vrai pour une configuration ω si et seulement si exactement deux états de travail de ω sont des états extrémité distingués, et que ces états extrémité sont situés chacun sur un agent situé à une extrémité différente de la chaîne.

À partir du protocole $Extremity$, nous pouvons donc construire les protocoles avec environnements

$$Extremity_1 = (Q_{Extremity}, \Sigma, \delta_{Extremity1})$$

$$\text{et} \qquad Extremity_2 = (Q_{Extremity}, Q_{Extremity} \times \Sigma, \delta_{Extremity2} \cup \epsilon).$$

Chacun sur sa composante, les deux travaillent comme $Extremity$, sauf qu'une règle ajoutée rend instable les situations où les deux composantes détectent la même extrémité. Chaque agent garde son entrée dans son environnement, même si elle n'est pas utilisée par aucun de ces deux protocoles, mais elle reste disponible pour d'autres protocoles.

Ces protocoles vérifient respectivement les contraintes de progression $(\theta_1, \sigma_1, \psi_1)$ et $(\theta_2, \sigma_2, \psi_2)$ avec :

$$\theta_1 = \theta_2 = \forall x,\ x : E_0$$

$$\sigma_1 = true \text{ et } \sigma_2 = true \| \psi_{Ext}$$

$$\psi_1 = \psi_{Ext} \text{ et } \psi_2 = \psi_{Ext2}.$$

Nous pouvons maintenant prouver le lemme 3.2.1 dont nous rappelons l'énoncé :

Lemme 3.2.1 *Il existe un protocole travaillant sur chaînes qui, sous réserve d'équité, marque les agents extrémaux de la chaîne, en distinguant une extrémité de l'autre.*

Les protocoles $Extremity_1$ et $Extremity_2$ vérifient les hypothèses de la proposition 3.2.1. Par construction, le protocole $Organize = Extremity_1 \| Extremity_2$ identifie chaque extrémité de la chaîne.

Pour une lecture plus facile de la suite du chapitre, les états extrémité de $Extremity_1$ sont renommés L pour "left extremity" et ceux de $Extremity_2$ sont renommés R pour "right extremity". Tout les autres états sont regroupés sous la notation m.

3.2.3.2 Faire les cent pas

Nous continuons à définir les briques que nous composerons pour obtenir la simulation d'une machine de Turing. Le protocole de population suivant travaille avec environnement et leader. Le leader sera étiqueté H (pour "head", la tête de la machine de Turing) au lieu de L, déjà utilisé pour représenté une des extrémité.

D'abord, nous définissons le protocole

$$Pendulum = (Q_{Pendulum}, Q_{env}, \delta_{Pendulum})$$

par son ensemble d'états de travail $Q_{Pendulum} = \{H_R, H_L, R, r, L, l, R_H, L_H\} \times \Sigma$, par son ensemble d'états d'environnement $Q_{env} = \Sigma \times \{R, m, L\}$, et par ses règles de transitions $\delta_{Pendulum}$ qui décrivent le comportement suivant pour le jeton tête (leader). Le jeton démarre dans un état H_R et se déplace jusqu'à être sur un agent ayant R dans sont environnement. La tête passe alors dans l'état R_H. Ensuite la tête passe dans l'état H_L et recommence à se déplacer, en quittant l'agent extrémal, elle passe cet agent dans l'état R. Lorsque la tête quitte un agent non-extrémal, celui-ci passe dans l'état r, et ce jusqu'à ce que la tête atteigne l'autre extrémité, qui a L dans son environnement. À ce moment, la tête passe dans l'état L_H. Durant ses déplacements la tête recopie l'entrée qui est dans l'environnement sur la seconde composante de l'espace de travail. En cas de problème (deux têtes ou bien un changement d'environnement dû au fait que les protocoles calculant les extrémités n'ont pas finis leur calcul), la situation problématique est détruite et un jeton H_R est généré.

Le protocole $Pendulum$ satisfait les contraintes de progression

$$(\{\exists x,\ x : H_R\}\ ,\ true \| 1_{Rm^*L}\ ,\ 1_{Rr^*L_H})$$

où $1_{\mathcal{E}}$ est un prédicat qui est vrai si et seulement si la configuration correspond à un mot vérifiant l'expression régulière $\mathcal{E}$.

3.2.3.3 Simuler une machine de Turing

Nous sommes maintenant prêt à relier les langages calculés par les protocoles de populations sur des chaînes aux machines de Turing et prouver le théorème 1.

Nous rappelons que la classe ZPSPACE (s) correspond aux langages acceptés par les machines de Turing Probabilistes avec Erreur Zéro utilisant une mémoire bornée par s. Ce sont des machines de Turing M qui répondent Oui, Non ou $Peut\text{-}etre$ et qui vérifient les deux conditions suivantes :

(i) si x appartient au langage reconnu par M,
alors, avec x en entrée, M répondra Oui avec probabilité non-nulle mais ne répondra jamais Non.

(ii) si x n'appartient pas au langage reconnu par M,

alors, avec x en entrée, M répondra Non avec probabilité non-nulle mais ne répondra jamais Oui.

La définition formelle de cette classe est donnée plus en détail au chapitre 2.3.2.

Nous rappelons l'énoncé du théorème 1 :

Théoreme 3.2.1 *Un langage L est calculable par un protocole de population travaillant sur les chaînes si et seulement si L est un langage symétrique de* $\mathrm{NSPACE}(n)$.

En d'autres termes :

$$PP(C_n) = \mathrm{NSPACE}_{\mathrm{sym}}(n) = \mathrm{ZPSPACE}_{\mathrm{sym}}(n).$$

Nous allons d'abord montrer le sens indirect $PP(C_n) \supset \mathrm{ZPSPACE}_{\mathrm{sym}}(n)$ puis le sens direct $PP(C_n) \subset \mathrm{NSPACE}_{\mathrm{sym}}(n)$.

3.2.3.4 Preuve du sens indirect du théorème 1

Le sens indirect se démontre en utilisant les constructions des sections précédentes. Étant donné un langage de $\mathrm{ZPSPACE}_{\mathrm{sym}}(n)$ reconnu par une machine de Turing M, nous construisons le protocole de population

$$Protocol(M) = (Q_{Protocol(M)} \times Q_{env}, \delta_M)$$

qui reconnaît ce langage en travaillant sur des chaînes de longueur n. En composant les protocoles construits précédemment, nous obtenons un Protocole qui organise une chaîne en un ruban de machine de Turing de n cases. Nous supposerons que l'entrée est dans l'environnement et est accessible par chacun des protocoles composés, et que l'environnement contient aussi les informations sur où sont les extrémités.

Les états de travail sont divisés entre information géographique — si l'agent héberge la tête, si l'agent est à l'extrémité gauche ou droite, si l'agent est à gauche ou à droite de la tête — état de l'automate de la machine de Turing simulée et case du ruban de la machine de Turing.

Les règles de transition δ_M sont construites à partir des règles de transition de la machine M. Par exemple, une règle $(q, e) \to (q', e', +1)$ est transformée en $\begin{pmatrix} \epsilon_1 \\ (H, q, e) \end{pmatrix} \begin{pmatrix} \epsilon_2 \\ (r, \varnothing, f) \end{pmatrix} \to (l, \varnothing, e')\ (H, q', f)$ où ϵ_i est l'environnement.

Nous appellerons ω_{input} le mot passé en entrée est stocké sur la première composante de l'environnement. Nous écrirons $M(\omega_{input})$ pour représenter le résultat du calcul de la machine de Turing M sur le mot ω_{input}.

Nous définissons ψ_M comme le prédicat sur les configurations tel que $\psi_M(C) = 1$ si et seulement si la configuration C correspond à l'un des deux cas suivants :

(i) la tête de la machine de Turing est dans un état final acceptant et $M(\omega_{input}) = 1$

(ii) la tête de la machine de Turing est dans un état final rejetant et $M(\omega_{input}) = 0$.

Le protocole $Protocol(M)$ satisfait les contraintes de progressions

$$(1_{Rr^*Hl^*L+R_Hl^*L+Rr^*L_H}\,,\, true \| 1_{Rm^*L}\,,\, \psi_M)\,.$$

Ce protocole est un protocole avec leader, ainsi que le protocole $Pendulum$ de la section précédente. La proposition 3.2.2 nous permet de les composer séquentiellement. Nous obtenons alors le protocole avec leader

$$\begin{aligned} Pendulum\,;P(M) \;=\; \Big(&\{H_R, H_L, R, r, L, l, R_H, L_H\} \\ &\cup(\{H, l, r, L, R\} \times (Q_M \cup \{\varnothing\}) \times \Sigma), \\ &\Sigma \times \{R, m, L\}, \\ &\varphi_{Pendulum}?\delta_{Pendulum} : (\varphi_{P(M)}?\delta_M : Id)\Big) \end{aligned}$$

qui satisfait les contraintes de progressions $(\exists x.\{x : H\}\,,\, true \| 1_{Rm^*L}\,,\, \psi_M)$.

Nous voulons maintenant composer en parallèle ce protocole avec le protocole $Organize = Extremity_1 \| Extremity_2$. Nous allons devoir utiliser l'opération $[\,\cdot\,]_{True}$ sur le protocole $Pendulum; P(M)$ afin d'être sûr que, ultimement, ce protocole soit exécuté lorsque le protocole $Organize$ s'est stabilisé. Le protocole $Pendulum; P(M)$ vérifie la condition de persistance, puisque la tête n'est jamais détruite. Il est donc possible de faire boucler ce protocole.

Nous obtenons finalement un protocole de population pour simuler une machine de Turing M et qui peut s'écrire sous la forme

$$Extremity_1 \| Extremity_2 \| [Pendulum; Protocol(M)]_{True}$$

et de plus il vérifie les contraintes de progression suivantes :

$$(\exists x.x : R \| \exists y.y : L \| \exists z.z : H, true, \psi'_M)$$

avec $\psi'_M(C_1, C_2, C_M, C_i) = 1$ si et seulement si nous avons $\psi_M(C_M, (C_1, C_2, C_i)) \wedge ((\psi_{Ext}(C_1, C_2, C_i)) \| \psi_{Ext}(C_1, C_2, C_i)) \wedge (\nexists x \in Q^2_{Extremity}, x \cong (E_0, E_0)))$. Cette construction est résumée par la figure 3.4.

Soit $L \in \mathrm{ZPSPACE}_{\mathrm{sym}}(n)$. En utilisant la simulation d'une machine de Turing décrite ci-dessus, nous obtenons un protocole de population travaillant sur une chaîne

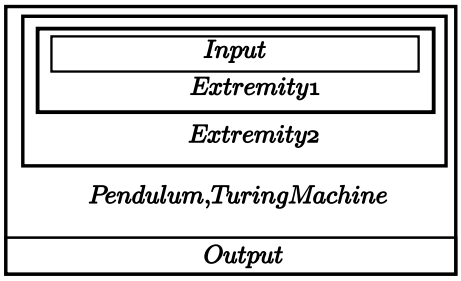


FIGURE 3.4 – Organisation du protocole de population simulant une machine de Turing.

pouvant simuler une machine de Turing M calculant L en remplaçant les choix probabilistes par des choix non-déterministes. Pour une entrée x donnée, la machine M ou bien répond "*Peut-etre*", ou bien donne une réponse sûre, et juste. Mais il existe au moins une exécution répondant cette réponse sûre et juste. Le protocole simulant M exécutera des exécutions possibles de M, et, par équité, il exécutera infiniment souvent une exécution renvoyant une réponse sûre. À partir d'un certain instant, toute les exécutions simulées sont correctes, au sens où elles correspondent à des exécutions possibles de M sur l'entrée x. À partir de cet instant, la simulation ne retourne comme réponse que la réponse juste ou "*Peut-etre*". Par suite, en retenant en permanence la dernière réponse sûre calculée, le Protocole simulant M renvoie toujours la réponse juste, à partir d'un certain moment.

3.2.3.5 Preuve du sens direct du théorème 1

Pour prouver qu'un langage calculé par un protocole de population appartient à la classe NSPACE (n), nous allons nous appuyer sur les faits suivants :

1. Savoir si une configuration de taille n est stable en sortie est un problème dans co-NSPACE (n).
2. co-NSPACE (n) = NSPACE (n).
3. Obtenir une configuration stable en sortie à partir d'une configuration initiale de taille n est dans NSPACE (n).

Pour le premier point, remarquons qu'une configuration stable en sortie est une configuration telle qu'il n'existe pas de configuration accessible avec une sortie différente. Par suite, ce problème est résolu par une machine de Turing de co-NSPACE (n).

Le second point a été démontré par Immerman dans [Imm88] et déjà rappelé dans le chapitre 2.3.1. Ainsi, le premier point peut se réécrire : savoir si une configuration

de taille n est stable en sortie est un problème dans NSPACE (n).

Le troisième point est un problème résoluble par une machine de Turing de NSPACE (n) puisque le problème est de trouver une exécution menant vers une configuration stable en sortie — propriété qui est reconnaissable par une machine de NSPACE (n).

Pour résumer, savoir si un mot d'entrée sera reconnu par un protocole de population est calculable par une machine de Turing non déterministe utilisant un espace linéaire.

3.3 Résumé et conclusion

Dans ce chapitre, nous avons donné deux constructions d'une simulation de machine de Turing sur un protocole de population travaillant sur un graphe réduit, chacune s'appuyant sur des approches un peu différentes. Les deux approches permettent démontrer l'équivalence entre les protocoles de population travaillant sur un graphe à degré borné par une constante et le modèle des machines de Turing non-déterministes utilisant un espace linéaire.

Notre approche repose sur un formalisme différent du formalisme classique des protocoles de population. Ce formalisme permet une plus grand modularité et également de définir formellement des protocoles couramment utilisé comme celui de l'élection d'un leader au sein de la population. Dans le chapitre suivant, nous utilisons notre formalisme pour construire une hiérarchie de modèle.

Chapitre 4

Une Hiérarchie étendue de Protocoles de Population

Dans ce chapitre, nous étendons les résultats précédents sur la simulation de machines de Turing pour construire toute une hiérarchie de modèles. Pour cela, nous considérons maintenant des protocoles de population utilisant des agents ayant une mémoire qui n'est pas forcément bornée par une constante.

Par ailleurs, nous n'allons pas nous restreindre aux protocoles calculant sur des chaînes (ou des graphes ayant un degré borné) ou des cliques, mais sur des familles de graphes intermédiaires. Nous allons utiliser des familles de graphes particulières qui nous permettront de construire une hiérarchie en réutilisant les constructions du chapitre précédent.

Nous obtenons ainsi la hiérarchie suivante, avec s_1 et s_2 qui bornent l'espace de calcul disponible sur chaque agent et avec s'_1 et s'_2 qui bornent la taille d'un certain sous-graphe du graphe de communication sous-jacent :

Théoreme 3 *Soient s_1, s_2, s'_1, s'_2 telles que $s_i(n) = O(n)$ et $s'_i(n) = O(\log\log n)$ pour $i = 1, 2$.*

Si $s'_1 \cdot s_1 = o(s'_2 \cdot s_2)$, alors

$$L_{s'_1}(s_1) \subseteq L_{s'_2}(s_2).$$

Et l'inégalité est stricte si $s'_2(n) \cdot s_2(n) = \Omega(\log n)$.

Les résultats de ce chapitre ont été publiés dans [BL13a, BL13b].

4.1 La variante considérée

En réalité, la variante considérée dans ce chapitre est une variante hybride entre la restriction du graphe de communication et les Passively Mobile Machines. Dans chaque cas, des constructions de hiérarchies ont été proposées en faisant varier un paramètre (taille d'un sous-graphe ou espace alloué à chaque agent). Cependant, dans les deux cas, il existe un intervalle de valeur pour le paramètre où il n'y a plus de caractérisation précise de la puissance de calcul du modèle. L'idée est de combiner les deux variantes afin d'avoir une hiérarchie plus étendue.

Nous définissons d'abord les familles de graphes qui serviront de graphe de communication à nos protocoles de population.

Définition 21 (Famille de graphes (s,d)-séparable) *Une famille de graphe* $\Big(G(n) = (V(n), E(n))\Big)_{n\in\mathbb{N}}$ *est* (s,d)-séparable *si pour tout n, l'ensemble de sommets $V(n)$ possède n sommets et peut être partitionné en U_1 et U_2 (U_2 peut être vide) tels que :*

- *$(U_1, E(n)_{|U_1})$ est un graphe connexe avec $s(n)$ sommets*
- *pour tout sommet $u \in U_1$, son degré (dans $G(n)$) est borné par $d(n)$*
- *pour tout sommet $v \in U_2$, v appartient à une clique de taille au moins $2^{d(n)+1}$.*

Cela impose les conditions $d(n) \leq n-1$ et $s(n) \leq n$, sinon la famille de graphes (s,d)-séparable est vide Les sous-graphes U_2 des graphes d'une famille (s,d)-séparable peuvent être vides si $d(n) > \log(n)-1$, dans ce cas là nous décrivons juste une famille de graphe dont le degré est borné par $d(n)$.

Par exemple pour $d(n) = 2$, un graphe (s,d)-séparable sera composé d'un chemin de taille $s(n)$ rattaché (par un ou deux sommets) à une clique d'au moins 8 sommets. La figure 4.1 montre un possible graphe d'une famille $s, 2$-séparable, et la figure 4.2 un graphe d'une famille $(s,4)$-séparable.

Ces familles de graphes sont intéressantes parce qu'il est possible de construire un protocole qui, pour tout graphe de la famille, *sépare* le sous-graphe couvert par des grandes cliques du sous-graphe ayant un degré borné.

Lemme 4.1.1 *Soit $G = (V, E)$ un graphe (s,d)-séparable avec $d = O(1)$.*

Il existe un protocole de population Clique qui distingue U_1 de U_2.

Cela signifie que l'ensemble des états peut être partitionné en Q_{clique} et $Q_{bounded}$ tels que pour toute exécution équitable (C_t) du protocole Clique, il existe t_0 tel que, pour tout $t \geq t_0$, un agent de la configuration C_t possède un état de Q_{clique} si et seulement si il appartient à U_2.

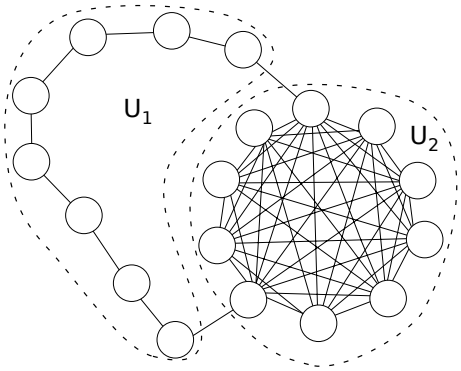


FIGURE 4.1 – Un graphe d'une famille $s, 2$-séparable

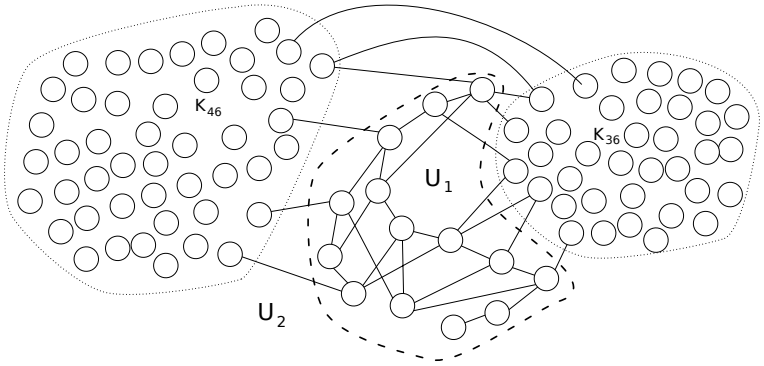


FIGURE 4.2 – Un graphe d'une famille $s, 4$-séparable

Preuve. Nous construisons le protocole *Clique* via deux routines.

La première routine $Color_{d+1}$ colorie les agents avec $d+1$ couleurs de telle manière qu'un agent qui appartient à une clique de taille 2^{d+1} a au moins un voisin dans chacune des couleurs.

Il est clair qu'un agent qui appartient au sous-graphe de degré borné par d ne peut pas avoir $d+1$ couleurs distinctes dans son voisinage. Ici, le coloriage est utilisé pour avoir une idée du degré de l'agent. En particulier, notre coloriage autorise deux agents voisins à avoir la même couleur. Un agent peut aussi avoir plusieurs voisins ayant la même couleur. Notre coloriage ne possède aucune contrainte autre que le fait qu'un sommet a une seule couleur.

L'ensemble des états est les $d+1$ couleurs et des états candidats pour chacune

d'entre elles :

$$Q_{Color_{d+1}} = \{i \mid 1 \leq i \leq d+1\} \cup \{0_i \mid 1 \leq i < d+1\}.$$

Les règles de transition se résument facilement : si deux candidats à la même couleur i — *i.e.* deux agents dans un même état 0_i — interagissent, alors un agent prend la couleur i et l'autre se porte candidat à la couleur $i+1$. Cette règle est légèrement modifiée pour le cas où $i = d$ puisqu'il n'y a pas de candidat à $d+1$: si deux candidats pour d interagissent, un se colore en d et l'autre se colore en $d+1$. Cela nous donne donc l'ensemble de règles :

$$\delta_{Color_{d+1}} = \{0_i\ 0_i \to 0_{i+1}\ i \mid 1 \leq i < d\} \cup \{0_d\ 0_d \to d\ d+1\}$$

Considérons des agents se trouvant dans une clique de taille 2^{d+1}. La moitié d'entre eux seront colorés avec la couleur 1. L'autre moitié se portera candidat à la couleur 2. La moitié de ces derniers (un quart du total donc) sera coloré avec 2. Le reste se portera candidat à la couleur 3 ; et ainsi de suite. Il apparaît donc que pour toute couleur $i \leq d$ on aura $2^{d+1-i} \geq 2$ agents arborent la couleur i. Les deux agents restants — il reste en effet $2^{d+1} - \Sigma_{i \leq d} 2^{d+1-i} = 2$ agents — arborent la couleur $d+1$. On voit donc que quelque soit la couleur d'un agent, il y aura au moins un de ses voisins qui sera de la couleur i, pour tout $i \leq d+1$.

La seconde routine $Count_{d+1}$ compte (jusqu'à $d+1$) combien de couleurs différentes il y a dans le voisinage de chacun des agents. Ainsi, on peut décider si un agent est dans le sous-graphe de degré borné — un agent dans ce sous-graphe ne peut pas avoir $d+1$ différentes couleurs dans son voisinage — ou est dans une grande clique — les agents dans ces cliques ont $d+1$ couleurs dans leur voisinage.

La routine $Count_{d+1}$ teste pour chaque agent la présence des couleurs $1, 2, \ldots, d, d+1$. Cette routine travaille avec $Color_{d+1}$ comme environnement. Les états des agents sont donc divisés en deux composantes : sur la première la routine $Color_{d+1}$ travaille, et sur la seconde l'agent enregistre quelles couleurs sont présentes dans son voisinage. Formellement, nous avons donc l'ensemble d'état

$$Q_{Count_{d+1}} = \left\{(i,j) \;\middle|\; i \in Q_{Color_{d+1}}, 0 \leq j \leq d+1\right\}$$

et l'ensemble de règles

$$\delta_{Count_{d+1}} = \left\{(i,j)\ (k,l) \to (i',j)\ (k',l) \;\middle|\; i\ k \to i'\ k' \in \delta_{Color_{d+1}}\right\}$$
$$\cup\{(i,j)\ (j+1,l) \to (i,j+1)\ (j+1,l)\}.$$

Il est important de noter qu'une fois qu'un agent a choisi une couleur dans $Color_{d+1}$ il ne changera pas de couleur, il sera toujours de cette couleur. Donc, si à un moment

donné, un agent voit un de ses voisins coloré avec i, alors cette couleur sera toujours présente dans son voisinage. Si les $d+1$ couleurs sont présentes à un moment donné dans le voisinage d'un agent, alors la routine $Count_{d+1}$ finira par tous les voir, et ce à grâce à la propriété d'équité.

L'ensemble des états $Q_{Count_{d+1}}$ peut être partitionné en deux sous-ensembles : $Q_{clique} = \left\{(i, d+1) \;\middle|\; i \in Q_{Color_{d+1}}\right\}$ et $Q_{bounded} = \left\{(i,j) \;\middle|\; i \in Q_{Color_{d+1}} \wedge j < d+1\right\}$. Tout agent appartenant à une grande clique finira par être dans un état de Q_{clique}, tandis que tout agent appartenant à un sous-graphe de degré borné restera toujours dans un état de $Q_{bounded}$. Et une fois dans un état de Q_{clique}, il n'est pas possible pour un agent de retourner dans un état de $Q_{bounded}$.

■

Lemme 4.1.2 *Soit G un (s,d)-séparable graphe avec $s(n) = \Omega(\log(n))$.*

Il existe un protocole de population $WriteInput$ qui organise le sous-graphe de degré borné en ruban de machine de Turing et qui écrit (en binaire) le multi-ensemble correspondant à l'entrée initiale sur le ruban.

Preuve. Le protocole $WriteInput$ utilises plusieurs routines.

D'abord, *Clique* (du lemme 4.1.1) reconnaît le sous-graphe de degré borné. Ensuite, les routines décrites dans les propositions 3.1.1 et 3.1.2 organisent le sous-graphe de degré borné en ruban de machine de Turing.

La routine suivante écrit le multi-ensemble correspondant à l'entrée (en binaire) sur le ruban. Elle travaille avec une copie de l'entrée. Cette copie va être modifiée : les lettres seront permutées (ce qui ne change fondamentalement pas l'entrée) et les lettres pourront être marquées (pour chaque $\sigma \in \Sigma$, il existe une lettre $\bar{\sigma}$).

Cette routine que nous allons décrire ci-après se réinitialise dès que les routines qui organisent le sous-graphe de degré borné en ruban font un pas. Nous supposerons donc que le graphe est déjà organisé : les agents appartenant à une grande clique sont marqués comme tels, les agents formant le sous-graphe de degré borné sont organisés en arbre couvrant qui est organisé en ruban de machine de Turing et un agent porte la tête de la machine de Turing — cela peut être fait en utilisant les résultat du chapitre précédent. Nous allons décrire le programme exécuté par la tête.

Sur le ruban, il y a un compteur temporaire et un compteur permanent pour chaque lettre de l'alphabet d'entrée Σ. Chaque compteur est initialisé à 0.

Dans un premier temps, la tête passe dans un état qui libère des jetons *swap* dans le graphe entier. Lorsque deux agents interagissent et que l'un d'eux est porteur d'un jeton "swap", ils échangent leur entrée. De cette manière chaque lettre de l'entrée peut se retrouver sur n'importe quelle agent au cours de l'exécution. Lorsque la tête

de la machine de Turing rencontre une lettre d'entrée non marquée σ, elle incrémente le compteur temporaire associé à σ et elle marque la lettre (qui devient alors $\bar{\sigma}$).

Après un certain temps, tout les jetons *swap* sont de retour à la tête. La tête rentre alors dans la seconde phase de son exécution et effectue l'opération suivante pour chaque lettre $\sigma \in \Sigma$: si la valeur du compteur temporaire associé à σ est plus grande que la valeur du compteur permanent, alors le compteur permanent prend la valeur du compteur temporaire, ensuite le compteur temporaire est remis à zéro (que sa valeur ait été sauvegardée ou non).

Dans un troisième temps, la tête envoie sur le graphe des jetons *unmark* qui enlève les éventuelles marques sur les lettres de l'entrée dans la population.

Une fois ces jetons de retour à la tête, la tête recommence du début et boucle sur ces trois temps.

Cette routine vérifie de bonnes propriétés qui assure son fonctionnement :

— il n'est pas possible qu'un compteur temporaire compte trop d'une lettre, et donc le compteur permanent stocke une valeur qui est inférieure ou égale à la bonne.

— à chaque instant il est possible que les interactions soient telles qu'un compteur temporaire compte le bon nombre d'occurrences de sa lettre.

La première propriété est assurée par le marquage des lettres comptées. Il n'est pas possible de compter deux fois une lettre. Éventuellement une lettre ne sera pas comptée parce que sa marque n'aura pas été effacée depuis un tour antérieur ou bien parce que la bonne série d'échange n'a pas eu lieu et qu'elle n'a pas rencontré la tête, mais aucune lettre ne peut être comptée deux fois.

La seconde propriété vient du fait que l'exécution finie suivante est possible à tout instant : le cycle en cours termine et les jetons *unmark* enlèvent toutes les marques ; les jetons *swap* réalisent une série d'échanges dans la population de manière à ce que toutes les lettres de l'entrée rencontrent la tête ; les valeurs des compteurs temporaires sont recopiées dans les compteurs permanents correspondants. Cette exécution mène à une configuration où chaque compteur permanent contient la valeur correcte. Par ailleurs, cette valeur ne peut plus être changée : la valeur d'un compteur permanent ne peut changer qu'en augmentant, or avec ce jeu de valeurs, les compteurs ont atteint leur valeur maximale.

La propriété d'équité nous garantit qu'une telle configuration finira par être atteinte.

Pour conclure, cette routine finira par écrire sur le ruban de la machine de Turing simulée le multi-ensemble correspondant à l'entrée distribuée sur la population.

Pour mener à bien tout cela, il faut que le ruban ait assez de cases, c'est-à-dire qu'il soit de taille $\Omega(\log n)$.

■

Théoreme 4.1.1 *Pour toute fonction $s(n) = \Omega(\log n)$, la classe des langages reconnus $L(s)$ vérifie*

$$L(s) = \text{NSPACE}_{\text{sym}}(s).$$

Ce théorème est une généralisation du théorème 3.2.1. La preuve est construite de la même façon. En posant $s(n) = n$, nous retrouvons exactement le théorème 3.2.1 et sa preuve.

Preuve. Nous fixons une fonction $s(n) = \Omega(\log n)$ et une constante $d = O(1)$.

D'abord nous prouvons que $L(s) \subseteq \text{NSPACE}_{\text{sym}}(s)$.
Une configuration d'une population de taille n sur un (s, d)-séparable graphe requiert $O(s(n) + \log(n - s(n))$ cases sur un ruban de machine de Turing pour être stockée. Puisqu'on a $s(n) = \Omega(\log n)$, l'espace requis est $O(s(n))$ cases. Pour calculer le résultat d'un calcul par protocole de population on a besoin de trouver un configuration telle qu'on ne puisse accéder ensuite à aucune configuration telle que la sortie est différente ou mal-définie. Pour $s(n) = \Omega(\log n)$, on a $\text{NSPACE}(s) = \text{co-NSPACE}(s)$ (voir *e.g.* [Sak96]). Vérifier qu'il n'existe pas de configuration accessible avec une sortie différente est un problème dans $\text{co-NSPACE}(s)$ et trouver une configuration qui vérifie cette propriété est donc un problème d'accessibilité. Nous pouvons donc calculer ce résultat avec une machine de Turing non-déterministe utilisant un espace s sur son ruban, c'est-à-dire appartenant à la classe $\text{NSPACE}(s)$. Et puisque les langages de $L(s)$ sont symétriques, nous avons

$$L(s) \subseteq \text{NSPACE}_{\text{sym}}(s).$$

Nous prouvons maintenant que $\text{ZPSPACE}_{\text{sym}}(s) \subseteq L(s)$.
Soit M une machine de Turing calculant un langage de $\text{ZPSPACE}_{\text{sym}}(s)$. En appliquant directement le lemme 4.1.2, nous obtenons un protocole simulant une machine de Turing utilisant un ruban avec $s(n)$ cases. En réutilisant la construction de la section 3.2.3 d'une machine de Turing par un protocole de population sur une chaîne, nous obtenons ici un protocole de population appartenant à la classe $L(s)$ qui simule la machine de Turing M. Par suite, nous avons

$$\text{ZPSPACE}_{\text{sym}}(s) \subseteq L(s).$$

Comme $s(n) = \Omega(\log n)$, il est connu que (voir *e.g.* [Sak96]) :

$$\text{NSPACE}_{\text{sym}}(s) = \text{ZPSPACE}_{\text{sym}}(s).$$

Ceci permet alors de conclure :

$$L(s) = \mathrm{NSPACE}_{\mathrm{sym}}(s) = \mathrm{ZPSPACE}_{\mathrm{sym}}(s).$$

■

Proposition 4.1.1 *Soit* $s'(n) = \Omega(\log n)$.
L'ensemble des langages calculés par le modèle des Passively Mobil Machines Protocols avec mémoire bornée par s' *est exactement* $\mathrm{NSPACE}_{\mathrm{sym}}(n \cdot s'(n))$.

Preuve. Soit $s'(n) = \Omega(\log n)$.

Considérons le modèle des Passively Mobil Machines Protocols avec une mémoire bornée par s'. Le graphe d'interaction complet peut être simulé sur n'importe quel graphe en ajoutant des règles d'échange. Nous supposerons donc que le graphe d'interaction est complet.

Le protocole $Identify$ suivant assigne un unique identifiant à chaque agent dans la population. Pour le protocole $Identify$, les rubans de chaque agent stockent l'identifiant de l'agent sous la forme d'un entier. On définit les règles de transition :

$$\delta_{Identify} = \{i\ i \to (i+1)\ i \mid\ i \in \mathbb{N}\}.$$

Comme le graphe d'interaction est complet, toute paire d'agents peut interagir. Par suite, si deux agents ont le même identifiant, la propriété d'équité nous assure que la population finira par passer dans une configuration où ils ont chacun un identifiant différent. Et donc si initialement tout les agents ont pour identifiant 0, finalement, lorsque le protocole $Identify$ est stabilisé, chaque agent aura un identifiant différent dans $[\![0, (n-1)]\!]$. L'espace mémoire nécessaire pour stocker un tel identifiant est $\Theta(\log n)$ cases sur le ruban.

Nous avons donc prouvé que si les agents utilise $\Theta(\log n)$ cases sur leur ruban, il est possible d'assigner à chacun un unique identifiant.

Guerraoui *et al.* ont donné dans [GR09] une construction détaillée permettant la simulation d'une machine de Turing calculant un langage de $\mathrm{NSPACE}_{\mathrm{sym}}(n \log n)$ sur une variante des protocoles de population où chaque agent possède un identifiant unique. Nous pouvons réutiliser leur résultat pour achever la preuve : si chaque agent possède au moins $\log(n)$ cases mémoire, alors le modèle est équivalent à une machine de Turing non-déterministe qui dispose d'un espace mémoire égal à la somme des espaces mémoires disponibles sur chacun des agents.

■

Si les agents ont un espace mémoire plus réduit, alors prendre en compte le graphe d'interaction peut permettre de gagner de la puissance de calcul.

Dans ce qui suit, nous considérerons le cas où $s'(n) = o(\log n)$.

Nous noterons $L_{s'}(s)$ la classe des langages calculés par le modèle des Machines Passivement Mobiles avec espace mémoire borné par s' sur une famille de graphe (s, d)-séparable où $d(n) \leq 2^{s'(n)}$.

Les résultats du chapitre 3 correspondent au cas où les agents ont une mémoire bornée par une constante. Ils sont extensibles au cas général où leur mémoire est bornée par s'. Dans ce cas là, les agents peuvent avoir jusqu'à $2^{O(s'(n))}$ "états" différents (en fait ce sont des configurations de machine de Turing).

Nous pouvons utiliser ces états supplémentaires pour construire un coloriage à distance 2 avec plus de couleurs, et ainsi il est possible de construire des arbres couvrant sur des graphes (s, d)-séparables où d n'est plus une constante en fonction de n. Plus précisément, nous obtenons le résultat suivant qui est une extension du lemme 4.1.1.

Lemme 4.1.3 *Soit $G = (V, E)$ un graphe (s, d)-séparable.*

Il existe un modèle de Passively Mobil Machines Protocols avec mémoire bornée par $\log(d(n))$ qui distingue U_1 de U_2.

Nous obtenons ainsi une manière de simuler une machine de Turing sur la partie à degré bornée du graphe. La machine simulée a un ruban réparti sur $s(n)$ agents, chacun simulant $s'(n)$ cases. L'espace mémoire total utilisable par la machine de Turing simulée est donc de $s(n) \cdot s'(n)$ cases.Nous obtenons alors la généralisation de la Proposition 4.1.1 :

Théoreme 4.1.2 *Considérons des fonctions d, s et s' telles que $d(n) = 2^{O(s'(n))}$, $d(n) = O(\log n)$, $s'(n) = o(\log n)$ and $s(n) \leq n$.*

La classe des languages calculables par le modèle des Machines Passivement Mobiles avec mémoire bornée par s' calculant sur une famille de graphes (s, d)-séparables est exactement $\mathrm{NSPACE}_{\mathrm{sym}}(s \cdot s')$. En d'autres termes, nous avons :

$$L_{s'}(s) = \mathrm{NSPACE}_{\mathrm{sym}}(s \cdot s').$$

4.2 La construction de la hiérarchie

De nombreuses hiérarchies ont été construites sur les machines de Turing, nous nous utiliserons la suivante :

Proposition 4.2.1 (Hiérarchie de machines de Turing, voir [Gef03]) *Soit s, s' des fonctions constructibles.*

Si $s(n) = o(s'(n))$, alors

$$\mathrm{NSPACE}\,(s) \subseteq \mathrm{NSPACE}\,(s')\,.$$

Et de plus, si $s(n) = o(s'(n))$ et $s'(n) = \Omega(\log n)$, alors l'inclusion est stricte

$$\mathrm{NSPACE}\,(s) \subsetneq \mathrm{NSPACE}\,(s')\,.$$

Grâce aux Théorèmes 4.1.1 et 4.1.2, il est possible de transposer la hiérarchie des machines de Turing non-déterministes utilisant un espace au moins logarithmique.

Théoreme 4.2.1 (Hiérarchie de Protocoles) *Soient s_1, s_2, s'_1, s'_2 telles que $s_i(n) = O(n)$ et $s'_i(n) = O(\log\log n)$ pour $i = 1, 2$.*

Si $s'_1 \cdot s_1 = o(s'_2 \cdot s_2)$, alors $L_{s'_1}(s_1) \subseteq L_{s'_2}(s_2)$.

Et l'inégalité est stricte si $s'_2(n) \cdot s_2(n) = \Omega(\log n)$.

Par exemple, si nous choisissons $s'(n) = \log\log n$, et que nous considérons la suite de fonctions $s_k(n) = \log^k n / s'(n)$. Pour tout $k \geq 1$, nous avons

$$L_{s'}(s_k) = \mathrm{NSPACE}_{\mathrm{sym}}\left(\log^k n\right).$$

Nous en déduisons alors la hiérarchie suivante :

$$\begin{aligned}
L_1(1) &\subsetneq L_{s'}(1) \\
&\subsetneq L_{s'}(s_1) = L_{s'}\left(\frac{\log n}{\log\log(n)}\right) \\
&\subsetneq L_{s'}(s_2) = L_{s'}\left(\frac{\log^2 n}{\log\log(n)}\right) \\
&\vdots \\
&\subsetneq L_{s'}(s_k) \\
&\vdots \\
&\subsetneq L_{s'}(n) = \mathrm{NSPACE}_{\mathrm{sym}}\left(n \cdot s'(n)\right)
\end{aligned}$$

$L_1(1)$ correspond à ce qu'il est possible de calculer avec le modèle classique : les agents ont $O(1)$ espace mémoire et la partie de degré borné du graphe ne comporte qu'un nombre constant de sommet. $L_1(1)$ est donc la classe des langages semi-linéaires. $L_{s'}(1)$ correspond au modèle des Passivement Mobiles Machines où les

agents disposent d'une mémoire de taille $O(s')$ et où le graphe d'interaction est complet. $L_{s'}(n)$ correspond au modèle des Passivement Mobiles Machines où les agents disposent d'une mémoire de taille $O(s')$ et où le graphe d'interaction est (entièrement) à degré borné.

La preuve de l'inégalité stricte $L_1(1) \subsetneq L_{s'}(1)$ est détaillée dans [CMN+11]. La preuve consiste en l'exhibition d'un langage non-semilinéaire appartenant à $L_{s'}(1)$; l'exemple choisi étant $\{\omega \mid \log(|\omega|_a) = |\omega|_b\}$.

Dans [CMN+11], les auteurs construisent une hiérarchie basée sur s', la taille de la mémoire disponible pour chaque agent. La hiérarchie est exactement caractérisée et stricte pour $s'(n) = \Omega(\log n)$, et pour tout $s'(n) = o(\log \log n)$, le modèle est équivalent au cas où $s'(n) = O(1)$, qui est équivalent au modèle classique des protocoles de population. En utilisant, les restrictions du graphe d'interaction, nous avons pu étendre et préciser cette hiérarchie.

4.3 Résumé et conclusion

Dans ce chapitre, nous avons utilisé les résultats du chapitre précédent afin de construire une hiérarchie de modèles allant des protocoles de population classiques aux machines de Turing non-déterministes utilisant un espace linéaire. Notre construction s'appuient sur deux constructions préexistantes de hiérarchies. Notre hiérarchie peut être vue comme une de chacune de ces deux hiérarchies. Notre hiérarchie est plus précise — elle distingue des cas non-distingué avant — et elle donne une caractérisation exacte de la puissance des modèles dans des cas où les hiérarchies précédentes ne donnaient qu'un encadrement.

Chapitre 5

Classification de Variantes

Dans ce chapitre, nous étudions quelques variantes des protocoles de population en considérant certaines variations sur la forme des règles et, éventuellement, sur l'acceptation du protocole.

Nous allons considérer trois grandes familles de relations de transition.

Définition 22 (Relations de transition) *Nous définissons les trois familles de protocoles de populations suivantes :*

— *Un protocole de population avec des* transitions classiques *est un protocole de population au sens usuel, c'est-à-dire où la relation de transition δ change possiblement les états des deux agents qui interagissent :*

$$\delta \subset Q^2 \times Q^2.$$

Le nombre d'agent est conservé.

— *Un protocole de population* temps réel *est un protocole de population où la relation de transition δ détruit un des deux agents qui interagissent et met à jour l'état de l'autre :*

$$\delta \subset Q^2 \times Q.$$

Le nombre d'agent diminue strictement au cours d'une exécution.

— *Un protocole de population avec des* transitions génératrices *est un protocole de population où la relation de transition δ détruit les deux agents qui interagissent et en créer un certain nombre de nouveaux :*

$$\delta \subset Q^2 \times \bigcup_{n \in \mathbb{N}} Q^n.$$

On suppose que δ est de taille finie.

Nous allons considérer plusieurs manières de lire la sortie sur la configuration finale.

Définition 23 (Sortie d'un Protocole) *Nous définissons trois manières d'obtenir la sortie d'un calcul :*

- *Un protocole utilise une* sortie Consensus *si le résultat de son calcul est obtenu en lisant via la fonction de sortie le résultat renvoyé par une configuration stable en sortie — où tout les agents renvoient la même sortie et telle qu'il n'existe pas de configuration accessible où un des agents renvoie une autre sortie.*
- *Un protocole utilise une* sortie du leader *si un calcul est acceptant si et seulement si , à partir d'un moment, à chaque instant, il existe un agent qui est dans un état acceptant*
- *Un protocole utilise la* sortie du Dernier agent *si c'est un protocole qui termine uniquement avec des configurations composées d'un unique agent, la sortie du protocole étant alors la sortie retournée, via la fonction de sortie, par l'agent.*

Nous définissons maintenant quelques classes de prédicats qui serviront à comparer la puissance des différentes variantes étudiées plus en avant.

Définition 24 (Classe de Prédicats) *Nous définissons les classes de prédicats sur vecteurs d'entier suivantes.*

- *La classe des prédicats* modulo *est l'ensemble des prédicats pouvant s'écrire sous la forme suivante*

$$P(\bar{x}) = \sum_{i \in I} a_i x_i \equiv b \mod c$$

  où I est un ensemble fini d'indices et, pour tout $i \in I$, $a_i \in \mathbb{Z}$ et $b, c \in \mathbb{Z}$.
- *La classe des prédicats* de seuil *est l'ensemble des prédicats pouvant s'écrire sous la forme suivante*

$$P(\bar{x}) = \sum_{i \in I} a_i x_i \geq b$$

  où I est un ensemble fini d'indices et, pour tout $i \in I$, $a_i \in \mathbb{Z}$ et $b \in \mathbb{Z}$.
- *La classe des prédicats* de seuil positif *est l'ensemble des prédicats pouvant s'écrire sous la forme suivante*

$$P(\bar{x}) = \sum_{i \in I} a_i x_i \geq b$$

  où I est un ensemble fini d'indices et, pour tout $i \in I$, $a_i \in \mathbb{Z}$ et $b \in \mathbb{N}$.

— *La classe des prédicats* de seuil nul *est l'ensemble des prédicats pouvant s'écrire sous la forme suivante*

$$P(\bar{x}) = \sum_{i \in I} a_i x_i \geq 0$$

où I est un ensemble fini d'indices et, pour tout $i \in I$, $a_i \in \mathbb{Z}$.

— *La classe des prédicats* de seuil monotone *est l'ensemble des prédicats pouvant s'écrire sous la forme suivante*

$$P(\bar{x}) = \sum_{i \in I} a_i x_i \geq b$$

où I est un ensemble fini d'indices et, pour tout $i \in I$, $a_i \in \mathbb{Z}$ et $b \in \mathbb{Z}$, de plus soit $\forall i \in I$, $a_i \geq 0$, soit $\forall i \in I$, $a_i \leq 0$.

— *La classe des prédicats* de seuil presque nul *est l'ensemble des prédicats pouvant s'écrire sous la forme suivante*

$$P(\bar{x}) = \sum_{i \in I} a_i x_i \geq b$$

où I est un ensemble fini d'indices et, pour tout $i \in I$, $a_i \in \mathbb{Z}$ et $b \in \mathbb{Z}$, de plus il n'existe pas de $i_0 \in I$ tel que $0 > a_(i_0) \geq b$.

Chacune de ces classes de seuil existe en version stricte *en remplaçant le symbole $\geq$ par $>$.*

Chacune de ces classes est un sous-ensemble de la classe des prédicats semi-linéaires.

Un prédicat de seuil nul strict, $16x_1 - 12x_2 - 32x_3 + 20x_4 > 0$ par exemple, peut être réécrit avec un seuil non-nul b tel que b est compris entre 0 et la plus petite combinaison linéaire positive des coefficients. Dans notre exemple, nous avons $2 \cdot 20 + 3 \cdot (-12) = 4$ et nous pouvons alors réécrire l'inégalité sous la forme $16x_1 - 12x_2 - 32x_3 + 20x_4 \geq 3$ puisqu'il n'existe aucune combinaison linéaire positive qui fasse 1, 2 ou 3. En relâchant un peu cette contrainte, c'est-à-dire en n'interdisant plus l'existence de combinaison linéaire positive mais juste l'existence de coefficient entre 0 et le seuil b, nous obtenons les prédicats de seuil presque nul. Ceci justifie le nom de prédicats de seuil presque nul.

Exemples :

— Le prédicat $3x_1 + 2x_2 \equiv x_3 \mod 5$ est un prédicat modulo.

— Le prédicat $3x_1 - 2x_2 + 7x_3 > 3$ est un prédicat seuil positif, et donc également un prédicat de seuil.

	Transitions classiques	Transitions génératrices	Temps réel
Sortie Consensus	Semi-linéaire, (section 5.1)	Semi-linéaire, (section 5.3)	$*$ Modulo, Seuil monotone, Seuil nul, (section 5.2.1)
sortie du leader	Semi-linéaire, (section 5.1)	Semi-linéaire, (section 5.3)	$*$ Modulo, Seuil monotone, Seuil presque nul, Seuil positif, (section 5.2.3)
Dernier agent	—		$*$ Modulo, Seuil monotone, (section 5.2.2)

TABLE 5.1 – Puissance des différentes variantes considérées dans ce chapitre. Les cases marquées $*$ ne sont pas des caractérisations exactes mais des familles de prédicats reconnus par la variante.

— Le prédicat $x_1 - 5x_2 - x_3 \geq 0$ est un prédicat seuil nul, et donc également un prédicat de seuil positif.
— Le prédicat $x_1 + 5x_2 + 2x_3 \geq 13$ et $-3x_1 - 4x_2 - x_3 < 0$ sont des prédicats monotones, contrairement aux prédicats précédents.
— Le prédicat $-7x_1 + 5x_2 - 9x_3 \geq -6$ est un prédicat de seuil presque nul contrairement à $-7x_1 + 5x_2 - 9x_3 \geq -8$.

Les résultats des variantes que nous avons regardées sont résumés dans le tableau 5.1. Dans certains cas, il n'y a pas de caractérisation exacte de démontrée, dans ces cas là nous donnons des classes de prédicats reconnus par ces variantes.

5.1 Les modèles avec transitions classiques

Ces modèles correspondent aux modèles des protocoles de population classique avec les différentes conventions de sorties décrites par Angluin *et al.* dans [AAD+04]. Nous rappelons ici leurs résultats.

Ces deux modèles sont en fait équivalents.

Proposition 5.1.1 ([AAD+04]) *Soit ψ un prédicat. Le prédicat ψ est reconnu par un protocole de population à transition classique avec sortie consensus si et seulement si il est reconnu par un protocole de population à transition classique avec sortie du leader.*

Et nous avons déjà rappelé le résultat suivant au chapitre 2.2.

Proposition 5.1.2 ([AAD+04]) *Soit ψ un prédicat.*

Le prédicat ψ est reconnu par un protocole de population à transition classique avec sortie consensus, ou avec sortie du leader, si et seulement si ψ correspond à un prédicat de l'arithmétique de Presburger — ce qui est équivalent à définir un ensemble semi-linéaire de $\mathbb{N}^\Sigma$.

5.2 Les modèles temps réel

On considère ici les variantes où, à chaque interaction entre une paire d'agents, un agent est détruit et l'agent restant change d'état en fonction de l'état des deux agents avant l'interaction. Ces modèles qualifiés de *temps réel* ne connaissent donc que des exécutions de taille $n-1$ (pour une population de taille n). Étant donné que toutes les exécutions sont de taille finie, la propriété d'équité utilisée jusqu'à présent ne donnent plus aucune contrainte sur quelle exécution est acceptable ou non.

On distinguera cependant différentes variantes selon le mode de terminaison et d'acceptation. Le critère de terminaison peut être la présence d'un unique agent, la stabilité en sortie de la population, ou encore l'absence de règle applicable. Le critère d'acceptation peut être la présence d'un agent qui accepte ou bien la stabilité en sortie de la population sur la réponse "oui".

Ces différents modèles présentent des puissances de calculs différentes.

5.2.1 Sortie consensus

Dans la variante temps réel considérée ici, le calcul termine lorsque tout les agents renvoient la même sortie et qu'aucun ne peut changer de valeur de sortie. C'est la notion de terminaison du modèle des protocoles de population classique.

Les particularités du modèle imposent certaines contraintes sur la relation de transition.

Lemme 5.2.1 *Soit (Q, δ) un protocole temps réel avec sortie consensus.*

Si p et q sont deux états de Q qui peuvent être produits par une entrée et tels que $out(p) \neq out(q)$,

alors il existe un état $r \in Q$ tel que $((p, q), r) \in \delta$

Preuve. Soit ω_p et ω_q deux configurations initiales telles que $\omega_p \to^* p$ and $\omega_q \to^* q$.

Nous considérons le mot initial $\omega_p\omega_q$. Pour toute exécution commençant par cette entrée, la population se stabilise sur une configuration où chaque agent retourne le bon résultat. En particulier, en considérant les exécutions qui mènent respectivement de ω_p à p et de ω_q à q, nous obtenons une configuration menant de $\omega_p\omega_q$ à pq or cette configuration n'est pas stable en sortie, puisque $out(p) \neq out(q)$. Il existe donc une règle qu'il est possible d'appliquer dans cette configuration : il existe r tel que $((p, q), r) \in \delta$.

■

Lemme 5.2.2 *Si P est un prédicat modulo, de seuil monotone ou de seuil nul, alors il existe un protocole de population temps réel avec sortie consensus qui reconnaît P.*

En d'autres termes, les formules atomiques suivantes sont calculables par ce modèle :

- $\sum_{i\in\Sigma} a_i x_i \equiv b \mod c$
- $\sum_{i\in\Sigma} a_i x_i \geq b$ *(ou $> b$), si* $a_i \geq 0$
- $\sum_{i\in\Sigma} a_i x_i \geq 0$ *(ou > 0)*

où, pour tout $i \in \Sigma$,a_i, b et c appartiennent sont des entiers de $\mathbb{Z}$.

Preuve. Nous exhibons, pour chacune des formules atomiques, un protocole de population temps réel avec sortie consensus la calculant.

- $\sum_{i\in\Sigma} a_i x_i \equiv b \mod c$: Nous considérons le protocole défini par $Q = [\![1, c]\!]$, $\delta = \{((p, q), p + q \mod c)\}$, $in(i) = a_i \mod c$ et $out(s) =$ "$s = b$".
- $\sum_{i\in\Sigma} a_i x_i \geq b$ (or $> b$), si $a_i \geq 0$: Nous considérons le protocole défini par $Q = [\![0, b + 1]\!]$, $\delta = \{((p, q), \min(p + q, b + 1))\}$, $in(i) = \min(a_i, b + 1)$ and $out(s) =$ "$s \geq b$" (ou $> b$).
- $\sum_{i\in\Sigma} a_i x_i \geq 0$ (ou > 0) : Nous considérons le protocole défini par $Q = [\![-M, M]\!]$ avec $M = 2.\max(|a_i|)$, $in(i) = a_i$, $out(s) =$ "$s \geq 0$" (ou > 0) et $\delta = \{((p, q), p + q)$ si $(p < 0 < q) \vee (q < 0 < p)\}$,.

■

Pour tout protocole temps réel, il est possible de construire un protocole classique qui calcule de la même manière. Pour cela on ajoute un état supplémentaire qui sera appelé "off" qui sert à simuler l'agent supprimer lors d'une interaction. Une règle du protocole temps réel $a\ b \to c$ est transformée en la règle classique $a\ b \to c$ off. Le protocole classique obtenu simule le protocole temps réel.

Les protocoles temps réel sont donc une restriction des protocoles classiques, à ce titre ils ne peuvent pas reconnaître de prédicat qui ne soit pas semi-linéaire. D'après le résultat précédent, cette variante peut reconnaître un certain nombre de prédicats (modulo, de seuil monotone ou un seuil nul). Cependant c'est une variante strictement plus faible que le modèle classique puisqu'il existe des prédicats de seuil non reconnus. Le résultat suivant montre qu'il existe des prédicats de seuil positif non reconnus.

Lemme 5.2.3 *Soit $(a_i)_{i\in\Sigma}$ une famille d'entiers et b un entier.*
Si il existe des indices i_0 et i_1 tels qu'on ait $a_{i_0} < 0 < a_{i_1} < b$,
alors le prédicat $\sum_{i\in\Sigma} a_i x_i \geq b$ n'est pas calculable.

Preuve. Supposons qu'il existe un protocole P calculant la formule atomique $\varphi(x) = \Sigma a_i x_i \geq b$ telle qu'il existe i_0 et i_1 tels que $a_{i_0} < 0 < a_{i_1} < b$.

Considérons le mot initial $i_1^* i_0^*$.

Par construction, nous avons $out(i_1) = false$ mais il existe $n_1 > 0$ tel que, pour tout $n \geq n_1$, nous avons $out(i_1^n) = true$ et $out(i_1^{n_1-1}) = false$. Cela se traduit mathématiquement par :

$$a_{i_1} \cdot n_1 \geq b > a_{i_1} \cdot (n_1 - 1).$$

Soit w_1 une configuration acceptante stable accessible depuis $i_1^{n_1}$.

Tout état présent dans w_1 doit pouvoir interagir avec i_1 . Nous pouvons appliquer le lemme 5.2.1 puisque leurs sorties sont différentes. Nous pouvons donc construire une configuration acceptante stable w_2 à partir de la configuration $w_1 i_1$. Par ailleurs, la taille de w_2 est inférieure ou égale à la taille de w_1. Nous construisons ainsi de proche en proche les configurations w_m pour $m \geq 1$: nous obtenons w_{m+1} en ajoutant un état i_1 à la configuration w_m et en le faisant interagir avec un des états de w_m. Pour tout $m \geq 1$, la taille de la configurations w_m est inférieure ou égale à la taille de la configuration w_1. Par suite, il existe une configuration apparaissant infiniment souvent dans la suite des $(w_m)_{m\geq 1}$. Alors il existe $k > l$ tels que $w_k = w_l$ et $(k-l)a_{i_1} > |a_{i_0}|$. Nous partons de :

$$a_{i_1} \cdot n_1 \geq b$$

et de

$$a_{i_1} \cdot l \geq 0.$$

Et nous obtenons :

$$a_{i_1} \cdot (n_1 + l) \geq b.$$

Comme $a_{i_0} < 0$, il existe $n_0 \in \mathbb{N}$ tel que

$$b > a_{i_1} \cdot (n_1 + l) + a_{i_0} \cdot n_0 \geq b + a_{i_0}.$$

En ajoutant $a_{i_1} \cdot (n_1 + l) + a_{i_0} \cdot n_0 \geq b + a_{i_0}$ et l'inégalité $(k - l)a_{i_1} > -a_{i_0}$, nous obtenons :

$$a_{i_1} \cdot (n_1 + k) + a_{i_0} \cdot n_0 > b.$$

Finalement, nous avons :

$$a_{i_1} \cdot (n_1 + k) + a_{i_0} \cdot n_0 > b > a_{i_1} \cdot (n_1 + l) + a_{i_0} \cdot n_0.$$

Nous avons donc $\varphi(i_1^{n_1+k} i_0^{n_0}) \neq \varphi(i_1^{n_1+l} i_0^{n_0})$. Cependant il existe deux exécutions du protocole P telles que :

- la première commence sur la configuration $i_1^{n_1+k} i_0^{n_0}$, atteint la configuration $w_k i_1^{n_1} i_0^{n_0}$ puis se stabilise sur une configuration $\bar{w}$;
- la seconde commence sur la configuration $i_1^{n_1+l} i_0^{n_0}$, atteint la configuration $w_k i_1^{n_1} i_0^{n_0}$ — puisque $w_k = w_l$ — puis se stabilise sur la configuration $\bar{w}$.

Cela contredit le fait que le protocole P calcule φ.

Pour conclure, il n'existe pas de tel protocole.

■

Ces deux résultats ne donnent pas de caractérisation précise de la puissance de calcul de cette variante, mais ils donnent cependant un encadrement qui suffit pour comparer cette variante avec les autres variantes des protocoles de population.

5.2.2 Sortie du dernier agent

Dans la variante considérée maintenant, les règles de transitions sont toujours de la même forme — de la forme $a\,b \to c$. Mais le calcul doit se terminer avec un unique agent, qui retourne la sortie du protocole, et donc le résultat du calcul. Il n'y a cependant aucune contrainte sur quelle paire interagit quand ou quelle règle est appliquée.

Comme la population finale doit être réduite à un seul agent, tout paire doit pouvoir interagir, c'est le sens du lemme suivant.

Lemme 5.2.4 *Considérons (Q, δ) un protocole de population temps réel avec sortie dernier agent.*

Si p et q sont deux états de Q, alors il existe un état $r \in Q$ tel que $((p, q), r) \in \delta$.

Preuve. S'il existe une paire d'agents p, q tels qu'il n'existe pas d'état r tel que $((p,q),r) \in \delta$, alors la configuration pq est une configuration stable avec plus d'un agent.

Cela contredit la définition du modèle. Ceci achève la démonstration du lemme.

■

Les formules calculées par ce modèles sont stables par les opérations booléennes de base.

Lemme 5.2.5 *Si φ et ψ sont calculables par protocole de population temps réel avec sortie dernier agent, alors $\varphi \vee \psi$, $\varphi \wedge \psi$ et $\neg\varphi$ sont calculables par ce modèle.*

Preuve. En exécutant en parallèle des protocoles calculant φ et ψ, la sortie renvoyée par les agents est, respectivement, le "ou" logique, le "et" logique ou le "non" logique.

Une exécution de ce protocole correspond à des exécutions possibles pour les autres protocoles, ils doivent donc calculer leur bon résultat et ainsi le protocole qui les compose renvoie le bon résultat.

■

Parmi les différentes variantes temps réel que nous considérons, c'est la seule dont l'ensemble des prédicats reconnus est stable par opération booléenne. Cela vient du fait que dans les autres variantes les agents ne peuvent pas interagir dans un ordre complètement arbitraire, certaines paires ne pouvant pas interagir à certains moments.

Lemme 5.2.6 *Si P est un prédicat modulo ou de seuil monotone, alors il existe un protocole de population temps réel avec sortie dernier agent qui reconnaît P.*

Cela signifie que, pour chacune des formules suivantes, il existe un protocole de population temps réel avec sortie dernier agent la calculant :

— $\Sigma_{i\in\Sigma} a_i x_i \equiv b \mod c$

— $\Sigma_{i\in\Sigma} a_i x_i \geq b$ *avec* $\forall i \in \Sigma, a_i \geq 0$ *ou* $\forall i \in \Sigma, a_i \geq 0$

où, pour tout $i \in \Sigma$,a_i, b et c appartiennent à $\mathbb{Z}$.

Preuve. Nous exhibons, pour chacune des formules atomiques, un protocole de population temps réel avec sortie dernier agent la calculant.

— Pour calculer $\Sigma_{i\in\Sigma} a_i x_i \equiv b \mod c$, nous utilisons $Q = [0 \dots c-1]$ où $i \in \Sigma$ est associé à l'état $a_i \mod c$. Nous posons $\delta = \{(p, q, p+q \mod c \mid p, q \in Q\}$ et $out(q) = true$ si et seulement si $q = b$.

— Nous supposons que pour tout i, $a_i \geq 0$ et $b \geq 0$. Le cas où ils sont négatifs se fait de manière symétrique. Pour calculer $\Sigma_{i\in\Sigma} a_i x_i \geq b$, nous utilisons $Q = [\![0..b+1]\!]$. L'état initial $i \in \Sigma$ est associé à l'état $\min(a_i, b+1)$. Nous posons $\delta = \{(p, q, \min(p+q, b+1) | p, q \in Q\}$ et $out(q) = true$ si et seulement si $q \leq b$. Comme tout les coefficients a_i sont positifs, une fois qu'un agent atteint l'état b (ou plus), sa somme partielle ne pourra plus diminuer et il est déjà possible de conclure sur le résultat du calcul.

■

Lemme 5.2.7 *La formule $\Sigma_{i\in\Sigma} a_i x_i \geq b$ telle qu'il existe i_1 et i_0 tels que $a_{i_1} > 0$ et $a_{i_0} < 0$ n'est pas calculable par cette variante.*

Preuve. Soit $\varphi(x) = \Sigma_{i\in\Sigma} a_i x_i \geq b$ une formule telle qu'il existe i_1 et i_0 tels que $a_{i_1} > 0$ et $a_{i_0} < 0$

Soit $P = (Q, \delta)$ un protocole temps réel avec sortie du dernier agent calculant φ.

On considère la suite d'état $(q_j)_{j\geq 1}$ définie par $q_1 = i_1$ et, pour tout $i > 1$, q_i vérifie $((q_{i-1}, i_1), q_i)$. Il existe un état $\bar{q}$ qui apparaît infiniment souvent dans la suite $(q_j)_{j\geq 1}$. Soit $j_1 > j_2$ tels que $q_{j_1} = q_{j_2} = \bar{q}$ et $a_{i_1} \cdot (j_1 - j_2) + a_{i_0} > 0$.

Il existe $k \in \mathbb{N}$ tels que

$$a_{i_1} \cdot j_2 + a_{i_0} \cdot (k+1) \geq b > a_{i_1} \cdot j_2 + a_{i_0} k.$$

Nous avons alors

$$a_{i_1} \cdot j_1 + a_{i_0} k > a_{i_1} \cdot (j_1 - j_1 + j_2) + a_{i_0}(k+1) = a_{i_1} \cdot j_2 + a_{i_0} \cdot (k+1) \geq b.$$

Et donc :

$$a_{i_1} \cdot j_1 + a_{i_0} k > b > a_{i_1} \cdot j_2 + a_{i_0} k.$$

Nous en déduisons donc que $\varphi(i_1^{j_1} i_0^k) \neq \varphi(i_1^{j_2} i_0^k)$. Cependant, pour chacun de ces mots initiaux il existe une exécution qui mène à la même configuration $\bar{q} i_0^k$. Par suite, le protocole P retourne nécessairement la même résultat sur les entrées $i_1^{j_1} i_0^k$ et $i_1^{j_2} i_0^k$, ce qui contredit le fait que P calcule φ.

Par suite, il n'existe pas de tel protocole P.

■

5.2.3 Sortie du leader

La variante que nous considérons à présent est toujours une variante temps réel — c'est-à-dire que $\delta \in Q^2 \times Q$ — mais la sortie est de type leader : la condition de terminaison requiert la stabilité de la population, c'est-à-dire que plus aucun agent ne puisse changer d'état et une configuration stable retourne *true* si et seulement si il y a au moins un agent dont l'état renvoie *true* en sortie. Le résultat d'un calcul sur une entrée ω est b si pour toute exécution commençant sur une configuration correspondant à $in(\omega)$ la population se stabilise sur une configuration renvoyant b.

Ce modèle est plus faible au sens large que le modèle classique. Le modèle classique peut calculer tout ce que ce modèle calcule.

Lemme 5.2.8 *Les langages reconnus par un protocole de population temps réel avec une sortie du leader sont reconnus par un protocole de population classique.*

Preuve. Dans [AAD+04], Angluin *et al.* montrent que les protocoles de population classiques ont exactement la même puissance de calcul si on change la condition d'acceptation de la manière suivante : une configuration est acceptante si et seulement si au moins un agent renvoie *true*.

Soit $P = (Q, \delta)$ un protocole de population temps réel avec une sortie du leader. Nous construisons $P' = (Q', \delta')$ un protocole de population classique avec la condition d'acceptation ci-dessus. Nous ajoutons un état : $Q' = Q \cup \{off\}$ avec $out(off) = false$. Cet état est un état éteint : un agent dans l'état *off* représentera un agent qui a été "détruit" pendant le calcul temps-réel. La relation de transition est adaptée comme suit :

$$\delta' = \{p\ q \to r\ off \mid ((p, q), r) \in \delta\}$$

Et le protocole P' reconnaît le même langage que P.

■

Ce modèle est plus fort au sens large que les protocoles temps réel avec sortie du leader : tout ce qui est calculable avec ceux-ci est également calculable avec un protocole temps réel avec une sortie dernier agent.

Lemme 5.2.9 *Les langages reconnus par un protocole temps réel avec une sortie dernier agent sont reconnus un protocole temps réel avec une sortie du leader.*

Preuve. Le modèle précédent correspond exactement aux protocoles de ce modèle qui terminent avec une population de taille 1 sur toutes leurs exécutions.

■

Les lemmes suivants donnent un ensemble de formules atomiques calculables par ce modèle et une formule atomique non-calculable. Certaines des formules calculables par ce modèle ne le sont pas par le précédent et la formule atomique non-calculable par ce modèle exhibée ci-après est calculable par le modèle classique.

Lemme 5.2.10 *Si P est un prédicat modulo, de seuil presque nul, de seuil positif ou de seuil monotone, alors il existe un protocole de population temps réel avec sortie du leader qui reconnaît P.*

Cela signifie que, pour chacune des formules suivantes, il existe un protocole temps réel avec sortie du leader la calculant.

- $\sum_{i\in\Sigma} a_i x_i \equiv b \mod c$
- $\sum_{i\in\Sigma} a_i x_i \geq b$ *(or $> b$) où au moins une des conditions suivantes est vérifiée :*
  1. $(\forall i, a_i \geq 0) \vee (\forall i, a_i \leq 0)$
  2. $b \geq 0$
  3. $\nexists i_0, b \leq a_{i_0} < 0$

où, pour tout i, $a_i, b, c \in \mathbb{Z}$.

Preuve. Le modèle précédent pouvait calculer les formules atomiques basée sur le modulo, le lemme 5.2.9 suffit donc à conclure.

Nous nous intéressons maintenant aux formules atomiques de type seuil. Nos preuves sont écrites pour des inégalités larges (c'est à dire "$\cdots \geq b$"), les preuves pour les inégalités strictes étant très similaires.

1. Ce cas était calculable par le modèle précédent, il l'est donc aussi par ce modèle étant donné le lemme 5.2.9.
2. Nous donnons ici un protocole calculant $\Sigma_{i\in\Sigma} a_i x_i \geq b$ avec $b \geq 0$. Nous utilisons $Q = [\![-M..M]\!]$ où $M = max(|a_i|, 2b)$.

   À la lettre $i \in \Sigma$ nous associons l'état $a_i \in Q$.

   Nous posons $\delta = \{(p, q, p+q) | p, q \in Q \wedge |p+q| \leq M \wedge out(p) \neq out(q)\}$ et $out(q) = true$ si et seulement si $q \leq b$.

   La relation de transition δ préserve la somme des états des agents. La population est stable soit lorsqu'il ne reste qu'un unique agent —qui retourne

le résultat correct— soit lorsqu'il reste plusieurs agents et que le somme des états de toute paire d'agents est, en valeur absolue, plus grande que M.

Cela signifie que s'il reste plusieurs agents, pour toute paire d'agent dans des états p et q, nous sommes dans l'un des deux cas suivants :

— nous avons $q \geq \frac{|M|}{2} \geq b$ ou $p \geq \frac{|M|}{2} \geq b$. Par suite l'exécution accepte.

— nous avons $p+q < -M \leq -2|b|$, $p \geq -M$ et $q \geq -M$; et alors nous avons $p < 0 \leq b$ et $q < 0 \leq b$. Par suite l'exécution refuse.

Finalement, le protocole accepte si la somme des états des agents est supérieure à b et refuse sinon.

3. Nous donnons maintenant un protocole calculant la formule $\Sigma_{i\in\Sigma} a_i x_i \geq b$ telle que $\nexists i_0, b \leq a_{i_0} < 0$. Nous supposerons l'existence de coefficients a_i positifs et d'autres négatifs, sinon nous sommes dans le premier cas.

   Nous définissons le protocole par l'ensemble d'états $Q = [\![2(b-M), b[\![\cup [\![0, M]\!]$, avec $M = max(|a_i|, |b|) = max(|a_i|)$, la fonction de sortie $out(i) =$ "$i \geq b$", et la fonction de relation définie par $((p, q), r) \in \delta$ pour $r = p + q \in Q$.

   Les états acceptants sont tous positifs et les états rejetant sont tous strictement négatifs.

   Si tout les agents de la population renvoient la même sortie, alors c'est le résultat du calcul et il est facile de voir que c'est le résultat correct.

   Soit C une configuration stable avec au moins deux agents dans des états p et q tels que $q < b < 0 \leq p$.

   Nous avons $2(b-M) \leq q < 0$ et $0 \leq p \leq M$ or la configuration est stable donc $p + q \notin Q$. Comme $2(b - M) \leq p + q < M$, nous avons donc nécessairement $b \leq p + q < 0$. Et puisque $p \leq M$, nous avons $b - M \leq q < b$.

   Soit $q' \in Q$ tel que $q + q' < 2(b - M)$.

   Nous avons alors $q' < b - M$ et $2(b-M) < q' + p < b$. Par suite, s'il existe un autre état $q' < b$, soit $p + q' \in Q$ soit $q + q' \in Q$ et alors la configuration n'est pas stable. Pour conclure, la configuration en peut pas contenir deux tels états p et q et d'autres états de valeur négative. Nous en déduisons que la somme des états des agents (qui est constante durant une exécution) est plus grande que b et que donc le mot d'entrée est dans le langage. Par ailleurs, l'exécution accepte puisqu'un agent accepte.

■

Lemme 5.2.11 *Soit $a_i, b, c \in \mathbb{Z}$.*

Si il existe i_0, i_1 tel que $b < a_{i_0} < 0 < a_{i_1}$, alors $\Sigma a_i x_i \geq b$ n'est pas calculable par un protocole temps réel avec sortie du leader.

Preuve. Comme $b < a_{i_0} < 0$ et qu'un mot x est accepté si et seulement si la quantité $\Sigma a_i x_i$ est plus grande que b, il existe $n_0 \geq 2$ tel que $w_0 = i_0^{n_0}$ doit être rejeté et $w_1 = i_0^{n_0+1}$ doit être accepté. On notera c_0 et c_1 des configurations stables telles que $w_0 \rightarrow^* c_0$ et $w_1 \rightarrow^* c_0.i_0 \rightarrow c_1$.

La configuration c_0 possède au moins un état acceptant et c_1 aucun. Par suite, il existe exactement un état acceptant $s+$ dans c_0 et $((s+, i_0), s_0) \in \delta$ où s_0 est un état rejetant. Une exécution commençant avec la configuration $s + .i_0^n$ doit se stabiliser sur une configuration rejetante pour tout $n \geq 1$.

Comme i_0 et s_0 acceptent, il existe nécessairement s_1 tel que $((s0, i_0), s_1) \in \delta$ et s_1 rejette. Nous définissons la suite $s_0, s_1, s_2, \ldots$ d'état de Q. Il existe k tel que $s_n = s_{n+k}$ pour n assez grand. Nous avons donc deux configurations $i_0^{n_1}$ et $i_0^{n_2}$ à partir desquelles il est possible d'accéder à la même configuration ω et $n_2 - n_1$ peut être plus grand que a_{i_1}.

Par suite, il existe des exécutions démarrant respectivement sur les mots $i_0^{n_1} i_1^p$ et $i_0^{n_2} i_1^q$ et se stabilisant sur une même configuration pour tout p et q ; mais il existe p et q tels que $i_0^{n_1} i_1^p$ accepte et $i_0^{n_1} i_1^q$ rejette. Le protocole ne sera donc pas capable de décider correctement dans tout les cas.

Finalement, il n'existe pas de protocole calculant de telles formules.

■

5.3 Les modèles avec transitions génératrices

Dans cette variante, les interactions ont toujours lieu par paire mais il y a à la sortie de l'interaction un nombre d'agents qui n'est pas forcément 2. Par exemple, si deux agents dans un état p interagissent, on peut obtenir 3 agents dans l'état q, ce qui nous donnerai une règle $p\ p \rightarrow q\ q\ q$. Il y a cependant toujours un nombre fini de règles. Nous allons voir que la relaxation de la contrainte sur le nombre d'agents après interaction ne change pas la puissance du modèle considéré.

Théoreme 5.3.1

1. *Les langages reconnus par des protocoles de population avec sortie consensus ayant une relation de transition génératrice sont exactement les ensembles semi-linéaires.*
2. *Les langages reconnus par des protocoles de population avec sortie du leader ayant une relation de transition génératrice sont exactement les ensembles semi-linéaires.*

Tout d'abord il faut remarquer que les protocoles de population avec une relation de transition classique sont des protocoles avec une relation de transition génératrice particuliers. Il s'ensuit donc naturellement que l'ensemble des langages reconnus par des protocoles de population ayant une relation de transition génératrice contient les langages semi-linéaires.

Nous allons maintenant prouver qu'un langage reconnu par un protocole avec transitions génératrices est semi-linéaire. Pour cela, nous allons réutiliser la preuve exacte de ce résultat pour les protocoles de population classiques. Nous ne retranscrirons pas l'intégralité de la preuve et tout les détails, nous nous vérifions juste que les principaux lemmes ne changent pas pour cette variantes.

Nous rappelons la définition des fonctions de troncatures définies à la section 2.2.2. Les fonctions de troncatures $\tau_k : \mathcal{C} \to \mathcal{C}$ définies pour $k \geq 1$ sont telles que :

$$\forall c \in \mathcal{C},\ \forall q \in Q,\ \tau_k(c)(q) = \min(k, c(q)).$$

L'ensemble des configurations $\mathcal{C}$ est partitionné en, d'un coté, l'ensemble des configurations stables $\mathcal{S}$ et, de l'autre coté, les configurations instables $\mathcal{U}$.

L'ensemble $\mathcal{U}$ est toujours clos vers le haut par inclusion, c'est-à-dire que :

$$\forall c \leq d,\ c \in \mathcal{U} \Rightarrow d \in \mathcal{U}.$$

Les ensembles de configurations sont les mêmes pour le modèle classique et pour le modèle avec transitions génératrices. Le lemme suivant est donc toujours vérifié.

Lemme 5.3.1

— *Il existe* $k \geq 1$ *tel que* $c \in \mathcal{U}$ *si et seulement si* $\tau_k(c) \in \mathcal{U}$.
— *Il existe* $k \geq 1$ *tel que* $c \in \mathcal{S}$ *si et seulement si* $\tau_k(c) \in \mathcal{S}$.

Nous redéfinissons les extensions d'une configuration c comme le sous-ensemble de $\mathbb{N}^\Sigma$ suivant :

$$X(c) = \left\{ x \in \mathbb{N}^\Sigma \;\middle|\; \exists d \geq c, c + x \to^* d \wedge \tau_k(c) = \tau_k(d) \right\}.$$

Les extensions ont toujours la bonne propriété suivante. Si x est une entrée telle que $x \to^* c$ où $c \in \mathcal{S}$ et y une entrée appartenant à $x + X(c)$, alors nous avons l'équivalence entre :

(1) x est accepté par P (2) y est accepté par P.

Et de plus, l'extension d'une configuration est toujours un monoïde. Le changement de fonction de transition ne change rien puisqu'à aucun moment l'hypothèse selon laquelle toutes les configurations d'une exécution ont la même taille intervient.

Par suite, un ensemble $Y \subset \mathbb{N}^\Sigma$ reconnu par un protocole peut toujours être recouvert par un nombre finis d'ensembles de la forme $x_i + M_i$, où x_i est une entrée et M_i une extension d'une configuration stable accessible depuis x_i.

Cela signifie donc qu'un langage calculé par un protocole de population avec transitions génératrices correspond à un ensemble semi-linéaire, tout comme la version avec transition classique correspondante.

5.4 Résumé et conclusion

Dans ce chapitre, nous avons étudié différentes variantes du modèle classique où la relation et la notion d'acceptation pouvaient prendre différentes formes. Angluin *et al.* ont montré que lire la sortie en consensus ou en leader ne change pas la puissance du modèle avec transitions classiques. Nous avons en plus montré qu'autoriser des transitions génératrices ne changeait pas non plus la puissance de calcul du modèle.

Les variantes temps réel ont une puissance de calcul plus faible. La manière de lire la sortie influence beaucoup la puissance du modèle résultant. Les classes des langages reconnus par ces modèles ne sont en général pas stable par opérations booléennes. Ils peuvent calculer les prédicats modulo et certains prédicats de seuil en fonction le modèle.

Notre étude pourrait être complétée en considérant les variantes probabilistes — où la propriété d'équité est supprimée et la prochaine paire d'agents à interagir est choisie aléatoirement. Dans le cas classique, cette variante peut augmenter la puissance de calcul comme nous l'avons vu au chapitre 2.4.3. L'utilisation de règles génératrices devrait permettre d'augmenter la mémoire utilisable par la machine de Turing simulée, et donc sa puissance de calcul.

Chapitre 6

Quand la Confiance règne

Nous considérons dans ce chapitre deux variantes des protocoles de population où les agents sont *confiants*. L'idée de cette restriction est d'ajouter une notion de confiance entre les agents : si deux agents ayant des opinions similaires se rencontrent, ils ne changent pas d'opinion. Il est possible de caractériser les langages reconnus par de tels modèles.

Plus précisément, pour représenter la notion d'opinion dans des protocoles de population, nous partitionnons l'ensemble des états en des sous-ensembles correspondants aux opinions possibles. Potentiellement deux agents qui sont dans des états différents peuvent avoir la même opinion. Dans notre étude, ici, nous nous restreignons aux cas où l'opinion d'un agents est sa sortie. Nous n'étudierons que les *protocoles confiants en la sortie*. Nous distinguons les protocoles de population confiants *faibles* et *forts*. Dans la variante forte, un agent a toujours une opinion et deux agents ayant la même opinion ne peuvent pas changer d'état en interagissant ensemble. Dans la variante faible, un agent peut être sans opinion et deux agents ayant la même opinion peuvent interagir et changer d'état, à condition qu'ils conservent tout de même la même opinion.

Nous verrons que ces variantes des protocoles calculent des prédicats sur les fréquences des occurrences de chaque symbole plutôt que sur les nombres des occurrences de manière générale.

Les résultats présentés dans ce chapitre ont été publiés dans [BLR13].

6.1 Le modèle

Nous posons ici la définition formelle du modèle des protocoles de population confiants. De même que pour les protocoles de population classiques, un calcul se

déroule sur un ensemble fini d'agents. Chaque agent commence avec une entrée $\sigma \in \Sigma$ – où Σ est un alphabet fini d'entrée — et a un état $q \in Q$ — où Q est un ensemble fini d'états. La population évolue par interaction de paires d'agents selon les règles décrites par la relation de transition δ.

Définition 25 (Protocole de Population Confiant) *Un* protocole de population confiant *est défini par un septuplet* $(Q, I, \Sigma, in, Y, out, \delta)$ *avec :*

- Q, *un ensemble fini d'états ;*
- I, *l'ensemble des opinions qui donne une partition de* Q *en* $|I|$ *sous-ensemble distincts :* $Q = Q_1 \cup Q_2 \cup \cdots \cup Q_{|I|}$ *et pour tout* $i \neq j$ *dans* I, $Q_i \cap Q_j = \emptyset$ *;*
- Σ, *un alphabet fini d'entrée ;*
- Y, *un alphabet fini de sortie ;*
- in, *une fonction d'entrée de* Σ *dans* Q *;*
- out, *une fonction de sortie de* Q *dans* Y, *la fonction out est constante sur chacun des ensembles* Q_i *où* $i \in I$ *:* $\forall i \in I,\ \exists y \in Y,\ \forall q \in Q_i,\ out(q) = y$ *;*
- $\delta \subset Q^2 \times Q^2$, *une relation transition.*

Remarque : Il est important de noter que, contrairement à [AAER07, AAD+04], à la plus part des travaux suivants et des chapitres précédents, nous nous restreignons pas au cas où $Y = \{true, false\}$. Cela permet de parler du problème de la moyenne par exemple.

La principale différence entre cette variante confiante et les protocoles de populations classiques réside en la restriction sur les interactions. Il y a plusieurs façons de restreindre les interactions d'agents ayants la même opinion. nous définissons ci-après les variantes *fortes* et *faibles* des protocoles de population confiants.

Définition 26 (Protocole de Population Confiant Fort) *Un* protocole de population confiant fort *est un protocole de population confiant tel que* δ *ne modifie pas l'état d'agents ayant la même opinion :*

$$\forall i \in I,\ \forall p, q \in Q_i,\ \forall r, s, \in Q, \quad si\ ((p, q), (r, s)) \in \delta,\ alors\ (r, s) = (p, q).$$

Définition 27 (Protocole de Population Confiant Faible) *Un* protocole de population confiant faible *est un protocole de population confiant dont l'ensemble d'opinion contient une opinion spéciale ? (l'opinion "sans opinion") et tel que* δ *vérifie la condition suivante :*

$$\forall i \in I \setminus \{?\},\ \forall p, q \in Q_i,\ \forall r, s, \in Q, \quad si\ ((p, q), (r, s)) \in \delta,\ alors\ r, s \in Q_i.$$

Tout langage reconnu par un protocole confiant fort est reconnu par un protocole confiant faible puisque un protocole confiant fort peut être vu comme un protocole confiant faible où l'opinion spéciale n'est pas utilisée et où la relation de transition a une contrainte supplémentaire. Il serait également possible de considérer des variantes intermédiaires — un protocole confiant fort avec une opinion "pas d'opinion" par exemple — et de telles variantes reconnaîtrait un ensemble de langages compris au sens de l'inclusion entre les langages reconnus par les protocoles confiants forts et faibles. Nous allons voir que les langages reconnus par les protocoles confiants forts et ceux reconnus par les protocoles confiants faibles sont en fait les mêmes. Ce résultat rend l'étude de variantes intermédiaires superflue.

Nous gardons le vocabulaire et la terminologie que nous avons définis et utilisés pour le cas classique, et les variantes des chapitres précédents. Nous gardons ainsi les notions de configurations, d'exécutions, d'équité et de langages calculés ou reconnus.

En particulier, nous conservons la notion d'acceptation — un mot est accepté si toute exécution équitable mène à des configurations stables en sortie — cependant nous nous intéresserons au cas *multivalué*, où Y n'est pas réduit à $\{true, false\}$. Un protocole de population confiant calcule alors une fonction f de $\mathbb{N}^\Sigma$ dans Y. L'espace des entrées sera alors partitionné selon la valeur de la sortie. Un sous-ensemble $E \subset \mathbb{N}^\Sigma$ sera dit *séparable* ou *calculable* par ce modèle si il existe un Protocole de Population confiant calculant une fonction f et une valeur de sortie $y \in Y$ telles que $E = f^{-1}(y)$.

6.2 Quelques protocoles confiants

Ci-après nous présentons deux exemples de Protocoles confiants.

6.2.1 Le problème du seuil nul

Définition 28 (Problème du Seuil Nul) *Le problème du* Seuil Nul *est le problème suivant :*

Soit a et b deux entiers,une fonction $g : \Sigma \to [\![a, b]\!]$, un entier c et $Y = \{true, false\}$. Nous voulons calculer la fonction f telle que :

$$\forall x \in \mathbb{N}^\Sigma, \quad f(x) = true \text{ si et seulement si } \sum_{s\in\Sigma} g(s)x(s) \geq 0.$$

Proposition 6.2.1 *Le problème du seuil nul peut être calculé par des protocoles de population confiants fort et faible.*

Preuve. Considérons le protocole $Seuil_0 = (Q, \{+, -\}, \Sigma, Y, in, out, \delta)$ défini par :
- $Q = Q_- \bigcup Q_+$, avec $Q_- = [\![a, 0[\![\cup \{0^-\}$ et $Q_+ = [\![0, b]\!]$
- $in = g$, $Y = \{true, false\}$, et $out(i) = true$ si et seulement si $i \in [\![0, b]\!]$.
- δ est définie par l'ensemble de règles suivant :

$$\begin{array}{rcll} i & j \to 0 & (i+j) & \text{si } i \cdot j < 0 \text{ and } i + j \geq 0 \\ i & j \to 0^- & (i+j) & \text{si } i \cdot j < 0 \text{ and } i + j < 0 \\ 0^- & i \to 0 & i & i \geq 0 \\ 0 & i \to 0^- & i & i < 0 \end{array}$$

D'abord, il faut remarquer que le protocole $Seuil_0$ est un protocole confiant à la fois fort et faible. Ensuite, nous pouvons remarquer que la somme des valeurs stockées sur les agents — qui vaut $\sum_{s \in \Sigma} g(s)x(s)$ si x a été passé en entrée — est constante au cours d'une exécution.

La preuve de correction se base sur la démonstration des points suivants :

1. Lors d'une exécution équitable $(C_t)_{t \geq 0}$, il existe un instant t_0 tel que pour tout $t \geq t_0$ tous les états non-nuls présents dans la configuration C_t soient de même signes (toujours tous positifs ou tous négatifs).
2. Il existe $t_1 > t_0$ tel que pour tout $t \geq t_1$ la configuration C_t est une configuration stable et tout les agents de C_t sont dans un état renvoyant le résultat correct.

Le premier point se démontre en remarquant que s'il existe deux agents dans des états de signes différents i et j et non nuls alors ils peuvent interagir, et ils deviennent de même signe et l'un d'eux s'annule. Le nombre d'agents dans un état non-nul a diminué, il n'est donc pas possible d'itérer ce pas infiniment souvent. En itérant ce pas de calcul, il est possible d'atteindre une configuration où tous les agents dans un état non-nul ont le même signe. À toute instant une telle configuration est accessible (en plusieurs pas de calcul), la propriété d'équité nous assure donc l'existence d'un t_0 telle que dans C_{t_0} tous les agents dans un état non-nul ont le même signe. Par ailleurs, une fois que tous les agents non-nuls sont tous de même signe, il n'est pas possible de régénérer des agents dans un état de l'autre signe.

Le second point se démontre en remarquant que les deux dernières règles permettent de mettre les états nuls dans l'état nul renvoyant la sortie correcte. La propriété d'équité permet d'assurer l'existence d'un instant t_1 à partir duquel une configuration où tous les états non-nuls sont de même signe et tous les états nuls renvoie la même sortie. Une telle configuration est stable, puisqu'aucune règle ne peut plus s'appliquer.

■

6.2.2 Le problème de la moyenne

Définition 29 (Le Problème de la Moyenne) *Le* Problème de la Moyenne *est le problème suivant (voir par exemple [HOT11]) :*

Soit $a, b \in \mathbb{N}$. *Soit une fonction* $g : \Sigma \to [\![a, b]\!]$. *Nous considérons l'alphabet de sortie* $Y = \{\{i\} \mid i \in [\![a, b]\!]\} \cup \bigcup_{i \in [\![a,b[\![} \{]i, i+1[\}$. *Nous voulons calculer la fonction* f *définie par :*

$$f(x) = \begin{cases} m & \text{si } m \in \mathbb{N} \\ ]\lfloor m \rfloor, \lfloor m \rfloor + 1[& \text{sinon.} \end{cases}, \text{ où } m = \frac{\sum_{\sigma \in \Sigma} g(\sigma) \cdot x(\sigma)}{\sum_{\sigma \in \Sigma} x(\sigma)}.$$

Proposition 6.2.2 *Le problème de la moyenne peut être calculé par des protocoles de population confiants forts et faibles.*

Preuve. Considérons le protocole $Moyenne = (Q, I, \Sigma, Y, in, out, \delta)$ défini par :

- $Q = \bigcup_{i \in [\![a,b]\!]} \{i, i^+, i^-\}$,
- $I = \{i | i \in [\![a, b]\!]\} \cup \bigcup_{i \in [\![a,b[\![} \{]i, i+1[\}$, la partition est composée des sous-ensembles $Q_i = \{i\}$, pour tout $i \in [\![a, b]\!]$, et $Q_{]i,i+1[} = \{i^+, (i+1)^-\}$, pour tout $i \in [\![a, b[\![$.
- $in = g$, $Y = [\![a, b]\!] \bigcup_{i \in [\![a,b[\![} \{]i, i+1[\}$, et $out(i^+) = out((i+1)^-) =]i, i+1[$, $out(i) = \{i\}$.
- δ est définie par la donnée des règles suivantes :

$$\begin{array}{cccc} i* & j* \to & \frac{i+j}{2} & \frac{i+j}{2} \\ i & i* \to & i & i \end{array} \quad \text{si } i + j \equiv 0 \mod 2 \text{ et } i \neq j$$

$$i* \quad j* \to k^+ \quad (k+1)^- \quad \text{si } i + j \equiv 1 \mod 2,\ i < j, \text{ où } k = \tfrac{i+j-1}{2}$$

avec $i* \in \{i, i^+, i^-\}$ et $j* \in \{j, j^+, j^-\}$.

D'abord, nous pouvons remarquer que ce protocole confiant est à la fois fort et faible.

Ensuite, nous posons la fonction $e : Q \to \mathbb{N}$ par $e(i^+) = e(i^-) = e(i) = i$. Par construction, la somme sur la population des valeurs images des états par la fonction e est constante et vaut $s = \sum_{\sigma \in \Sigma} g(\sigma) \cdot x(\sigma)$où x est l'entrée.

La preuve de correction passe par la preuve des deux points suivants :

1. Pour toute exécution équitable $(C_n)_{n \in \mathbb{N}}$, il existe t_0 tel que pour $t \geq t_0$ les agents ont le même entier m dans leur état. La moyenne vaut alors $\frac{s}{n} = \frac{\sum_{\sigma \in \Sigma} g(\sigma) \cdot x(\sigma)}{\sum_{\sigma \in \Sigma} x(\sigma)}$ et nous avons $|m - \frac{s}{n}| < 1$

2. Il existe $t_1 > t_0$ tel que pour tout $t \geq t_1$ C_t est une configuration stable avec la bonne interprétation.

Le premier point se montre en remarquant que tant que deux états utilisant des entiers différents alors ils peuvent interagir. En faisant continuellement interagir l'état utilisant le plus grand entier et l'état utilisant le plus petit entier, il est possible d'atteindre une configuration où tous les états ont la même valeur par la fonction e. Et par équité, une telle configuration finira par être atteinte.

Pour le second point, il suffit de voir qu'une fois atteinte une configuration où tout les états utilisent le même entier est atteinte, alors l'utilisation de la seconde ou troisième règle amènera tous les agents à se mettre d'accord sur la sortie. L'équité nous assure encore une fois que cette configuration, stable, finira par être atteinte.

■

6.3 Calculabilité

6.3.1 Partition de l'espace

Nous rappelons qu'un sous ensemble X de $\mathbb{N}^\Sigma$ peut être *séparé* par un protocole de population ayant comme alphabet d'entrée Σ et comme alphabet de sortie Y si il existe une fonction $f : \mathbb{N}^\Sigma \to Y$ calculable par ce protocole de population et $y \in Y$ tel que $X = f^{-1}(y)$.

Proposition 6.3.1

- *Les ensembles séparés par chacune des deux versions des Protocoles de Population confiants sont stables par union et intersection :*
  si A et B sont des sous-ensembles de $\mathbb{N}^\Sigma$ qui peuvent être séparés par un protocole de population confiant, alors $A \cup B$ et $A \cap B$ peuvent être séparés chacun par un protocole de population confiant.
- *Les ensembles séparés par chacune des deux versions des Protocoles de Population confiants sont stables par complémentaire :*
  si A est un sous-ensembles de $\mathbb{N}^\Sigma$ qui peut être séparé par un protocole de population confiant, alors $\mathbb{N}^\Sigma \setminus A$ peut être séparé par un protocole de population confiant.
- *Soit $(A_j)_{j\in J}$ une partition de $\mathbb{N}^\Sigma$, où J est un ensemble fini d'indices, telle que, pour tout $j \in J$, A_j peut être séparé par un protocole de population confiant. Il existe un même protocole de population confiant séparant chacun des A_j.*

Preuve.

- La preuve de la stabilité par union et intersection des langages reconnus par les protocoles de population confiants utilise la même construction que pour la preuve

de la stabilité par union et intersection des langages reconnus par les protocoles de population classiques : le protocole calculant l'union (resp. l'intersection) de deux langages L et L' est construit comme à partir du produit cartésien des protocoles calculant les langages L et L'.

— À partir d'un Protocole confiant $(Q, I, \Sigma, Y, in, out, \delta)$ séparant A en calculant une fonction f, il est possible de construire un Protocole confiant $(Q, I, \Sigma, Y', in, out', \delta)$ séparant $\mathbb{N}^\Sigma \setminus A$ en changeant l'alphabet de sortie et la fonction de sortie de la manière suivante : $Y' = \{true, false\}$ et $out'(q) = true$ si et seulement si $out(q) \neq y_a$ où y_a est la valeur de sortie de Y telle que $A = f^{-1}(y_A)$.

— Pour chaque A_j, il existe un protocole de population confiant P_j le séparant. Nous construisons un Protocole confiant P comme le produit cartésien de tous les protocoles P_j. Pour tout $j \in J$, le protocole P peut séparer le sous-ensemble A_j en ne prenant en compte que la composante du protocole P correspondant au protocole P_j.

■

Par voie de conséquence, nous obtenons le théorème suivant.

Théoreme 6.3.1 *Soit $X = (X_1, \ldots, X_k)$ une partition finie de $\mathbb{N}^\Sigma$.*

Si toute partie X_i peut être définie par une combinaison booléenne de problème de seuil nul, alors X peut être séparée par un protocole de population confiant fort ou faible.

6.3.2 Faisons confiance à la sortie

Nous nous plaçons maintenant dans le cas spécial où l'ensemble des opinions est l'alphabet de sortie $I = Y$; autrement dit *out* est donc la fonction identité sur I. Nous nous référerons à ces protocoles sous le nom de *Protocoles de de Population confiant en leur sortie*. Notre but est de comprendre à quoi ressemblent les ensembles séparés par de tels protocoles.

Théoreme 6.3.2 *Une partition $(A_j)_{j\in J}$ de $\mathbb{N}^\Sigma$ telle que, pour tout $j \in J$, A_j peut être séparé par un même protocole de population confiant en sa sortie est telle que chaque sous-ensemble de la partition peut-être défini comme une combinaison booléenne de problème de seuil nul, les inégalités pouvant être strictes.*

Tout d'abord remarquons que chaque partie A_j correspond exactement à une sortie possible, c'est-à-dire à exactement une opinion du protocole. Ce fait justifie que par la suite nous indicerons les ensembles A_j par les opinions $i \in I$.

La preuve de ce théorème est basée sur les lemmes suivants, que nous démontrons d'abord.

Lemme 6.3.1 *Soit $(A_i)_{i\in I}$ une partition séparée par un protocole de population confiant en sa sortie.*

Pour tout $i \in I$, l'ensemble A_i vérifie les deux propriétés suivantes :
(C^+) : $\forall a, b \in A_i$, $a + b \in A_i$
(C^*) : $\forall a \in \mathbb{N}^\Sigma$, $\forall \lambda \in \mathbb{N} \setminus \{0\}$, $\lambda a \in A_i$ si et seulement si $a \in A_i$

Preuve. Soit $i \in I$ et $a, b \in A_i$.

Pour toute exécution équitable commençant avec la configuration initiale a, la population se stabilise sur une configuration où tout les agents ont l'opinion i. Cela signifie qu'il existe une configuration $\bar{a} \in \mathbb{N}^{Q_i}$ telle que $a \to^* \bar{a}$. Et de même, il existe une configuration $\bar{b} \in \mathbb{N}^{Q_i}$ telle que $b \to^* \bar{b}$. En utilisant ces exécutions finies, il est possible de construire une exécution allant de $a + b$ à $\bar{a} + \bar{b}$. Tout les agents sont dans un état appartenant à Q_i, et donc, même s'il y a encore des interactions possibles, ces interactions ne peuvent produire que des états de Q_i. Par suite, la configuration $\bar{a} + \bar{b}$ est stable. Par conséquence, $a + b$ appartient à A_i.

Soit $i \in I$, $a \in A_i$ et $\lambda \in \mathbb{N} \setminus \{0\}$.

Du premier point, il est possible de déduire que si $a \in A_i$ alors $\lambda a \in A_i$.

Montrons la réciproque. Comme les $(A_i)_{i\in I}$ forment une partition de l'espace, il existe des uniques indices $i, j \in I$ tel que $a \in A_i$ et $\lambda a \in A_j$. Comme $a \in A_i$, alors $\lambda a \in A_i$ et donc $i = j$. Ceci montre l'équivalence : $a \in A_i$ si et seulement si $\lambda a \in A_i$

■

Pour la suite, S sera un ensemble semi-linéaire de $\mathbb{N}^\Sigma$ qui sera identifié à $\mathbb{N}^d$ avec $d = |\Sigma|$. Par ailleurs, l'ensemble S vérifiera les propriétés (C^+) et (C^*) du lemme 6.3.1.

Le fait que S soit semi-linéaire se traduit par l'existence d'ensembles finis d'indices K et J_i pour $i \in K$ et des vecteurs v_j de $\mathbb{N}^d$ pour $j \in \bigcup_{i\in K} J_i$ tels que :

$$S = \bigcup_{i\in K} \left\{ u_i + \sum_{j\in J_i} a_j v_j \;\middle|\; a_j \in \mathbb{N} \right\}.$$

Lemme 6.3.2

$$S = \mathbb{N}^d \cap \bigcup_{i\in K} \left\{ (1+a)u_i + \sum_{j\in J_i} a_j v_j \;\middle|\; a, a_j \in \mathbb{Q}^+ \right\}.$$

Preuve. Soit $x \in \mathbb{N}^d \cap \bigcup_{i\in K} \left\{ (1+a)u_i + \sum_{j\in J_i} a_j v_j \;\middle|\; a, a_j \in \mathbb{Q}^+ \right\}$.

Il existe $i_0 \in K$, $a \in \mathbb{Q}^+$ et $a_j \in \mathbb{Q}^+$ pour tout $j \in J_{i_0}$ tels que $x = (1+a)u_{i_0} + \sum_{j\in J_{i_0}} a_j v_j$.

Pour tout $j \in J_{i_0}$, a_j est un rationnel positif donc il existe $p_j \in \mathbb{N}$ et $q_j \in \mathbb{N} \setminus \{0\}$ tels que

$a_j = \frac{p_j}{q_j}$. Et de même, il existe $p \in \mathbb{N}$ et $q \in \mathbb{N} \setminus \{0\}$ tels que $a = \frac{p}{q}$. Nous obtenons alors :

$$x = \left(1 + \frac{p}{q}\right) u_{i_0} + \sum_{j \in J_{i_0}} \frac{p_j}{q_j} v_j$$

Posons $y = ((p+q) \cdot \Pi_{j \in J_{i_0}} q_j).x$.

Pour tout $j \in J_{i_0}$, nous avons $q_j \geq 1$ et $q \geq 1$. Par suite, nous avons

$$(p+q) \cdot (\Pi_{j \in J_{i_0}} q_j) - 1 \geq 0.$$

Nous avons alors :

$$y - \Big((p+q) \cdot \Pi_{j \in J_{i_0}} (q_j - 1).u_{i_0}\Big) = u_{i_0} + \sum_{j \in J_{i_0}} (p_j \cdot q \cdot \Pi_{j' \neq j} q_{j'}) \cdot v_j.$$

Ce qui prouve que :

$$y - \Big((p+q) \cdot \Pi_{j \in J_{i_0}} (q_j - 1).u_{i_0}\Big) \in \left\{ u_{i_0} + \sum_{j \in J_i} a_j v_j \;\middle|\; a_j \in \mathbb{N} \right\}.$$

Puisque $u_{i_0} \in S$ et que S vérifie (C^*), nous avons :

$$\Big((p+q) \cdot \Pi_{j \in J_{i_0}} (q_j - 1)\Big) \cdot u_{i_0} \in S.$$

Comme y est la somme de deux éléments de S et que S vérifie (C^+), nous obtenons que $y \in S$.

Finalement, comme S vérifie (C^*), nous avons $x \in S$.

L'inclusion réciproque est une conséquence de l'inclusion $\mathbb{N} \subset \mathbb{Q}^+$.

En effet, nous avons :

$$S = \bigcup_{i \in K} \left\{ u_i + \sum_{j \in J_i} a_j v_j \;\middle|\; a_j \in \mathbb{N} \right\}.$$

Comme S vérifie (C^*) et (C^+), nous avons, pour tout i :

$$\{(1+a)u_i \mid a \in \mathbb{N}\} \subset S$$

et donc :

$$S = \bigcup_{i \in K} \left\{ u_i + \sum_{j \in J_i} a_j v_j \;\middle|\; a_j \in \mathbb{N} \right\} \cup \{(1+a)u_i \mid a \in \mathbb{N}\}$$

ce qui peut être réécrit :

$$S = \bigcup_{i \in K} \left\{ (1+a)u_i + \sum_{j \in J_i} a_j v_j \;\middle|\; a, a_j \in \mathbb{N} \right\}$$

Par suite, nous obtenons :

$$S = \mathbb{N}^d \cap \bigcup_{i \in K} \left\{ (1+a)u_i + \sum_{j \in J_i} a_j v_j \;\middle|\; a, a_j \in \mathbb{N} \right\}.$$

Et comme, $\mathbb{N} \subset \mathbb{Q}^+$, nous avons finalement :

$$S \subset \mathbb{N}^d \cap \bigcup_{i \in K} \left\{ (1+a)u_i + \sum_{j \in J_i} a_j v_j \;\middle|\; a, a_j \in \mathbb{Q}^+ \right\}.$$

Nous pouvons donc conclure que

$$S = \mathbb{N}^d \cap \bigcup_{i \in K} \left\{ (1+a)u_i + \sum_{j \in J_i} a_j v_j \;\middle|\; a_j \in \mathbb{Q}^+ \right\}.$$

■

Puisque S vérifie (C^+), pour tout i, l'élément $u_i + u_i$ appartient à S et donc peut s'écrire sous la forme $u_i + \sum_{j \in J_i} a_j v_j$ avec $a_j \in \mathbb{Q}^+$. Par suite, u_i est une combinaison linéaire positive des $(v_j)_{j \in J_i}$. Sans perte de généralité, nous supposerons donc que u_i appartient à l'ensemble des $(v_j)_{j \in J_i}$. Nous obtenons alors que

$$S = \mathbb{N}^d \cap \bigcup_{i \in K} \left\{ u_i + \sum_{j \in J_i} a_j v_j \;\middle|\; a_j \in \mathbb{Q}^+ \right\}.$$

Pour la suite de la preuve du théorème, nous aurons besoin d'une version forte du théorème de Caratheodory. La version originale peut s'énoncer comme suit :

Lemme 6.3.3 (Théorème de Caratheodory[Car35]) *Si $x \in \mathbb{Q}^d$ est une combinaison linéaire positive de $\{v_j\}_{j \in J_i}$, alors il existe un sous-ensemble linéairement indépendant $\{v_{l_1}, \ldots, v_{l_k}\} \subseteq \{v_j\}_{j \in J_i}$ tel que x est une combinaison linéaire positive de $\{v_{l_1}, \ldots v_{l_k}\}$.*

Pour notre preuve, nous avons besoin de contraindre le sous-ensemble linéairement indépendant de façon à garantir que le vecteur u_i y appartienne.

Lemme 6.3.4 (Théorème de Caratheodory Étendu) *Si $x \in \mathbb{Q}^d$ est une combinaison linéaire positive de $\{v_j\}_{j \in J_i}$, où il existe $v_{l_0} \neq 0$ et $\forall j \in J_i$, $v_j \in \mathbb{N}^d$, alors il existe un sous-ensemble linéairement indépendant $\{v_{l_0}, v_{l_1}, \ldots, v_{l_k}\} \subseteq \{v_j\}_{j \in J_i}$ tel que x est une combinaison linéaire positive de $\{v_{l_0}, v_{l_1}, \ldots v_{l_k}\}$.*

Preuve. Soit x une combinaison linéaire positive de k vecteurs $v_{l_1}, \ldots, v_{l_k}$ linéairement indépendants. Ce sont les k vecteurs donnés par la version original du théorème, par exemple. Nous avons des entiers positifs $(a_m)_{0\leq m\leq k}$ tels que $x = \sum a_m v_{l_m}$.

Si $v_{l_0} \in \{v_{l_1}, \ldots v_{l_k}\}$, alors le théorème étendu est vérifié.

Supposons que $v_{l_0} \notin \{v_{l_1}, \ldots v_{l_k}\}$.

Il y a alors deux cas :

$\{v_{l_0}, v_{l_1}, \ldots v_{l_k}\}$ est un ensemble linéairement indépendant. Dans ce cas, x est une combinaison linéaire positive de $\{v_{l_0}, v_{l_1}, \ldots v_{l_k}\}$ et alors le théorème étendu est vérifié.

$\{v_{l_0}, v_{l_1}, \ldots v_{l_k}\}$ n'est pas un ensemble linéairement indépendant. Dans ce cas là, il existe des entiers non tous nuls $(b_m)_{0\leq m\leq k}$ tels que $v_{l_0} = \sum_{1\leq m\leq k} b_m v_{l_m}$. Il convient de noter que les entiers b_m ne sont pas forcément positifs, par contre au moins un de ces entiers doit être positif puisque $v_{l_0} \geq 0$. Nous choisissons M tel que $b_M > 0$ et que $\frac{a_M}{b_M}$ est minimal parmi l'ensemble des valeurs $\frac{a_m}{b_m}$ pour les indices m tels que $b_m > 0$.

À partir de l'égalité $v_{l_0} = \sum_{1\leq m\leq k} b_m v_{l_m}$, nous pouvons obtenir l'égalité suivante $v_M = \frac{1}{b_M} v_{l_0} - \sum_{m\neq M} \frac{b_m}{b_M} v_{l_m}$. En remplaçant v_M par cette valeur, nous obtenons :

$$\begin{aligned} x = \sum a_m v_{l_m} &= \sum_{m\neq M} a_m v_{l_m} + a_M \left(\frac{1}{b_M} v_{l_0} - \sum_{m\neq M} \frac{b_m}{b_M} v_{l_m} \right) \\ &= \frac{a_M}{b_M} v_{l_0} + \sum_{m\neq M} \left(a_m - a_M \frac{b_m}{b_M} v_{l_m} \right) \end{aligned}$$

Pour achever la preuve, nous prouvons maintenant que $\left(a_m - a_M \frac{b_m}{b_M} v_{l_m} \right) \geq 0$ pour tout $m \neq M$.

— Si $b_m \leq 0$, alors $-a_M \frac{b_m}{b_M} \geq 0$ et donc $\left(a_m - a_M \frac{b_m}{b_M} v_{l_m} \right) \geq 0$.

— Si $b_m > 0$, et comme M minimise $\frac{a_M}{b_M}$, nous avons $\frac{a_m}{b_m} \geq \frac{a_M}{b_M}$ et donc nous avons $\left(a_m - a_M \frac{b_m}{b_M} v_{l_m} \right) \geq 0$.

Cela conclut la preuve que nous pouvons forcer l'appartenance de v_{l_0} à l'ensemble linéairement indépendant de vecteurs générant x.

■

Avec ce lemme, nous avons :

$$\left\{ \sum_{j\in J} a_j v_j \;\middle|\; a_j \in \mathbb{Q}^+ \right\} = \bigcup_{\substack{\{v_{l_1},\ldots,v_{l_k}\} \\ \text{linéairement indépendant}}} \left\{ \sum_{j\in\{l_1,\ldots,l_k\}} a'_j v_j \;\middle|\; a'_j \in \mathbb{Q}^+ \right\}$$

Par suite, nous pouvons écrire :

$$S = \mathbb{N}^d \cap \bigcup_{i\in K} \bigcup_{\substack{\{v_{l_1},\ldots,v_{l_k}\}\subset J_i \\ \text{linéairement indépendant}}} \left\{ u_i + \sum_{1\leq m\leq k} a_m v_{l_m} \;\middle|\; a_m \in \mathbb{Q}^+ \right\}$$

Considérons maintenant l'ensemble $V = \{v_{l_1}, \ldots, v_{l_k}\} \subset J_i$ de vecteurs linéairement indépendants et l'ensemble $H_i = \left\{ u_i + \sum_{1 \leq m \leq k} a_m v_{l_m} \;\middle|\; a_m \in \mathbb{Q}^+ \right\}$ pour $i \in K$ fixé. Nous avons $k \leq d$. Si $k < d$, nous pouvons compléter V avec $k - d$ vecteurs linéairement indépendants $w_{k+1}, \ldots w_d$ tels que $B = V \cup \{w_{k+1}, \ldots w_d\}$ est une base de $\mathbb{Q}^d$; et si $k = d$ alors on pose $B = V$.

Nous effectuons le changement de coordonnée suivant : la nouvelle base est B et la nouvelle origine est u_i. Soit $(x_1, \ldots, x_d)$ les nouvelles coordonnées de x dans ce système.

Nous avons :

$$x \in H_i \text{ si et seulement si } \begin{cases} x_m \geq 0 & \text{si } m \leq k \\ x_m = 0 & \text{sinon.} \end{cases}$$

Il existe donc des vecteurs $y_1, \ldots, y_d$ et des constantes $c_1, \ldots c_d$ tels que :

$$x \in H_i \text{ si et seulement si } \begin{cases} y_m.x \geq c_m & \text{si } m \leq k \\ y_m.x = c_m & \text{sinon.} \end{cases}$$

Autrement dit, chaque ensemble $\{u_i + \sum_{1 \leq m \leq k} a_m v_{l_m} | a_m \in \mathbb{Q}^+\}$ peut être caractérisé par une conjonction d'inégalités et égalités linéaires.

Comme chaque inégalité est définie à partir d'un sous-ensemble de vecteur d'une famille $(v_j)_{j \in J_i}$ pour un i donné. Pour tout i, la famille $(v_j)_{j \in J_i}$ est finie. Par suite, l'ensemble $\{u_i + \sum_{1 \leq m \leq k} a_m v_{l_m} | a_m \in \mathbb{Q}^+\}$ peut être caractérisé par un nombre fini de d'inégalités et égalités linéaires.

Par suite, l'ensemble S peut être caractérisé par une disjonction (pour chaque ensemble possible J_i) de disjonction (pour chaque sous-ensemble linéairement indépendant de vecteur de J_i) de conjonctions d'inégalités et d'égalités linéaires.

Nous devons encore prouver que le terme de droite de ces formules est 0 – ou peut être remplacer par 0.

D'abord, nous prouvons un lemme qui est une version faible de (C^*) pour notre ensemble :

Lemme 6.3.5 *Soit $V = \{v_j\}_{j \in J'}$ une famille de vecteurs telle que $u_i \in V$. Considérons $H_i = \left\{ u_i + \sum_{j \in J'} a_j v_j \;\middle|\; a_j \in \mathbb{Q}^+ \right\}$.*

Si $x \in H_i$, alors, pour tout $\lambda \in \mathbb{N} \setminus \{0\}$, nous avons $\lambda x \in H_i$.

Preuve. Soit $x \in H_i$ et $\lambda \in \mathbb{N} \setminus \{0\}$.

Nous pouvons écrire :

$$x = u_i + \sum_{j \in J'} x_j v_j.$$

Alors :

$$\lambda x = u_i + (\lambda - 1)u_i + \sum_{j \in J'} (\lambda x_j) v_j.$$

Par définition, $u_i + \sum_{j \in J'} (\lambda x_j) v_j$ appartient à H_i et $(\lambda - 1)u_i$ est soit égal à $\mathbf{0}$ soit appartient à l'ensemble $u_i + u_i\mathbb{N}$ qui est inclus dans H_i puisque $u_i \in V$

Et nous pouvons donc conclure que $\lambda x \in H_i$.

■

Soit $y_m \cdot x = c_m$ une des égalités servant à caractériser H_i. Comme $2x \in H_i$, nous avons

$$y_m \cdot 2x = c_m = 2(y_m \cdot x) = 2c_m.$$

Nous en déduisons que $c_m = 0$.

Soit $y_m \cdot x \geq c_m$ une des inégalités vérifiées par tout les éléments de H_i. Regardons séparément les cas où $c_m < 0$ et $c_m > 0$.

Cas $c_m < 0$. Soit $x \in H_i$ tel que $y_m \cdot x \in [c_m, 0[$. Il existe $\lambda \in \mathbb{N}$ tel que $\lambda y_m \cdot x < c_m$. Par suite, nous avons $\lambda x \notin H_i$, ce qui contredit le lemme 6.3.5 et le fait que $x \in H_i$. Finalement, nous en déduisons que : $y_m \cdot x \geq c_m$ si et seulement si $y_m \cdot x \geq 0$.

Cas $c_m > 0$. Soit $x \notin H_i$ tel que $y_m \cdot x \in]0, c_m[$. Il existe $\lambda \in \mathbb{N}$ tel que $\lambda y_m \cdot x \geq c_m$. Par suite, nous avons $\lambda x \in H_i$, ce qui contredit le lemme 6.3.5 et le fait que $x \notin H_i$. Finalement, nous en déduisons que : $y_m \cdot x \geq c_m$ si et seulement si $y_m \cdot x > 0$.

De cette étude de cas, nous pouvons en déduire le théorème suivant :

Théoreme 6.3.3 *Si S est un ensemble semi-linéaire vérifiant (C^+) et (C^*), alors S peut être décrit par une combinaison booléenne finie de formules de la forme $y \cdot x \propto 0$ avec $\propto \in \{\geq, >, =\}$.*

6.3.3 Description plus géométrique des ensembles calculés

Nous allons ici donner une description des sous-ensembles de $\mathbb{N}^\Sigma$ — identifié avec $\mathbb{N}^{|\Sigma|}$ — qui peuvent être calculés par la restriction de la sous-section précédente, les protocoles de population confiants en leur sortie.

Nous allons d'abord montrer les lemmes suivants.

Lemme 6.3.6 *Soit S un ensemble semi-linéaire vérifiant (C^+) et (C^*). On a :*

$$\mathbb{N}^d \cap \bigcup_{i \in K} \sum_{j \in J_i} v_j \mathbb{Q} \subseteq S \subseteq \mathbb{N}^d \cap \bigcup_{i \in K} \sum_{j \in J_i} v_j \mathbb{Q}^+.$$

Ce lemme peut être compris de la façon suivante : l'ensemble S est compris entre le cône généré par les vecteurs et son intérieur strict.

Il se déduit directement du lemme 6.3.2 – et de sa preuve.

Lemme 6.3.7 *Soit G un sous-espace vectoriel de $\mathbb{Q}^d$ et S un ensemble semi-linéaire de $\mathbb{N}^d$.*

L'ensemble $S \cap G$ est un ensemble semi-linéaire.

Preuve. Nous montrons d'abord cette propriété pour les sous-espaces vectoriels de dimension $d-1$.

Soit G un sous-espace vectoriel de dimension $d-1$ de $\mathbb{Q}^d$ et S un ensemble semi-linéaire de $\mathbb{N}^d$. Il existe $\alpha_1, \ldots, \alpha_d, \beta \in \mathbb{Q}$ tels que pour tout $x = (x_1, \ldots, x_d) \in \mathbb{Q}^d$

$$\sum_{1 \leq i \leq d} \alpha_i x_i = \beta \text{ si et seulement si } x \in G.$$

Nous en déduisons l'existence d'entiers $a_1, \ldots, a_d, b \in \mathbb{Z}$ tels que

$$\sum_{1 \leq i \leq d} a_i x_i = b \text{ si et seulement si } \sum_{1 \leq i \leq d} \alpha_i x_i = \beta \text{ si et seulement si } x \in G.$$

Nous en déduisons que $G \cap \mathbb{N}^d$ est un ensemble semi-linéaire.

Et par suite $(G \cap \mathbb{N}^d) \cap S = G \cap S$ est un ensemble semi-linéaire.

Nous étendons ce résultat à tout les sous-espaces vectoriels en remarquant que si G est un sous-espace vectoriel de dimension d' alors il existe $d - d'$ sous-espaces vectoriels $G_1, \ldots, G_{d-d'}$, tous de dimension $d-1$, tels que :

$$G = \bigcap_{1 \leq i \leq d-d'} G_i.$$

Nous avons donc

$$G \cap S = \bigcap_{1 \leq i \leq d-d'} (G_i \cap S).$$

Pour tout i,$G_i \cap S$ est un ensemble semi-linéaire puisque G_i est un sous-espace de dimension $d-1$ et S un ensemble semilinéaire. Il s'ensuit que $G \cap S$ est l'intersection d'ensemble semi-linéaire et c'est donc un ensemble semi-linéaire.

■

Un cône est l'intersection d'un nombre fini de demi-espaces.

Soit G un des hyperplans définissants le cône.

L'ensemble $S' = S \cap G$ est donc semi-linéaire, d'après le lemme 6.3.7. Et S' vérifie toujours les propriétés (C^+) et (C^*).

On peut dont encore appliquer le lemme 6.3.6, et ce de manière récursive tant que les deux inclusions sont strictes.

L'ensemble S peut alors être vu comme une union d'intérieur de cônes pointant en $\mathbf{0}$ où les cônes de dimension non maximale sont inclus dans les faces de cônes de dimension supérieure présents dans la liste. Le cône de dimension maximale est donné par l'ensemble des vecteurs générant l'ensemble semi-linéaire S.

Cette suite de cônes donne aussi une manière plus canonique d'écrire les formules calculables par notre modèle. Tout point x à l'intérieur d'un cône donné vérifie une formule

qui est la conjonction d'une conjonction d'égalité de la forme $a \cdot x = 0$ et d'une conjonction d'inégalité stricte de la forme $a \cdot x > 0$. Plus précisément, il existe une famille finie $(a_l)_{l \in H}$ de vecteurs d'entiers et une famille $(L_h, L'_h)_{h \in H'}$ de partition de H — telles que, pour tout $h \in H'$, nous avons $L, L' \subset K$ et $L \cap L' = \emptyset$ — tels que S peut s'écrire sous la forme :

$$S = \bigcup_{(L_k, L'_k)_{k \in K'}} \left(\bigcap_{l \in L_k} (a_l \cdot x = 0) \cap \bigcap_{l \in L'_k} (a_l \cdot x > 0) \right)$$

6.3.4 Interprétation en termes de fréquences

Il est possible de proposer une autre interprétation des prédicats calculés par cette variante des protocoles de population.

Proposition 6.3.2 (Stabilité par Multiplication et Division) *Les ensembles calculés par protocoles de population confiants forts et faibles sont stables par multiplication et division.*

Pour toute fonction f calculée par un protocole de population confiant fort (resp. faible), pour toute entrée non nulle $E \in \mathbb{N}^\Sigma$, pour tout entier $k \in \mathbb{N} \setminus \{0\}$,

$$f(E) = f(k \cdot E)$$

et si, pour tout $\sigma \in \Sigma$, $E(\sigma)$ est un multiple de k

$$f(E) = f\left(\frac{1}{k} E\right).$$

Preuve. Soit f une fonction calculée par un protocole de population confiant $(Q, I, \Sigma, in, Y, out, \delta)$. Soit E une entrée possible. Soit $k > 0$ un entier.

Il existe une exécution allant d'une configuration initiale correspondant à $in(E)$ à une configuration stable C_s. La configuration C_s est donc constituée d'agents ayant tous la même interprétation de sortie.

Nous pouvons alors construire une exécution allant d'une configuration initiale correspondant à $in(k \cdot E)$ – qui peut être vue comme k copies de $in(E)$ – à une configuration C'_s constituée de k copies de C_s. La configuration C'_s est donc constituée d'agents ayant tous la même interprétation de sortie et donc elle est stable.

L'interprétation de C'_s et de C_s sont les mêmes et donc :

$$f(E) = f(k \cdot E).$$

Si pour tout $\sigma \in \Sigma$, $E(\sigma)$ est un multiple de k, alors il existe E' telle que $E = k \cdot E'$ et en appliquant le résultat précédent à E', nous obtenons :

$$f(E) = f\left(\frac{1}{k} E\right).$$

■

Avec ce résultat, il devient très aisé de démontrer que certaines fonctions ne sont pas calculables par des protocoles de populations confiants :

— Est-ce que la lettre x est présente au moins 2 fois dans l'entrée ? (en d'autres termes $n_x \geq 2$) Nous considérons l'entrée $E = (x, x, y, y)$, nous avons $f(E) = true \neq false = f\left(\frac{1}{2}E\right)$.

— Est-ce que la lettre x est présente un nombre pair de fois dans l'entrée (en d'autres termes $n_x \equiv 0 \mod 2$) Nous considérons la entrée $E = (x, y, y)$, nous avons alors $f(E) = false \neq true = f(2 \cdot E)$.

Le précédent théorème nous amène à une nouvelle interprétation des ensembles calculés par des protocoles de population confiants : les prédicats calculables par protocoles de population confiants sont ceux qui sont basé sur la fréquence des lettres de l'entrée.

En effet il est possible de réécrire un problème de seuil nul

$$\sum_{\sigma \in \Sigma} g(\sigma) n_\sigma \geq 0$$

sous la forme

$$\sum_{\sigma \in \Sigma} g(\sigma) f_\sigma \geq 0$$

où $f_\sigma = \frac{n_\sigma}{n}$ et avec $n = \sum_\sigma n_\sigma$. La quantité f_σ correspond à la fréquence d'apparition de la lettre σ dans la population.

6.4 Résumé et conclusion

Dans ce chapitre, nous avons étudié une variante du modèle classique des protocoles de population où les agents possèdent une opinion, qui est fonction de l'état de l'agent. Plus précisément, nous nous somms restreint, dans ce chapitre, aux cas où l'opinion d'un agent est la sortie associée à son état. La relation de transition est restreinte de façon à ne permettre un changement d'état, ou d'opinion, que si les agents ont des opinions différentes. Nous avons pu caractériser les prédicats que pouvait reconnaître notre modèle : ce sont exactement les combinaisons booléennes de prédicats de seuil nul. Nous avons proposé une autre interprétation de ces ensembles reconnus comme des cônes de $\mathbb{N}^d$ pointant en $\mathbf{0}$, les faces sont incluses en partie ou non suivant un schéma récursif.

Deuxième partie

Couplages Autostabilisants

Chapitre 7

Problème du couplage en distribué

Ce chapitre introduit le problème du couplage sur un graphe et les notions d'algorithmique distribuée que nous utiliserons pour résoudre le problème du couplage. Dans un premier temps, nous définissons donc les notions de couplage maximum, maximal et d'approximation du couplage maximum. Ensuite, nous définissons notre cadre de travail en algorithmique distribuée. Ce chapitre possède aussi un état de l'art et où nous décrivons plus en détail deux algorithmes distribués qui calculent respectivement un couplage maximal et une $\frac{2}{3}$-approximation d'un couplage maximum. Nos contributions sur ces problèmes se trouvent dans les chapitres suivants, respectivement les chapitres 8 et 9 pour les problèmes du couplage maximal et de la $\frac{2}{3}$-approximation d'un couplage maximum.

7.1 Couplage

7.1.1 Définition

Un couplage est un des objets de la théorie des graphes les plus étudiés. Trouver un couplage revient à apparier les sommets en respectant la relation des arêtes et de telle façon qu'aucun sommet ne soit apparié à plusieurs autres sommets.

Définition 30 (Couplage) *Soit $G = (V, E)$ un graphe. Un* couplage *de G est un ensemble d'arêtes disjointes M.*

Formellement, M est un couplage si

$$(M \subset E) \wedge (\forall e_1 \neq e_2 \in M,\ e_1 = (v_1, u_1) \wedge e_2 = (v_2, u_2) \Rightarrow \{v_1, u_1\} \cap \{v_2, u_2\} = \emptyset)$$

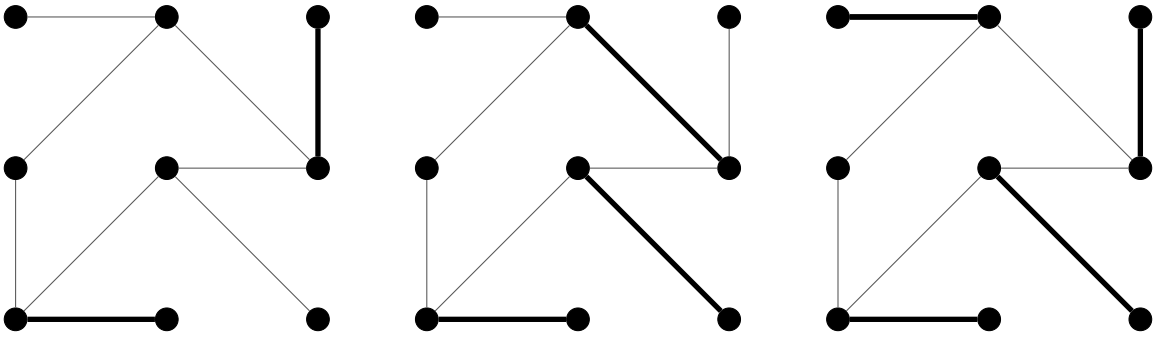
Un sommet apparié *est un sommet v tel qu'il existe une arête e incidente à v appartenant au couplage M.*

Un exemple de couplage pour un graphe est donné par la figure 7.1a. Dans la suite, nous nous intéresserons à certains couplages plus particuliers.

Définition 31 (Couplage maximal et maximum) *Soit $G = (V, E)$ un graphe.*

- *Un couplage M de G est un couplage* maximal *s'il n'existe pas de couplage M' tel que $M \subsetneq M'$*
- *Un couplage M est un couplage* maximum *s'il n'existe pas de couplage M' tel que $|M| < |M'|$.*

Autrement dit, un couplage maximal est un couplage qui est le plus grand pour la relation d'inclusion. Un tel couplage est présenté dans la figure 7.1b. Un couplage maximum est un couplage qui est le plus grand en nombre d'arêtes. C'est en particulier un couplage maximal, mais un couplage maximal n'est pas forcément maximum. Le couplage maximum n'est pas forcément unique. Un exemple de couplage maximum est présenté dans la figure 7.1c.



(a) Un exemple de couplage non maximal.

(b) Un exemple de couplage maximal.

(c) Un exemple de couplage maximum.

FIGURE 7.1 – Exemples de différents couplages sur un même graphe. Les arêtes appartenant aux couplages sont en gras.

Définition 32 *Un* chemin augmentant *est un ensemble d'arêtes $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-2}, v_{k-1}), (v_{k-1}, v_k)$ tel que $(v_2, v_3), (v_4, v_5) \ldots (v_{k-2}, v_{k-1})$ soient des arêtes de M et que v_1 et v_k ne soient pas appariés.*

On dira qu'un tel chemin augmentant est de longueur $k - 1$.

Un chemin augmentant est représenté dans la figure 7.2.

7.1.2 Propriétés du couplage maximal

Proposition 7.1.1 ([Pre99]) *Un couplage maximal est une $\frac{1}{2}$-approximation d'un couplage maximum.*

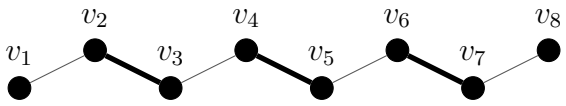


FIGURE 7.2 – Exemple de chemin augmentant de longueur 7. Les arêtes du couplage sont en gras.

Preuve. Soient M un couplage maximal et M' un couplage maximum.

Pour chaque arête e appartenant au couplage M, chacun des sommets de e appartient à au plus une arête du couplage M', puisque M' est aussi un couplage maximal.

Par suite, pour chaque arête du couplage M, on peut associer au plus deux arêtes du couplage M'. Par suite,

$$|M| \geq \frac{1}{2}|M'|$$

Cela signifie qu'un couplage maximal est une $\frac{1}{2}$-approximation d'un couplage maximum.

■

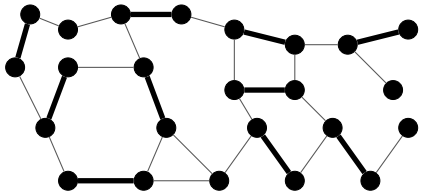
En rajoutant une propriété supplémentaire sur le couplage maximal, il est possible d'obtenir une meilleur approximation d'un couplage maximum.

Proposition 7.1.2 (à partir de [HK73]) *Un couplage maximal sans chemin augmentant de longueur 3 est une $\frac{2}{3}$-approximation d'un couplage maximum.*

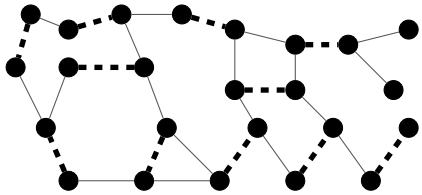
Preuve. Soit $G = (V, E)$ un graphe. Soit M un couplage maximal de G sans chemin augmentant de longueur 3. Soit M_{opt} un couplage maximum de G.

Nous considérons le graphe $G' = (V, M \Delta M_{opt})$ — où Δ désigne la différence symétrique entre deux ensembles. La figure 7.3 montre la construction d'un tel graphe. Le graphe G' a tous les sommets de G et uniquement les arêtes qui appartiennent au couplage M ou au couplage M_{opt} mais pas aux deux. Chacun de ses sommets a un degré d'au plus deux (une arête au plus pour chaque couplage). Le graphe G' est donc composé de cycles et de chaînes. Il y a autant d'arêtes du couplage M que du couplage maximum M_{opt} dans les cycles et dans les chaînes de longueur paire.

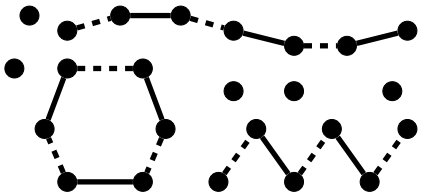
Les chaînes de longueur impaire correspondent en fait à des chemins augmentants pour le couplage M. Par propriété sur le couplage M, ces chaînes ont une longueur d'au moins 5 arêtes. Considérons une de ces chaînes de longueur $2\ell + 1$ (avec $\ell \geq 2$), le couplage M_{opt} participe pour $\ell + 1$ arêtes et le couplage M pour ℓ arêtes, sinon on pourrait augmenter la taille de M_{opt} ce qui serait une contradiction. Par suite, la proportion entre le nombre d'arêtes de M et le nombre d'arêtes de M_{opt} dans cette chaîne est de $\frac{\ell}{\ell+1} \geq \frac{2}{3}$.



(a) Un graphe G, les arêtes en gras sont des arêtes de M.



(b) Un graphe G, les arêtes en pointillés sont des arêtes de M_{opt}.



(c) Le graphe G' obtenu, les arêtes en gras viennent de M et les arêtes en pointillés de M_{opt}.

FIGURE 7.3 – Obtention du graphe $G' = (V, M_{opt}\Delta M)$ à partir de $G = (V, E)$

Par suite, dans chaque composante connexe de G' le nombre d'arêtes de M est au moins égal à $\frac{2}{3}$ du nombre d'arêtes de M_{opt}. Nous pouvons donc conclure que M est une $\frac{2}{3}$-approximation d'un couplage maximum.

■

7.2 Algorithmique distribuée

Un *système distribué* est un ensemble de nœuds (ou processus ou agents) distribués dans l'espace. Certains nœuds peuvent communiquer entre eux. On se placera uniquement dans des cas de communications par paire. La relation de communication peut être représentée par un graphe : les sommets sont les nœuds et deux nœuds sont reliés par une arête s'ils peuvent communiquer.

Un *algorithme distribué* est un ensemble fini de règles gardées. Une *règle* est composée d'une garde et d'une instruction :

$$Algorithme \equiv Regle\ 1|Regle\ 2|...|Regle\ n$$

$$Regle \equiv Garde \rightarrow Instruction$$

La *garde* est une expression booléenne qui dépend de la valeur des variables du nœud et de ses voisins, et dont le résultat conditionne l'exécution de l'*instruction*. L'*instruction* est une

séquence non vide d'actions internes et d'écritures sur les *variables locales*, c'est-à-dire les variables du nœud. Notons qu'un processus est une machine séquentielle, donc un processus ne peut exécuter qu'une seule règle gardée à la fois. Une règle est dite *activable* si un nœud vérifie sa garde.

Une *configuration* du système est la donnée pour chaque nœud de son état interne, c'est-à-dire de la valeur de ses variables locales. Une *transition* est un ensemble non vide d'évaluations de garde de règles activables et d'exécution d'action internes par nœud exécutant l'instruction d'une règle dont la garde a été vérifiée.

Une *exécution* est une suite alternée — finie ou infinie — de configurations et de transitions $\mathcal{E} = C_1, A_1, \ldots, C_i, A_i, \ldots$, telle que $\forall i \geq 1$, en appliquant les règles de A_i dans la configuration C_i on obtient la configuration C_{i+1}. Le plus souvent, on décrira une exécution uniquement comme une suite de configurations $C_1, \ldots, C_i, \ldots$ telles que, pour tout $i \geq 1$, il existe un ensemble de règles A_i qui, appliquées dans C_i, permettent d'obtenir la configuration C_{i+1}. Une exécution est dite *maximale* si elle est infinie ou si elle est finie et qu'il n'est plus possible d'activer de règle dans la dernière configuration. La *trace d'une exécution* $\mathcal{E} = C_1, A_1, \ldots, C_i, A_i, \ldots$ est la suite $A_1, \ldots, A_i, \ldots$ des règles appliquées.

Le *modèle de communication* décrit la manière dont s'effectuent les communications entre nœuds et quelles sont les variables locales auxquelles a accès un nœud. Le modèle renseigne quel agent peut lire ou écrire dans quel registre mémoire d'un nœud donné.

Dans le modèle, il peut, ou non, y avoir une hypothèse d'*anonymat*. On dira qu'un système est anonyme si les nœuds du réseau ne possèdent pas d'identifiants propres.

Dans la suite, nous considérerons les deux modèles de communication définis ci-après : *le modèle à état* et *le modèle à numérotation des ports*.

Définition 33 (Modèle à état) *Dans le* modèle à état*, chaque nœud possède une ou plusieurs variables locales sur lesquelles lui seul possède les droits d'écriture. Un nœud peut lire ses propres variables locales ainsi que celles de ses nœuds voisins.*

Nous présentons dans l'appendice A une façon de simuler ce modèle en utilisant le modèle suivant qui, lui, permet d'avoir une notion d'anonymat.

Définition 34 (Modèle à numérotation des ports) *Dans le* modèle à numérotation des ports*, un nœud possède une ou plusieurs variales locales par lien de communication. Un nœud v peut lire et écrire dans ses variales locales. Et pour chacun des voisins u de v, le nœud v peut lire dans les variales locales de u qui sont associées au lien de communication représenté par l'arête (v, u).*

Dans la figure 7.4, nous montrons l'exemple d'un nœud u qui a deux voisins v_1 et v_2. Le nœud u possède deux variables locales a et b, chacune étant associée à une arête (respectivement les arêtes (u, v_1) et (u, v_2)). Similairement, les nœuds v_1 et v_2 possèdent chacun une variable locale associée à l'arête qui les relie au nœud u (respectivement les variables locales d et c). Le nœud u peut lire et écrire dans les variables a et b, et il ne peut que lire dans les variables d et c, qui appartiennent à ses voisins.

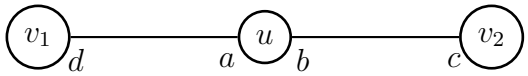


FIGURE 7.4 – Vue locale du nœud u.

7.2.1 Démons

Un *démon* — ou *adversaire* — est un prédicat sur les exécutions. Un démon peut être vu comme un adversaire qui choisit quelle règle est appliquée par quel nœud et à quel moment. Le démon doit cependant respecter des contraintes plus ou moins fortes dans ses choix. Certains démons doivent, par exemple, activer tous les nœuds en permanence alors que d'autres ne peuvent activer qu'un seul nœud à chaque tour. Agir sous un démon donné revient à éliminer certaines exécutions jugées non conformes parmi l'ensemble de toutes les exécutions. Nous verrons ici quelques démons qui nous servirons dans la suite, pour une étude plus complète se reporter à [Pil05].

Associée à la notion de démon, il y a la notion d'*atomicité*. L'atomicité nous dit quelle est la plus petite suite d'actions qui ne peut pas être interrompue, c'est-à-dire que si elle commence alors elle termine avant qu'une autre puisse commencer.

Il existe des démons plus ou moins forts. Le démon défini ci-après est le plus fort, il s'appelle le démon lecture/écriture et il n'obéit à aucune contrainte.

Définition 35 (Démon lecture/écriture) *Le* démon lecture/écriture *est le prédicat qui est constant en* $Vrai$.

Ce démon autorise toute les exécutions possibles. Rien n'est impossible. À chaque instant, ce démon active une règle pour un ou plusieurs nœuds. L'atomicité associée à ce démon est la lecture ou l'écriture d'une variable par un nœud. C'est la plus petite atomicité possible, elle est réduite à une action de la part d'un nœud. Ce démon est le démon le plus fort dans le sens où si un algorithme est correct pour ce démon alors il sera correct pour n'importe quel autre démon.

Il est souvent compliqué de travailler avec le démon lecture/écriture, et on préférera parfois utiliser des démons plus restrictifs comme le démon distribué.

Définition 36 (Démon distribué) *Le* démon distribué *est le prédicat suivant : la trace d'une exécution est une suite alternée d'évaluations d'un ensemble de gardes immédiatement suivi de l'exécution de l'ensemble des instructions associées à ces gardes.*

Sous ce démon, plusieurs nœuds peuvent chacun exécuter une règle simultanément, mais ils doivent terminer l'exécution de leur règle avant que d'autres règles commencent à être appliquées. L'atomicité associée est l'exécution d'une règle, évaluation de la garde et exécution des instructions associées. Cette atomicité est l'*atomicité de la règle gardée*. Il n'est pas

possible de commencer l'exécution d'une règle pour un nœud tant qu'il y a des nœuds qui ont commencé et n'ont pas fini d'exécuter une règle. Par contre, plusieurs nœuds peuvent exécuter en parallèle des règles, mais alors les évaluations des gardes sont simultanées et les exécutions des instructions associées aussi.

Définition 37 (Démon synchrone) *Le* démon synchrone *est le prédicat suivant : la trace d'une exécution est une suite d'activation d'une garde possible pour tout nœud pouvant activer une règle suivie de l'exécution de l'ensemble des instructions associées.*

Un démon synchrone est un démon distribué qui, à chaque pas, active tous les nœuds ayant une règle activable. Tous les nœuds exécutent leur programme interne à la même vitesse, ce qui justifie le nom de démon synchrone. L'atomicité associée à ce démon est la même que pour le démon distribué.

Définition 38 (Démon centralisé) *Le* démon centralisé *est le prédicat suivant : la trace d'une exécution est une suite d'évaluation d'une garde suivie de l'exécution de l'instruction associée.*

Le démon centralisé est un démon distribué qui active un unique nœud à la fois. Il utilise l'atomicité de la règle gardée mais interdit l'exécution de plusieurs règles en parallèle.

7.2.2 Auto-stabilisation

Dans un système distribué, la question de la terminaison de l'algorithme se pose différemment par rapport à un algorithme centralisé. Nous considérerons que l'algorithme a terminé lorsque le système rentre dans une configuration stable.

Définition 39 (Configuration stable) *Une* configuration stable *est une configuration telle qu'il n'existe pas de nœud activable, c'est-à-dire qu'aucun nœud ne vérifie la garde d'aucune règle.*

Une configuration stable est donc une configuration où il n'est plus possible de faire de pas de calcul. Lorsque le système atteint une configuration stable, il ne peut plus évoluer.

Définition 40 (Algorithmes Autostabilisant et Silencieux) *Un algorithme est* autostabilisant *pour une spécification $\mathcal{M}$ si, pour toute exécution maximale $\mathcal{E} = C_1 \dots C_i \dots$ de l'algorithme, il existe t_0 tel que le suffixe $C_{t_0}, C_{t_0+1}, \dots$ de l'exécution $\mathcal{E}$ vérifie la spécification $\mathcal{M}$.*

Un algorithme est silencieux *si pour toute exécution maximale $\mathcal{E} = C_1 \dots C_i \dots$ de l'algorithme, l'exécution $\mathcal{E}$ est finie est la configuration finale C_{t_0} est stable.*

Un algorithme autostabilisant est un algorithme qui quelque soit la configuration fera évoluer le système vers un ensemble de configurations donné ; ces configurations, ne sont pas nécessairement des configurations stables puisque des nœuds peuvent encore être activables.

7.2.3 Complexité

Dans un algorithme distribué, les ressources de temps et d'espace sont elles-mêmes distribuées. La mesure de la complexité d'un algorithme distribué requiert donc une approche spécifique.

Le calcul de la complexité en espace d'un algorithme peut ainsi être fait en considérant la somme des espaces utilisés par chacun des nœuds du réseau ou bien en considérant une borne globale sur l'espace utilisé par un nœud.

Le calcul de la complexité en temps d'un algorithme distribué peut également se faire de plusieurs manières, selon que l'on s'intéresse au temps que met le système à se stabiliser sur une réponse ou bien au nombre d'opérations réalisées par l'ensemble des nœuds. Le *temps de convergence* d'une exécution d'un algorithme autostabilisant est la longueur du préfixe de l'exécution avant le suffixe vérifiant la spécification.

Selon le démon utilisé, la notion naturelle de complexité à utiliser n'est pas la même.

Le temps d'exécution d'un algorithme distribué est le plus souvent compté ou bien en *pas*, ou bien en *étape*.

Définition 41 (Pas et étape) *Un* pas *est l'exécution par un nœud d'une action atomique.*

Dans un algorithme fonctionnant sous un démon distribué, une étape est une activation simultanée d'un sous-ensemble de nœuds activables.

Pour le démon lecture/écriture, chaque nœud effectue son calcul à une vitesse qui lui est propre et nous nous intéresserons au nombre de pas effectués par l'ensemble des nœuds du système.

Pour les démons distribués, démon synchrone compris, des nœuds effectuent des pas simultanément — un sous-ensemble des nœuds activables pour le démon distribué et tous les nœuds activables pour le démon synchrone. La complexité en temps que nous utiliserons pour ces démons sera le nombre d'étapes.

Les algorithmes que nous avons développés sont des algorithmes probabilistes. L'aspect probabiliste d'un algorithme peut se traduire de plusieurs manières : le temps de convergence peut être probabiliste ou la correction — l'algorithme peut éventuellement renvoyer un résultat qui n'est pas correct — peut être probabiliste (éventuellement les deux). Il existe plusieurs manières de mesurer la complexité d'un algorithme probabiliste. Si la convergence est probabiliste, alors la complexité peut être mesurée en utilisant ou bien l'*espérance* de la longueur d'une exécution — quand elle existe — ou bien en donnant une borne sur la taille de l'exécution *avec forte probabilité*, *id est* avec probabilité $1 - \varepsilon$ pour $0 < \varepsilon \leq 1$.

7.3 Des algorithmes pour un couplage

Dans la littérature, il existe beaucoup d'algorithmes de couplage autostabilisants. Selon le modèle de communication, le démon et l'atomicité considérés, de nombreuses variantes

ont été développées. Dans le chapitre suivant, nous présentons notre propre contribution. Notre algorithme a été développé à partir d'un algorithme plus ancien. Notre variante pour le premier cas se place dans un système de contraintes plus fortes.

7.3.1 Survol d'algorithmes de couplage présents dans la littérature

Pour le cas des graphes non-pondérés, Hsu et Huang [HH92] proposèrent un algorithme, ils prouvèrent une borne de $O(n^3)$ sur le nombre de pas sous le démon centralisé. L'analyse de la complexité a été complétée par Hedetniemi *et al.* [HJS01] et ramenée à $O(m)$ pas. Manne *et al.* [MMPT08] présentèrent un algorithme autostabilisant calculant une 2/3-approximation d'un couplage maximum. Nous nous intéresserons plus en détail à cet algorithme à la section 7.3.3 et au chapitre 9.

Pour le cas des graphes pondérés, Manne et Mjelde [MM07] ont presenté le premier algorithme autostabilisant calculant une $\frac{1}{2}$-approximation d'un couplage pondéré maximum. Ils ont pu déterminer que leur algorithme se stabilisait après au plus un nombre exponentiel de pas sous un démon distribué. Turau et Hauck [TH11] porposèrent une version modifiée de cet algorithme stabilisant en $O(nm)$ pas, toujours sous le démon distribué.

Les algorithmes mentionnés ci-dessus, excepté celui de Hsu et Huang [HH92], suppose une identité unique pour chacun des nœuds. L'algorithme de Hsu et Huang est le premier travaillant sur un réseau anonyme. Cet algorithme travaille sous le démon centralisé afin de pouvoir briser la symétrie. En effet, Manne *et al.* [MMPT09] prouvèrent que, dans un réseau anonyme, il n'existe pas d'algorithme autostabilisant déterministe calculant un couplage maximal sous le démon synchrone. C'est un résultat général qui est toujours vérifié sous différents démons distribués, sous différentes atomicités et différents modèles de communication.

Goddard *et al.* [GHJS08] ont proposé une méthode générale pour convertir n'importe quel algorithme déterministe anonyme stabilisant sous démon centralisé en un algorithme probabiliste stabilisant sous démon distribué, en utilisant seulement un espace supplémentaire constant et sans identifiant. L'espéreance du ralentissement est borné par $O(n^3)$ pas. La composition de ces algorithmes permet de calculer un couplage maximal en $O(mn^3)$ pas sur un réseau anonyme sous un démon distribué.

En ce qui concerne les réseaux anonymes, Gradinariu et Johnen [GJ01] ont proposé un algorithme autostabilisant probabiliste afin d'assigner à chaque nœud un identifiant local unique à distance 2. Ils ont utilisé cet algorithme pour exécuter l'algorithme de Hsu et Huang sous le démon distribué. Cependant, ils ne purent prouver que la finitude du temps de stabilisation. Chattopadhyay *et al.* [CHS02] ont amélioré ce résultat en donnant un algorithme de couplage maximal ayant une complexité de $O(n)$ étapes en moyenne sous un démon distribué équitable.

L'algorithme précédent, comme l'algorithme que nous présentons au chapitre 8, font

	[HH92, HJS01]	Composition de [HH92, HJS01] avec [GHJS08]	[GJ01]	[CHS02]	**Notre algorithme, chapitre 8**
Démon	centralisé	distribué	distribué	distribué équitable	**distribué**
Complexité	$O(m)$ pas	$O(mn^3)$ pas en moyenne	fini	fini	**$O(n^3)$ pas avec forte probabilité**

TABLE 7.1 – Résumé comparatif de différents algorithmes autostabilisants calculant un couplage maximal.

tout les deux l'hypothèse de l'anonymat et fonctionnent sous démon distribué. Cependant le premier, de [CHS02], suppose une propriété d'équité supplémentaire pour le démon, tandis que notre algorithme se place sous un démon distribué général. De plus, aucune borne sur la complexité de leur algorithme n'est donnée.

La table 7.1 compare les algorithmes sus-cités en terme de démon et de complexité. La dernière colonne donne les résultat pour l'algorithme que nous présentons au chapitre 8.

7.3.2 Un algorithme autostabilisant pour le couplage maximal

L'algorithme présenté ci-après a été proposé par Manne, Mjedle, Pilard et Tixeuil [MMPT09]. Il calcule un couplage maximal sur un graphe quelconque, ce qui correspond à une $\frac{1}{2}$-approximation d'un couplage maximum. Sa correction et sa complexité ont été étudiées sous le démon distribué, avec l'atomicité de la règle gardée.

Chaque agent i possède deux variables internes : p_i qui agit comme un pointeur vers un des voisins de i (ou vers personne éventuellement) — l'ensemble des voisins de i sera noté $N(i)$ — et m_i qui dit si i est apparié (marié) ou non. Cet algorithme comprend quatre règles :

Update met à jour la variable m_i si besoin

Mariage fait pointer i — en mettant à jour la valeur de p_i — s'il est non marié, vers un nœud j qui pointe vers i.

Seduction fait pointer i, s'il n'est pas marié, vers un de ses voisins qui n'est pas marié, lui non plus.

Abandon annule le pointeur de i si celui-ci pointe vers un nœud qui est marié avec un autre sommet.

La règle *Seduction* est construite pour éviter de créer des cycles de nœuds pointants chacun vers un autre nœud. Pour cela, un nœud i ne peut séduire un nœud j que si $i < j$. Les éventuels cycles pouvant être présents initialement sont brisés grâce à la règle *Abandon*.

Un nœud i abandonne sa proposition au nœud j si j est marié ailleurs ou si $j \leq i$ et que j ne propose pas à i.

Variables du nœud i :
$m_i \in \{true, false\}$
$p_i \in \{\perp\} \cup N(i)$

Règles :
Update :
si $m_i \neq PRmarried(i)$
alors $m_i := PRmarried(i)$
Mariage :
si $m_i = PRmarried(i) \wedge p_i = \perp \wedge \exists j \in N(i).p_j = i$
alors $p_i := j$
Seduction :
si $m_i = PRmarried(i) \wedge p_i = \perp \wedge \forall k \in N(i).p_k \neq i$
$\wedge \exists j. \in N(i).\,(p_j = \perp \wedge j > i \wedge \neg m_j)$
alors $p_i := \max\{j \in N(i).\,(p_j = \perp \wedge j > i \wedge \neg m_j)\}$
Abandon :
si $m_i = PRmarried(i) \wedge p_i = j \neq \perp \wedge p_j \neq i \wedge (m_j \vee j \leq i)$
alors $p_i := \perp$

Prédicat :
$PRmarried(i) \equiv \exists j \in N(i).\,(p_i = j \wedge p_j = i)$

Algorithme calculant un couplage maximal de [MMPT09].

Théoreme 7.3.1 ([MMPT09]) *Soit $G = (V, E)$ avec $|V| = n$ et $|E| = m$.*
Cet algorithme se stabilise sur une configuration renvoyant un coupage maximal après $O(m)$ pas sous le démon distribué.

La preuve de la borne sur la complexité s'effectue en comptant combien de fois chaque règle peut être appliquée par chaque nœud.

Cet algorithme n'est pas un algorithme anonyme. En effet, les identifiants des nœuds sont fortement utilisés dans la règle *Seduction* par exemple qui n'autorise un nœud à séduire un autre que si celui-ci a un identifiant plus élevé. Nous allons adapter cet algorithme aux réseaux anonymes dans le chapitre 8.

7.3.3 Un algorithme qui calcule une $\frac{2}{3}$-approximation

L'algorithme qui suit a été proposé par Manne, Mjelde, Pilard et Tixeuil [MMPT08]. Leur algorithme est un algorithme autostabilisant qui calcule une $\frac{2}{3}$-approximation d'un

couplage maximum.

Pour ce faire, il part d'un couplage maximal, obtenu par l'algorithme précédent par exemple. Il cherche tous les chemins augmentant de longueur 3 et en les utilisant pour augmenter le couplage, l'algorithme calcule un couplage qui est une $\frac{2}{3}$-approximation.

Pour chaque arête (u, v) appartenant au couplage maximal, les sommets u et v cherchent des sommets non-appariés voisins ; s'ils en trouvent chacun un, ces quatre sommets constituent un chemin augmentant de longueur 3. Pour cela, ces sommets appariés — l'ensemble des sommets appariés est noté $\mu(V)$ — stockent chacun dans deux variables locales (α, β) les identifiants de deux voisins non-appariés — l'ensemble des sommets non-appariés est noté $\sigma(V)$. Ensuite, ils peuvent comparer leurs voisins non-appariés et éventuellement ils peuvent en trouver chacun un différent pour s'apparier à nouveau (ou se rapparier). Si c'est le cas, ils ont trouvé un chemin augmentant de longueur 3. Ensuite l'algorithme ajoute les deux nouvelles arêtes et enlève (u, v).

Variables du nœud v :
$m_v \in \{\perp\} \cup N(v)$
$p_v \in \{\perp\} \cup N(v)$
$(\alpha_v, \beta_v) \in (\{\perp\} \cup N(v))^2$
$s_v \in \{true, false\}$

Règle pour les nœuds v non-appariés, *i.e.* **dans $\sigma(V)$:**
SingleNode
si $(p_v = \perp \wedge Min\{w \in N(v) \mid p_w = v\} \neq \perp) \vee p_v \notin \mu(N(v)) \cup \{\perp\} \vee$
$(p_v \neq \perp \wedge p_{p_v} \neq v)$
alors $p_v = Min\{w \in N(v) \mid p_w = v\}$

Règles pour les nœuds v appariés, *i.e.* **dans $\mu(V)$:**
Update
si $p_v \notin \sigma(N(v)) \cup \{\perp\} \vee ((\alpha_v, \beta_v) \neq BestRematch(v) \wedge (p_v = \perp \vee p_{p_v} \notin \{v, \perp\}))$
alors $(\alpha_v, \beta_v) = BestRematch(v)$
$(p_v, s_v) = (\perp, false)$

MatchFirst
si $AskFirst(v, m_v) \neq \perp \wedge (p_v \neq AskFirst(v, m_v) \vee s_v \neq (p_{p_v} = v))$
alors $p_v =$ **AskFirst**(v, m_v)
$s_v = (p_{p_v} = v)$

MatchSecond
si $AskSecond(v, m_v) \neq \perp \wedge s_{m_v} = true \wedge p_v \neq AskSecond(v, m_v)$
alors $p_v = AskSecond(v, m_v)$

ResetMatch
si $AskFirst(v, m_v) = AskSecond(v, m_v) = \perp \wedge (p_v, s_v) \neq (\perp, false)$
alors $(p_v, s_v) = (\perp, false)$

Fonctions :
BestRematch(v)
$a = Min\ \{u \in \sigma(N(v)) \wedge (p_u = \perp \vee p_u = v)\}$
$b = Min\ \{u \in \sigma(N(v)) \setminus \{a\} \wedge (p_u = \vee p_u = v)\}$
retourner (a, b)

AskFirst(v, w)
si $\alpha_v \neq \perp \wedge \alpha_w \neq \perp \wedge 2 \leq Unique(\{\alpha_v, \beta_v, \alpha_w, \beta_w\}) \leq 4$
si $\alpha_v < \alpha_w \vee (\alpha_v = \alpha_w \wedge \beta_v = \perp) \vee (\alpha_v = \alpha_w \wedge \beta_w \neq \perp \wedge v < w)$
retourner α_v
retourner $\perp$

AskSecond(v, w)
si $AskFirst(w, v) \neq \perp$
retourner $Min(\{\alpha_v, \beta_v\} \setminus \{\alpha_w\})$
retourner $\perp$

Algorithme calculant une $\frac{2}{3}$-approximation du couplage maximum de [MMPT08].

Les nœuds non-appariés, appelés *Single Node*, ont comme unique rôle d'accepter des proposition de rappariement. La priorité est donné aux nœuds dont l'identifiant est le plus élevé afin d'éviter les situations de blocage.

Soit l'arête (u, v) appartennant au couplage maximal utilisé comme entrée. En utilisant les fonction *AskFirst* et *AskSecond*, ils déterminent s'ils peuvent se rapparier et dans le cas où c'est possible dans quel ordre ils le font. Le premier à se rapparier est celui dont le nouveau voisin a l'identifiant le plus élevé. Quand le premier s'est rapparié, le second se rapparie à son tour. En cas de soucis, si un des deux nouveaux voisins s'est apparié avec un autre sommet, les rappariements de u et v sont annulés. Encore une fois la priorité est donnée aux sommets d'identifiant le plus élevé, cela évite les situations de blocage cyclique où chacun essaye de se rapparier et que cela bloque les autres.

Théoreme 7.3.2 ([MMPT08]) *Soit $G = (V, E)$ avec $|V| = n$ et $|E| = m$.*
Cet algorithme stabilise en $O(2^n)$ pas sous le démon distribué sur une configuration correspondant à une $\frac{2}{3}$-approximation d'un couplage maximum.

Cependant, les auteurs de [MMPT08] n'exhibent pas d'exécution de longueur exponentielle. Ils ne prouvent donc pas que l'algorithme sous démon distribué n'a pas une complexité

polynomiale. Dans le chapitre 9, nous allons exhiber une exécution sous-exponentielle. Nous modifierons ensuite cet algorithme en utilisant des méthodes probabilistes afin d'obtenir un algorithme polynomial calculant une $\frac{2}{3}$-approximation du couplage maximum.

Manne et al. prouvent également dans [MMPT08] qu'en se plaçant sous un démon plus restrictif, le démon centralisé, la complexité de l'algorithme peut descendre à $O(n^2)$ étapes, c'est-à-dire $O(n^3)$ pas.

Chapitre 8

Couplage dans un réseau anonyme

Dans ce chapitre, nous proposons un algorithme pour résoudre le problème du couplage maximal distribué dans un réseau anonyme. L'algorithme de couplage $\mathcal{A}_1$ présenté dans ce chapitre utilise le modèle à état décrit dans la section 7.2, et il est basé sur l'algorithme proposé par Manne et al. [MMPT09], ce dernier est décrit dans le chapitre précédent à la section 7.3.2. Notre algorithme calcule un couplage maximal en ajoutant une contrainte d'anonymat sur les agents. Pour rompre la symétrie due à l'anonymat, nous utilisons de l'aléatoire. Notre algorithme a une convergence probabiliste mais la correction est déterministe (quelque soit l'exécution qui termine, les configurations finales sont correctes). Nous verrons ensuite des bornes sur la complexité de notre algorithme en moyenne dans un premier temps puis avec forte probabilité. Dans chaque cas, nous obtenons une borne polynomiale sur la complexité, en $O(n^3)$ pas. Dans la dernière section, nous verrons une amélioration sur cette borne dans un cas particulier. Les résultats de ce chapitre ont été soumis mais ne sont pas encore publiés.

8.1 L'algorithme

Dans notre algorithme, dénommé $\mathcal{A}_1$ dans tout le chapitre, chaque nœud u possède une variable locale p_u représentant le nœud avec lequel u est apparié. Si u n'est pas apparié, la variable p_u prend alors la valeur $\perp$. L'utilisation d'une telle variable peut sembler en contradiction avec la contrainte d'anonymat. Un agent n'a pas accès à l'identifiant de ses voisins, et donc *a priori* il n'est pas capable d'assigner une valeur à sa variable locale et de comprendre le contenu de la variable de ses voisins. Il est possible de s'en sortir en utilisant les identifiants de port. Dans l'appendice A, nous présentons un algorithme qui permet de simuler le modèle à état dans un modèle à numérotation des ports. Dans la suite, nous nous utiliserons donc un modèle à état.

L'*algorithme de couplage* $\mathcal{A}_1$ nous assure qu'un couplage maximal est construit après

un certain temps. Plus formellement, la spécification $\mathcal{M}$ suivante est définie pour $\mathcal{A}_1$:

Définition 42 (Spécification $\mathcal{M}$) *Soit un graphe $G = (V, E)$.*

L'ensemble $M = \{(u, v),\ p_u{=}v \wedge p_v{=}u\}$ est un couplage maximal de G, i.e. *la formule suivante est vérifiée $\mathcal{M} = \mathcal{M}_1 \wedge \mathcal{M}_2 \wedge \mathcal{M}_3$ où :*

$\mathcal{M}_1$: $\forall (u, v) \in M,\ u \in V \wedge v \in V \wedge (u, v) \in E$ *(Consistance)*

$\mathcal{M}_2$: $\forall u, v, w \in V,\ (u, v) \in M \wedge (u, w) \in M \Rightarrow v = w$ *(Couplage)*

$\mathcal{M}_3$: $\forall (u, v) \in E, \exists w \in V,\ (u, w) \in M \vee (v, w) \in M$ *(Maximalité)*

Afin de simplifier les algorithmes et les raisonnements, nous supposerons que si un nœud u stocke une valeur v dans sa variable p_u telle que $v \notin V$ ou $v \notin N(u)$, alors u interprète sa valeur comme $\perp$.

L'algorithme $\mathcal{A}_1$ est composé de trois règles décrites ci-après. Si un nœud u ne pointe vers personne ($p_u = \perp$), alors qu'un de ses voisins pointe vers u, alors u accepte la proposition, c'est-à-dire que u pointe vers ce voisin (règle *Mariage*). Si un nœud u pointe vers un de ses voisins qui pointe lui-même vers un troisième nœud, alors u abandonne, c'est-à-dire que u réinitialise son pointeur à $\perp$ (règle *Abandon*). Si un nœud u ne pointe vers personne et qu'aucun de ses voisins ne pointe vers lui, alors u cherche un voisin qui pointe aussi vers personne. S'il existe un tel voisin, alors u pointe dans sa direction (règle *Séduction*). Cette séduction peut mener soit à un mariage entre ces deux nœuds (si le voisin de u accepte la proposition en exécutant la règle *Mariage*), soit à un abandon si le voisin de u opte finalement pour un autre nœud que u (u applique alors la règle *Abandon*).

Nous utiliserons dans la suite la fonction $choose(X)$ qui choisit uniformément un élément parmi un ensemble fini X.

Algorithme 1 ($\mathcal{A}_1$) *Le nœud u effectue une action selon l'une de ces règles :*

(Mariage) $(p_u = \perp) \wedge (\exists v \in N(u),\ p_v = u) \rightarrow p_u := v$

(Abandon) $(\exists v \in N(u),\ p_u = v \wedge p_v \neq u \wedge p_v \neq \perp) \rightarrow p_u := \perp$

(Séduction) $(p_u = \perp) \wedge (\forall v \in N(u),\ p_v \neq u) \wedge (\exists v \in N(u),\ p_v = \perp) \rightarrow$
Si $choose(\{0, 1\}) = 1$
alors $p_u := choose(\{v \in N(u),\ p_v = \perp\})$
sinon $p_u := \perp$

Afin de simplifier l'écriture de l'algorithme, nous ne précisons pas comment est choisi le nœud avec lequel se marie u dans la règle *Mariage*. En effet, la façon dont s'effectue ce choix n'a aucune conséquence en terme de complexité ou de correction de l'algorithme.

Considérons par exemple le graphe $G = (V, E)$ avec $V = \{0, .., 4\}$ et $E = \{(0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (3, 4), (2, 3)\}$. Une exécution possible $C_0C_1C_2C_3C_4C_5$ de l'algorithme $\mathcal{A}_1$ par les nœuds de ce graphe est représentée sur la figure 8.1.

Dans la configuration initiale, tous les nœuds pointent vers $\perp$. Au premier temps, le démon active tous les nœuds, tous vérifient la garde de la règle *Séduction* avec probabilité $\frac{1}{2}$, tous les nœuds sauf le 3 décident de "séduire" un voisin, et le nœud 3 décide de ne rien faire. Au deuxième temps, le démon active les nœuds 0 et 3, qui sont alors les seuls activables. Le nœud 0 applique la règle *Abandon* puisque le nœud 1 pointe vers 3 ; le nœud 3 applique la règle *Mariage* et il pointe ainsi vers le nœud 1, les nœuds 1 et 3 sont à présent appariés. Et rien ne pourra les séparer. Au troisième temps, le démon active uniquement le nœud 4 qui applique la règle *Abandon*. Au quatrième temps, le démon active seulement le nœud 2 qui applique aussi la règle *Abandon* puisque le nœud 3 est apparié. Au cinquième temps, le démon active tous les nœuds activables (à savoir les nœuds 0 et 2), tous deux appliquent la règle *Séduction* et s'apparient.

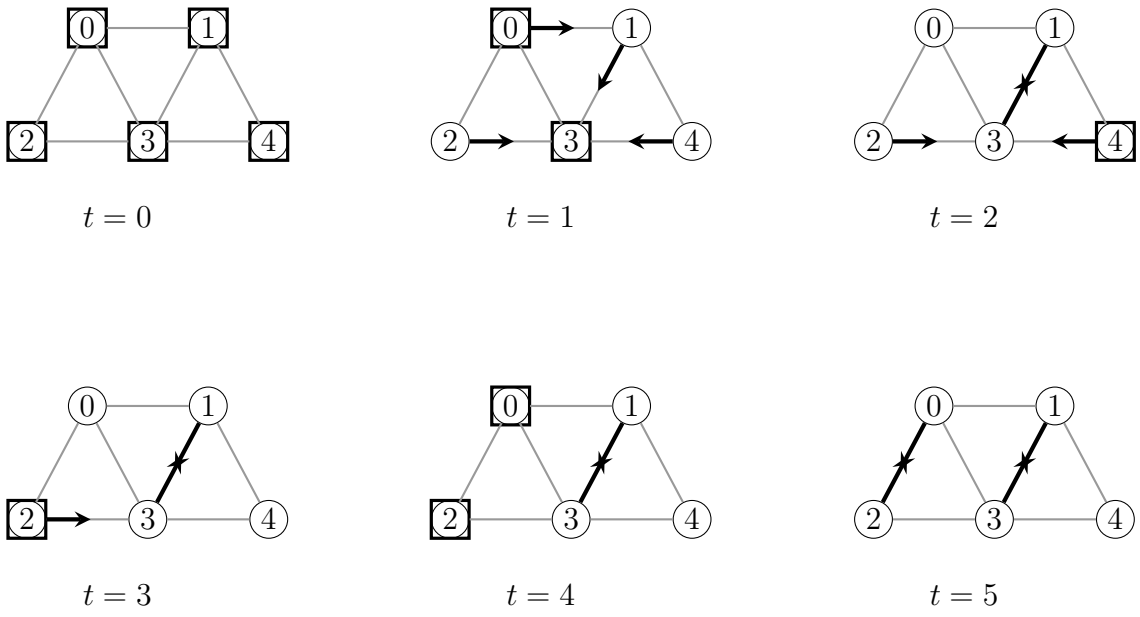


FIGURE 8.1 – Le graphe G durant une exécution dans les configurations C_t pour différents instants t. Les flèches représentent les pointeurs correspondant aux variables locales p. L'absence de flèche signifie que le nœud pointe vers $\perp$. Les nœuds encadrés sont ceux qui sont activés.

8.2 Correction de l'algorithme et vitesse de convergence

Nous vérifions d'abord que notre algorithme calcule bien un couplage maximal, c'est-à-dire qu'une configuration stable vérifie nécessairement la spécification $\mathcal{M}$.

Lemme 8.2.1 *Soit C une configuration de l'algorithme $\mathcal{A}_1$ où aucun nœud n'est activable. La configuration C vérifie la spécification $\mathcal{M}$.*

Preuve. Soit une configuration stable C.

Il n'est pas possible d'appliquer les règles *Abandon* ou *Mariage*. Donc pour tout nœud u de V, nous avons :

$$(p_u = \perp \wedge \forall v \in N(u),\, p_v \neq u) \vee (\exists v \in N(u),\, p_u = v \wedge p_v = u)$$

Et donc l'ensemble $M = \{(u,v) | p_u = v \wedge p_v = u\}$ est un sous-ensemble de E et par suite $\mathcal{M}_1$ est vérifiée.

Prouvons ensuite que, dans une configuration stable, M est un couplage maximal. Deux arêtes de l'ensemble M ne peuvent pas être adjacentes puisqu'un nœud u a un unique pointeur p_u. Par suite, $\mathcal{M}_2$ est vérifiée et M est donc un couplage. Puisqu'il n'est pas possible d'appliquer la règle *Séduction*, pour tout nœud u tel que $p_u = \perp$, tout voisin v de u pointe vers quelqu'un, c'est-à-dire que $p_v \neq \perp$ et donc v appartient à une arête de M. Et donc $\mathcal{M}_3$ est vérifiée et M est maximal.

■

D'après ce lemme, nous savons que si l'algorithme $\mathcal{A}_1$ se stabilise, alors il a calculé un couplage maximal du graphe.

Les preuves de la convergence et de la complexité de l'algorithme sont basées sur une fonction de potentiel. Pour définir cette fonction, nous avons tout d'abord besoin de définir les notions de *nœud célibataire*, d'*arêtes saturées* et de *presque arêtes saturées*.

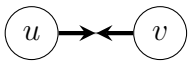
Posons $\mathcal{C}$ l'ensemble de toutes les configurations possibles de l'algorithme. Soit $C \in \mathcal{C}$ une configuration. Un nœud u pointant vers personne et n'ayant aucun voisin pointant vers lui est appelé nœud célibataire. Nous pouvons définir le prédicat *Single(u)* de la manière suivante : *Single(u)* $\equiv [p_u = \perp \wedge (\forall v \in N(u),\, p_v \neq u)]$. De plus, nous définissons l'ensemble $\mathcal{S}(C)$ comme étant l'ensemble des nœuds célibataires dans la configuration C. Remarquons qu'un nœud célibataire est activable si et seulement si au moins un de ses voisins pointe vers $\perp$.

Nous définissons maintenant les deux familles d'arêtes suivantes :

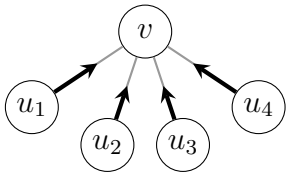
— une *arête saturée* est une arête (u,v) telle que $p_u = v$ et $p_v = u$

— une *arête presque saturée* est une arête (u,v) telle que $p_u = v$ et $p_v = \perp$. Un tel nœud v est appelé *indécis*.

La figure 8.2a montre un exemple d'une arête saturée et la figure 8.2b des exemples d'arêtes presque saturées, dans cet exemple toutes les arêtes presque saturées partagent le même nœud indécis v.



(a) Un exemple d'arête saturée



(b) Un exemple d'arêtes presque saturées ; le nœud v étant indécis

FIGURE 8.2 – Exemples d'arêtes saturée ou presque saturées ; les variables locales $p.$ sont représentées par des flèches

Un nœud qui est dans une arête saturée ne pourra plus jamais être activé. Puisque chaque nœud ne possède qu'un unique pointeur, deux arêtes saturées ne peuvent pas être adjacentes. Il ne peut donc y avoir plus de $n/2$ arêtes saturées.

L'activation d'un nœud indécis produit nécessairement une arête saturée. Un nœud indécis peut appartenir à beaucoup de presque arêtes saturées. Il ne peut pas y avoir plus de $n-1$ presque arêtes saturées.

Nous pouvons maintenant définir la fonction de potentiel $f : \mathcal{C} \to \mathbb{N} \times \mathbb{N}$ par $f(C) = (g, a)$ où g est le nombre de arêtes saturées dans la configuration C et a le nombre de presque arêtes saturées. Nous rappelons la définition de l'ordre lexicographique sur $\mathbb{N} \times \mathbb{N}$: on a $(a, b) \preceq (a', b')$ si une des deux conditions suivantes est vérifiée :

1. $a < a'$
2. $a = a'$ et $b \leq b'$

Lemme 8.2.2 *Soit $\mathcal{E} = C_0, A_0, \ldots, C_i, A_i, C_{i+1}, \ldots$ une exécution de $\mathcal{A}_1$.*
Nous avons :

$$\forall i \in \mathbb{N}, f(C_i) \preceq f(C_{i+1})$$

Preuve. Un nœud qui appartient à une arête saturée ne peut pas être activé, donc une arête saturée ne peut pas être détruite.

Par ailleurs le seul moyen de réduire la quantité de presque arêtes saturées est d'appliquer la règle *Mariage* sur un nœud indécis. En effet, si l'arête $e = (u, v)$ est une presque arête saturée, où v le nœud indécis de l'arête e, alors, au sein de l'arête e, le seul nœud activable est v et ce nœud ne peut exécuter qu'un pas de type mariage. Rappelons que v peut appartenir à plusieurs presque arêtes saturées. Ainsi l'exécution de la règle *Mariage*

par v détruit toutes les presque arêtes saturées incidentes à v et crée une nouvelle arête saturée.

Et donc la fonction f augmente dans tous les cas.

■

Maintenant, nous évaluons la probabilité qu'entre deux configurations la fonction f augmente strictement.

Lemme 8.2.3 *Soit $\mathcal{E} = C_0, A_0, \ldots, C_i, A_i, C_{i+1}, \ldots$ une exécution de $\mathcal{A}_1$.*

Pour tout $i \geq 0$, si A_i contient une action d'un nœud célibataire dans C_i, alors $f(C_i) \prec f(C_{i+1})$ avec probabilité plus grande que $\frac{1}{4}$. Plus formellement,

$$\forall i \in \mathbb{N}, Pr[f(C_i) \prec f(C_{i+1}) | \exists u \in \mathcal{S}(C_i),\ u \text{ est activé dans } C_i] \geq \frac{1}{4}$$

Preuve. Soit u un nœud célibataire activé dans C_i. Soit $F(u) = \{v \in N(u) \,|\, p_v = \perp\}$ et d sa cardinalité (*i.e.* $d = |F(u)|$).

Comme le nœud u est activé dans C_i, $F(u)$ est non vide et $d \geq 1$.

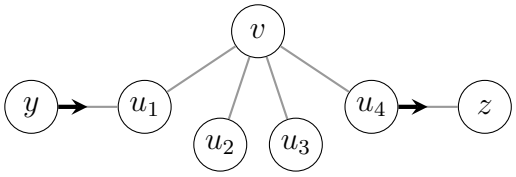


FIGURE 8.3 – Un exemple de configuration où le nœud v est un nœud célibataire avec $F(u) = \{u_1, u_2, u_3\}$.

Soit $v \in F(u)$.

Notons que le nœud u décide de séduire le nœud v avec probabilité $\frac{1}{2d}$. En effet, le nœud u séduit un de ses voisins avec probabilité $\frac{1}{2}$, de plus, si le nœud u décide de séduire (*i.e.* de changer sa variable local p_u), il séduit le nœud v avec probabilité $\frac{1}{d}$. Supposons que le nœud u séduit v, trois cas possibles se dessinent (voir figure 8.3) :

- $v \notin \mathcal{S}(C)$ **et** v **est activé dans** C_i **:** il existe un nœud s tel que $p_s = v$ et v doit appliquer la règle *Mariage*. Cette action génère une nouvelle arête saturée contenant v.
- $v \in \mathcal{S}(C)$ **et** v **est activé dans** C_i **:** le nœud v applique la règle *Séduction* et pointe vers $\perp$ avec probabilité $\frac{1}{2}$. Ainsi, l'arête (u, v) devient une arête presque saturée avec probabilité $\frac{1}{2}$.
- v **n'est pas actif dans** C_i **:** le nœud v continue à pointer sur $\perp$ (avec probabilité 1). Et ainsi, l'arête (u, v) devient une arête presque saturée avec probabilité 1.

Soit Pr_{uv} la probabilité de créer :
— une arête presque saturée entre u et v, v étant indécis,
— ou une arête saturée contenant v,
pendant $C_i \to C_{i+1}$ étant donné que :
— u est un nœud célibataire activé dans C_i et
— $v \in F(u)$ et
— $|F(u)| = d$.
De l'étude de cas précédente, il se déduit que $Pr_{uv} \geq \frac{1}{4d}$.

De plus, nous posons Pr_u comme étant la probabilité pour u de créer :
— une arête presque saturée entre u et un de ses voisins, le voisin en question étant le nœud indécis.
— une arête saturée contenant un voisin de u
durant $C_i \to C_{i+1}$ étant donné que u est un nœud célibataire activé dans C_i.

Nous avons :

$$Pr_u = \sum_{v \in F(u)} Pr_{uv} \geq \frac{1}{4}$$

Et donc :

$$Pr[f(C_i) \prec f(C_{i+1}) | \exists u \in \mathcal{S}(C_i) \text{ activé dans } C_i] \geq \frac{1}{4}$$

■

Cela signifie que si le démon active un nœud appartenant à $\mathcal{S}(C)$ alors, avec probabilité au moins $\frac{1}{4}$, la fonction de potentiel f augmente.

Lemme 8.2.4 *Dans toute exécution contenant $n+1$ pas, au moins un nœud indécis ou célibataire est activé.*

Preuve. Nous allons construire une exécution $\mathcal{E}$ la plus longue possible qui n'active ni de nœuds célibataires, ni de nœuds indécis. La règle *Séduction* ne peut être exécutée que par des nœuds célibataires et la règle *Mariage* ne peut être exécutée que par des nœuds indécis. La seule règle qui peut être exécutée dans $\mathcal{E}$ est par conséquent la règle *Abandon*. Si un nœud u exécute cette règle, alors ce nœud devient célibataire et cela ne change pas le statut des nœuds indécis ou célibataires voisins du nœud u. Chaque nœud ne peut donc appliquer la règle *Abandon* qu'une fois au plus. Par suite, $\mathcal{E}$ compte au plus n actions.

Pour conclure, si une exécution contient $n+1$ actions, alors l'une de ces actions est une application de la règle *Mariage* ou de la règle *Séduction*. Et une application d'une de ces règles implique qu'un nœud indécis ou célibataire a été activé.

■

Théoreme 8.2.1 *Sous un démon distribué, avec l'atomicité règle gardée, l'algorithme de couplage $\mathcal{A}_1$ est autostabilisant et silencieux pour la spécification $\mathcal{M}$. De plus, le temps moyen pour atteindre une configuration stable est $O(n^3)$ pas.*

Preuve. D'après lemme 8.2.1, une configuration stable pour l'algorithme $\mathcal{A}_1$ vérifie la spécification $\mathcal{M}$. Maintenant, nous nous concentrons sur le temps moyen pour atteindre une configuration stable.

Soit X la variable aléatoire du nombre de pas avant d'augmenter la fonction de potentiel f. D'après le lemme 8.2.4, une suite de $n+1$ actions contient au moins une action par un nœud indécis ou célibataire. Soit m cette action. Si m est une action par un nœud indécis, alors m est l'exécution de la règle *Mariage*, et donc la fonction de potentiel f augmente strictement durant m. Si m est un pas d'un nœud célibataire, alors d'après le lemme 8.2.3, la fonction de potentiel f augmente strictement avec probabilité au moins $\frac{1}{4}$ durant ce pas. Donc dans tous les cas, la fonction de potentiel f augmente strictement durant m avec probabilité au moins $\frac{1}{4}$. En cas d'échec (de l'augmentation de f), il peut y avoir au plus n nouveau pas avant une nouvelle activation d'un nœud célibataire ou indécis. Et il ne peut pas y avoir plus de n pas simultanés lorsqu'un nœud célibataire ou indécis est activé. Nous avons donc une suite d'épreuves de Bernouilli tous les $2n$ pas au plus, chacune ayant une probabilité de succès d'au moins $\frac{1}{4}$. Par suite, nous obtenons la borne suivante :

$$\mathbb{E}[X] \leq 4 \cdot 2 \cdot n$$

Par définition, la fonction f a $O(n^2)$ valeurs possibles. Par conséquent, l'espérance du temps pour la fonction de potentiel f d'atteindre sa valeur maximal est $O(n^3)$. Après cela l'algorithme peut exécuter au plus n pas. Ensuite l'algorithme doit nécessairement activer un indécis ou un célibataire, mais ce n'est pas possible puisque la fonction de potentiel f a atteint sa valeur maximale.

Pour conclure, l'algorithme $\mathcal{A}_1$ atteint une configuration stable après une exécution dont l'espérance du nombre de pas est $O(n^3)$.

■

Le résultat précédent donne une borne supérieure sur la complexité en moyenne. Maintenant, nous bornons le temps de convergence avec forte probabilité, c'est-à-dire avec une probabilité aussi proche de 1 que voulu.

Théoreme 8.2.2 *Soit $\varepsilon > 0$. Soit $k \geq max\{4(n+1)^3, -32(n+1)\ln\varepsilon\}$.*

Après k pas de calcul, l'algorithme a convergé avec probabilité supérieure à $1-\varepsilon$.

Preuve. D'abord, nous rappellons l'inégalité d'Hoeffding appliquée aux variables aléatoires de Bernouilli uniformément distribuées X_1, X_2, ... X_ℓ qui prennent la valeur 1 avec probabilité $\frac{1}{4}$. Soit S_ℓ la variable aléatoire définie par $S_\ell = \sum_{i=1}^{\ell} X_i$, en d'autres termes

S_ℓ correspond aux nombre de succès parmi ℓ essais. D'après l'inégalité de Hoeffding, nous avons, pour tout $\varepsilon' > 0$,

$$Pr[S_\ell \leq (\frac{1}{4} - \varepsilon')\ell] \leq e^{-2\varepsilon'^2\ell}$$

Ensuite, nous calculons la probabilité de ne pas atteindre une configuration stable en au plus k pas de calcul. D'après le lemme 8.2.4, dans toute exécution de $n+1$ pas, au moins un nœud indécis ou célibataire est activé. Donc le nombre minimum de fois où un nœud indécis ou célibataire est activé pendant k pas est $\lfloor \frac{k}{n+1} \rfloor = \ell$. D'après le lemme 8.2.3, lorsqu'un nœud célibataire ou indécis est activé, la fonction de potentiel f augmente strictement avec probabilité au moins $\frac{1}{4}$. Cette activation peut être vue comme une distribution de Bernoulli qui prend la valeur 1 avec probabilité $\frac{1}{4}$.

Soit C_k l'événement *"L'algorithme a convergé en k pas"*. Soit F_k la variable aléatoire du nombre de fois où la fonction de potentiel f a augmenté (strictement) durant les k premiers pas de l'exécution. Notons que $\neg C_k$ implique $F_k < \frac{n^2}{2}$, puisque la fonction f peut prendre au plus $\frac{n}{2} \cdot n$ valeurs différentes.

$$Pr[\neg C_k] \quad \leq \quad Pr\left(F_k < \frac{n^2}{2}\right)$$

Soit Y_i la variable aléatoire comptant le nombre d'augmentations de la fonction f en i activations de nœuds célibataires ou indécis.

$$Pr[\neg C_k] \quad \leq \quad Pr\left(Y_\ell < \frac{n^2}{2}\right) \text{ avec } \ell = \left\lfloor \frac{k}{n+1} \right\rfloor \tag{8.1}$$

$$\leq \quad \sum_{i=0}^{\frac{n^2}{2}-1} \binom{\ell}{i} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{\ell-i} \tag{8.2}$$

Cette valeur représente la probabilité que moins de $\frac{n^2}{2}$ variables de Bernoulli indépendantes de paramètre $\frac{1}{4}$ conduisent à un résultat positif sur ℓ essais. L'équation (8.1) peut être réécrite comme suit :

$$Pr[\neg C_k] \quad \leq \quad Pr\left(S_\ell < \frac{n^2}{2}\right) \tag{8.3}$$

Nous appliquons l'inégalité d'Hoeffding, avec $\varepsilon' = \left(\frac{1}{4} - \frac{n^2-2}{2\ell}\right)$:

$$Pr\left(S_\ell \leq \frac{n^2}{2} - 1\right) \quad \leq \quad e^{-2\left(\frac{1}{4} - \frac{n^2-2}{2\ell}\right)^2 \ell} \tag{8.4}$$

Notons $A = 2\left(\frac{1}{4} - \frac{n^2-2}{2\ell}\right)^2 \ell$. Les équations (8.3) et (8.4) peuvent être reformulées de la façon suivante :

$$Pr[\neg C_k] \leq e^{-A} \tag{8.5}$$

Ainsi, une condition suffisante pour que l'algorithme converge après k pas avec probabilité plus grande que $1 - \varepsilon$ est :

$$Pr[C_k] \geq 1 - \varepsilon \;\Leftrightarrow\; Pr[\neg C_k] \leq \varepsilon$$

Or, nous avons :

$$e^{-A} \leq \varepsilon \;\Leftrightarrow\; A \geq -ln\,\varepsilon \tag{8.6}$$

Et donc d'après les équations (8.5) et (8.6), nous avons :

$$A \geq -ln\,\varepsilon \Rightarrow Pr[\neg C_k] \leq \varepsilon \tag{8.7}$$

Il faut maintenant choisir une bonne valeur pour k pour que $A \geq -ln\,\varepsilon$.
Posons $k = \alpha\frac{(n+1)^3}{2}$, où $\alpha \geq 8$, et donc $\ell = \frac{\alpha(n+1)^2}{2}$. Nous obtenons alors :

$$\frac{2-n^2}{2\ell} = \frac{3+2n}{\alpha(n+1)^2} - \frac{1}{\alpha}$$

Et donc :

$$A = \alpha(n+1)^2\left(\frac{1}{4} - \frac{1}{\alpha} + \frac{3+2n}{\alpha(n+1)^2}\right)^2$$

Puisque $\alpha \geq 8$, nous avons :

$$\frac{1}{4} - \frac{1}{\alpha} \geq \frac{1}{8} \text{ et donc } \frac{1}{4} - \frac{1}{\alpha} + \frac{3+2n}{\alpha(n+1)^2} > \frac{1}{8} \text{ et } \left(\frac{1}{4} - \frac{1}{\alpha} + \frac{3+2n}{\alpha(n+1)^2}\right)^2 > \frac{1}{64}$$

Donc :

$$\alpha(n+1)^2\left(\frac{1}{4} - \frac{1}{\alpha} + \frac{3+2n}{\alpha(n+1)^2}\right)^2 > \frac{\alpha(n+1)^2}{64} \Rightarrow A \geq \frac{\alpha(n+1)^2}{64}$$

Étant donné que $\alpha \geq \frac{-64ln\,\varepsilon}{(n+1)^2}$, nous avons :

$$\frac{\alpha(n+1)^2}{64} \geq -ln\,\varepsilon \Rightarrow A \geq -ln\,\varepsilon$$

En utilisant l'équation (8.7), nous obtenons :

$$\alpha \geq \max\left\{8, \frac{-64ln\,\varepsilon}{(n+1)^2}\right\} \Rightarrow Pr[\neg C_k] \leq \varepsilon$$

Ainsi, pour $\alpha \geq \max\left\{8, \frac{-64ln\,\varepsilon}{(n+1)^2}\right\}$, nous obtenons la convergence avec probabilité plus grande que $1-\varepsilon$.

Finalement, puisque $k = \alpha\frac{(n+1)^3}{2}$, pour $k \geq \max\{4(n+1)^3, -32(n+1)\ln\,\varepsilon\}$, l'algorithme a convergé après k pas avec probabilité plus grande que $1-\varepsilon$.

■

8.3 Complexité dans un cas particulier

Nous allons nous concentrer sur le cas où :

1. le graphe de communication est le graphe complet de n sommets : $G = (\{1,\ldots,n\}, \{(i,j), 1 \leq i < j \leq n\})$,
2. le démon utilisé est le démon synchrone, il active en même temps tous les processus qui peuvent exécuter une règle.

Dans un graphe complet de n sommets, un couplage maximal possède $\lfloor\frac{n}{2}\rfloor$ arêtes et ainsi il couvre $2\cdot\lfloor\frac{n}{2}\rfloor$ sommets. Nous définissons la fonction de potentiel correspondant aux nombre d'arêtes dans le couplage. Pour cela, nous définissons $f(C)$ de la façon suivante : $f(C) = n_g$ pour toute configuration C ayant n_g arêtes saturées.

Donc, pour une configuration donnée C, si $f(C) < \lfloor\frac{n}{2}\rfloor = f_{max}$ alors C n'est pas une configuration stable. Cela signifie que des sommets sont activables.

Lemme 8.3.1 *Pour tout entier $n \geq 1$,*

$$\left(1-\frac{1}{2n}\right)^n \leq \frac{5}{8} \tag{8.8}$$

Preuve. Dans un premier temps, nous allons prouver que la suite $u_n = \left((1-\frac{1}{2n})^n\right)_{n\geq 1}$ est croissante. Pour cela posons $f(x) = \left(1-\frac{1}{2x}\right)^x$ pour $x \geq 1$ et calculons $f'(x)$.

$$f'(x) = \left(\ln(1-\frac{1}{2x}) + \frac{1}{x-\frac{1}{2}}\right)\cdot\left(1-\frac{1}{2x}\right)^x \tag{8.9}$$

Comme $x \geq 1$, $(1-\frac{1}{2x})^x > 0$. Donc le signe de la fonction $f'(x)$ est le signe de la fonction $g(x) = \left(\ln(1-\frac{1}{2x}) + \frac{1}{x-\frac{1}{2}}\right)$. La fonction logarithme ln est concave, et donc sa courbe est située sous sa tangente en 1. Nous avons donc que, pour tout $z > 0$, $\ln z \leq z-1$ En posant

$z = \frac{1}{y}$, nous obtenons que pour tout $y > 0$ nous avons $ln(y) \geq 1 - \frac{1}{y}$. Nous obtenons alors :

$$\begin{aligned} \ln(1 - \frac{1}{2x}) + \frac{1}{x - \frac{1}{2}} &\geq 1 - \frac{1}{1 - \frac{1}{2x}} + \frac{1}{x - \frac{1}{2}} \\ &\geq \frac{1}{2x - 1} \\ \text{et donc } g(x) &> 0 \end{aligned}$$

Donc la suite u_n est suite croissante. Maintenant, nous allons prouver qu'elle est bornée supérieurement : il suffit de calculer la limite de u_n quand n tends vers $+\infty$.

Nous rappellons le développement limité au voisinage de 0 à l'ordre 1 suivant :

$$\ln(1 - y) = -y + \underset{y \to 0}{o}(y).$$

Donc nous avons $\underset{n \to +\infty}{lim} \left(n \cdot \ln(1 - \frac{1}{2n}) \right) = -\frac{1}{2}$. Ce qui nous permet de conclure que

$$\underset{n \to +\infty}{lim} (1 - \frac{1}{2n})^n = \frac{1}{\sqrt{e}}$$

Comme $\frac{1}{\sqrt{e}} < \frac{5}{8}$, nous obtenons donc $(1 - \frac{1}{2n})^n \leq \frac{5}{8}$.

■

Lemme 8.3.2 *Soit C une configuration ayant n_I nœuds indécis et n_C nœuds célibataires et telle que $f(C) < f_{max}$. Pour $i = 1, 2, 3$, nous notons C_i la configuration obtenue après i activations de tous les nœuds activables.*

1. *Si $n_I > 0$, alors $f(C_1) > f(C)$*
2. *Si $n_C > 1$, alors $Pr[f(C_2) > f(C)] \geq \frac{3}{16}$*
3. *Si $n_I = 0$ et $n_C \leq 1$, alors $Pr[f(C_3) > f(C)] \geq \frac{3}{16}$*

Preuve.

1. Supposons que la configuration C possède n_I nœuds indécis avec $n_I > 0$. Comme tous les nœuds indécis sont activés, chacun d'entre-eux n'applique la règle qu'après l'activation de ces nœuds. Chaque agent indécis applique la règle *Mariage*, et ainsi chacun forme une nouvelle arête saturée en un pas. Donc on a

$$f(C_1) \geq f(C) + n_I > f(C).$$

2. Supposons maintenant que la configuration C ne possède pas de nœud indécis. Supposons qu'elle possède n_C nœuds célibataires avec $n_C > 1$.

   Cela signifie qu'il y a au moins deux sommets célibataires. Les sommets célibataires vont tous exécuter la règle *Séduction*. Comme il n'existe pas de sommet indécis, chaque nœud célibataire (s'il modifie la valeur de sa variable locale p) va choisir la valeur de sa variable parmi les autres célibataires.

   Soit u un nœud célibataire. Nous allons trouver une borne inférieure de la probabilité de l'événement $\mathcal{E}$: le nœud u devient indécis dans C'.

   Pour cela, il faut tout d'abord, que le nœud u choisse de ne pas modifier la valeur de sa variable locale p. Il le fait avec la probalité $1/2$. De plus, il faut qu'au moins autre sommet célibataire dans C pointe sur u dans C'. Il est plus facile de calculer la probabilité de l'événement $\mathcal{E}'$ suivant : aucun nœud célibataire ne choisit le nœud u.

$$Pr(\mathcal{E}) = \frac{1}{2} \cdot (1 - Pr(\mathcal{E}')) \tag{8.10}$$

   Soit v un autre nœud célibataire. Nous allons calculer $\mathcal{E}'_v$ la probabilité que v ne choisisse pas le nœud u.

$$Pr(\mathcal{E}'_v) = \frac{1}{2} + \frac{1}{2} \cdot \left(\frac{n_C - 2}{n_C - 1}\right) = 1 - \frac{1}{2} \cdot \frac{1}{n_C - 1}$$

   Le premier terme de la somme correspond à la probabilité que v choisisse de ne pas modifier sa valeur p_v et le second terme correspond à la probabilité de ne pas choisir u pour la valeur p_v sachant que la valeur p_v est modifiée.

   Donc, maintenant, on peut calculer $Pr(\mathcal{E}')$

$$Pr(\mathcal{E}') = \prod_{0<i\leq n_C-1} \left(1 - \frac{1}{2} \cdot \frac{1}{n_C - 1}\right) = \left(1 - \frac{1}{2(n_C - 1)}\right)^{n_C-1}$$

   Par le lemme 8.3.1, on obtient $Pr(\mathcal{E}') \leq \frac{5}{8}$ et donc $(1 - Pr(\mathcal{E}') \geq 1 - \frac{5}{8}$. Donc, l'équation (8.10) peut être réécrite de la façon suivante :

$$Pr(\mathcal{E}) = \frac{1}{2} \cdot (1 - Pr(\mathcal{E}')) \geq \frac{3}{16} \tag{8.11}$$

   Avec probabilité $\frac{3}{16}$, au moins un nœud indécis est créé en un pas, lors du pas suivant il appliquera nécessairement la règle $Mariage$ et créera ainsi une nouvelle arête saturée. Nous avons donc :

$$Pr[f(C_2) > f(C)] \geq \frac{3}{16}$$

3. Supposons que $n_I = 0$ et $n_C \leq 1$. La configuration C est instable, et donc des nœuds peuvent être activés. Comme il n'y a pas d'indécis et au plus un célibataire, les seuls nœuds pouvant être activés sont des nœuds pointant vers un nœud qui pointe lui-même vers un troisième nœud. Lors de l'activation, ces nœuds appliqueront la règle *Abandon* et ils créeront autant de nœuds célibataires dans la configuration C_1. Comme il n'y a que la règle *Abandon* qui a été appliquée, il n'existe pas de sommet dans C_1 pouvant appliquer cette règle ; il n'y a pas non plus de création de sommet indécis. Comme $f(C) = f(C_1) < f_{max}$, il existe donc au moins deux sommets célibataires dans C_1. La configuration C_1 vérifie donc le second point de ce lemme.

   Nous en déduisons donc que :

$$Pr[f(C_3) > f(C)] \geq \frac{3}{16}.$$

■

Il est possible de condenser ce lemme sous la forme :

Lemme 8.3.3 *Soit C une configuration telle que $f(C) < f_{max}$. Nous notons C' la configuration obtenue après 3 activations de tous les nœuds activables. Nous avons :*

$$Pr[f(C') > f(C)] \geq \frac{3}{16}$$

Cela signifie que si la fonction de potentiel peut encore augmenter, alors elle le fera en au plus 3 activations de tous les agents activables avec une probabilité supérieure à $\frac{3}{16}$.

Théoreme 8.3.1 *Soit $\varepsilon > 0$. Soit $k \geq max\{16n^2, -192n \ln \varepsilon\}$.*

Après k pas de calcul, l'algorithme a convergé avec probabilité supérieure à $1 - \varepsilon$ sur le graphe complet sous le démon synchrone.

Preuve. Soit C une configuration arbitraire telle que $f(C) < f_{max}$ et C' la configuration après 3 activations du démon.

Rappellons $f(C) = \frac{n_g}{2}$ avec n_g nœuds adjacents à une arête saturée. Au pire $3(n - n_g)$ actions ont été réalisées pour transformer C en C'.

En appliquant le lemme 8.3.3, la probabilité que $f(C') > f(C)$ est plus grande que $\frac{3}{16}$.

Ensuite le raisonnement est le même que pour la preuve du théorème 8.2.2. Les calculs étant légèrement différents, ils sont présentés intégralement ci-après.

D'abord, on rappelle l'inégalité de Hoeffding appliquée aux variables aléatoires indépendantes d'une distribution de Bernouilli X_1, X_2, ... X_ℓ qui prennent la valeur 1 avec probabilité $\frac{3}{16}$, et 0 sinon. Soit S_ℓ la variable aléatoire telle que $S_\ell = \sum_{i=1}^{\ell} X_i$ (c'est le nombre de succès en ℓ essais). L'inégalité d'Hoeffding, nous donne donc pour $\varepsilon_0 > 0$,

$$Pr[S_\ell \leq (\frac{3}{16} - \varepsilon_0)\ell] \leq e^{-2\varepsilon_0^2 \ell}$$

Ensuite on calcule la probabilité de ne pas atteindre une configuration stable en au plus k étapes.

À chaque instant, la fonction de potentiel f a une possibilité d'augmenter au cours des 3 prochaines étapes. Donc le nombre de fois où f a une chance d'augmenter pendant les k premières étapes est au moins $\lfloor \frac{k}{3} \rfloor = \ell$. Nous verrons donc ces possibilités d'incrémentation comme une distribution de Bernouilli qui prend la valeur 1 avec probabilité $\frac{3}{16}$.

Appellons C_k l'événement *"l'algorithme a convergé en k actions"*. Soit F_k la variable aléatoire du nombre de fois où la fonction de potentiel f a augmenté durant les k premiers pas de l'exécution. Si l'algorithme n'a pas convergé en k pas, alors la fonction de f a été incrémentée moins de $\frac{n}{2}$ fois.

$$Pr[\neg C_k] \quad \leq \quad Pr\left(F_k < \frac{n}{2}\right)$$

Donc parmis les l premières activations de célibataires, au plus $\frac{n}{2} - 1$ ont réussi :

$$Pr[\neg C_k] \quad \leq \quad \sum_{i=0}^{\frac{n}{2}-1} \binom{\ell}{i} \left(\frac{3}{16}\right)^i \left(\frac{13}{16}\right)^{\ell - i} \text{ avec } \ell = \left\lfloor \frac{k}{3} \right\rfloor \tag{8.12}$$

L'équation (8.12) peut être réécrite comme suit :

$$Pr[\neg C_k] \quad \leq \quad Pr\left(S_\ell < \frac{n}{2}\right) \tag{8.13}$$

Et nous appliquons donc l'inégalité d'Hoeffding avec $\varepsilon_0 = \left(\frac{3}{16} - \frac{n-2}{2\ell}\right)$:

$$Pr\left(S_\ell \leq \frac{n}{2} - 1\right) \quad \leq \quad e^{-2\left(\frac{3}{16} - \frac{n-2}{2\ell}\right)^2 \ell} \tag{8.14}$$

On pose $A = 2\left(\frac{3}{16} - \frac{n-2}{2\ell}\right)^2 \ell$, ce qui nous permet de réécrire les équations (8.13) et (8.14) :

$$Pr[\neg C_k] \leq e^{-A} \tag{8.15}$$

Par suite, une condition suffisante pour que l'algorithme ait convergé en k pas avec probabilité plus grande que $1 - \varepsilon$ est que :

$$Pr[C_k] \geq 1 - \varepsilon \;\Leftrightarrow\; Pr[\neg C_k] \leq \varepsilon$$

Nous avons :

$$e^{-A} \leq \varepsilon \;\Leftrightarrow\; A \geq -ln\,\varepsilon \tag{8.16}$$

D'après les équations (8.15) et (8.16), on a :

$$A \geq -ln\,\varepsilon \Rightarrow Pr[\neg C_k] \leq \varepsilon \tag{8.17}$$

Il faut choisir k tel que $A \geq -ln\,\varepsilon$.
Nous posons $k = \alpha\frac{3n}{2}$ avec $\alpha \geq \frac{32}{3}$ et donc $\ell = \frac{\alpha n}{2}$. Nous obtenons :

$$A = \alpha n\left(\frac{3}{16} - \frac{n-2}{\alpha n}\right)^2$$

Comme $\alpha \geq \frac{32}{3}$, nous avons :

$$0 \leq \frac{n-2}{\alpha n} < \frac{1}{\alpha} \leq \frac{1}{2}\cdot\frac{3}{16}$$

et donc :

$$\left(\frac{1}{2}\cdot\frac{3}{16}\right)^2 \leq \left(\frac{3}{16} - \frac{1}{\alpha}\right)^2 < \left(\frac{3}{16} - \frac{n-2}{\alpha n}\right)^2.$$

Nous obtenons donc :

$$A \geq \alpha n\frac{9}{1024} > \frac{\alpha n}{128}$$

En prenant $\alpha \geq \max\{\frac{32}{3}, \frac{-128ln\,\varepsilon}{n}\}$, nous avons :

$$\frac{\alpha n}{128} \geq -ln\,\varepsilon \Rightarrow A \geq -ln\,\varepsilon$$

D'après l'équation (8.17), nous avons donc :

$$\alpha \geq \max\left\{\frac{32}{3}, \frac{-128ln\,\varepsilon}{n}\right\} \Rightarrow Pr[\neg C_k] \leq \varepsilon$$

Et donc, pour $\alpha \geq \max\{\frac{32}{3}, \frac{-128ln\,\varepsilon}{n}\}$, nous avons la convergence avec probabilité supérieure à $1-\varepsilon$.

Pour conclure, puisque $k = \alpha\frac{3n}{2}$, pour $k \geq \max\{16n, -192\ln\varepsilon\}$, l'algorithme a convergé après k étapes avec probabilité plus grande que $1-\varepsilon$.

Avec probabilité d'au moins $1-\varepsilon$, l'algorithme converge donc en au plus m pas avec $m \geq \max\{16n^2, -192n\ln\varepsilon\}$.

■

Cela signifie par exemple que l'algorithme converge en au plus $16n^2$ pas avec probabilité $1 - e^{-\frac{n}{12}}$.

8.4 Résumé et conclusion

Dans ce chapitre, nous avons présenté notre contribution au problème du couplage maximal distribué. À partir d'un algorithme existant, nous avons contruit un algorithme qui fonctionne sur un graphe anonyme avec un démon distribué tout en gardant une complexité polynomiale. Nous avons utilisé pour cela de l'aléatoire. Nous avons prouvé que notre algorithme calcule bien un couplage maximal. Ensuite nous évaluons le temps nécessaire pour atteindre une configuration stable. Nous avons prouvé dans un premier temps que le temps moyen de convergence est en $O(n^3)$ puis dans un second temps que l'algorithme converge en $O(n^3)$ pas avec forte probabilité. Finalement, nous prouvons que dans un cas particulier — le graphe de communication est complet et le démon est synchrone — l'algorithme converge en $O(n^2)$ pas avec forte probabilité. Ce dernier résultat nous laisse penser que notre borne sur le temps de convergence avec forte probabilité dans le cas général peut être abaissée à $O(n^2)$ pas.

Chapitre 9

Complexité d'un algorithme calculant une $\frac{2}{3}$-approximation du couplage maximum

Dans ce chapitre, nous présentons une borne inférieure sur la complexité d'un algorithme présenté par Manne *et al.* en 2008 [MMPT08]. Il a été démontré que cet algorithme calcule une $\frac{2}{3}$-approximation d'un couplage maximum et que ses exécutions avaient une longueur inférieure à $2^{O(n)}$ pas. Cependant est restée ouverte la question de savoir si cette borne sur la complexité est optimale, c'est-à-dire de savoir s'il existe des cas où l'algorithme mettra nécessairement au moins 2^n (à une constante multiplicative près) pas avant de se stabiliser. Dans ce chapitre, nous ne prouvons pas que cette borne est optimale mais nous obtenons une borne inférieure sous-exponentielle en exhibant une exécution de longueur $2^{\Omega(\sqrt{n})}$. En particulier, nous prouvons ainsi que cet algorithme n'est pas polynomial.

L'exécution que nous allons décrire correspond à la simulation d'un compteur allant de 0 à $2^N - 1$, pour un entier N donné. Cette exécution aura lieu sur un graphe noté G_N qui possède $\Theta(N^2)$ sommets. Le graphe G_N est construit à partir de N sous-graphes ; chacun de ces sous-graphes représente un bit. Le graphe entier représente alors un entier codé sur N bits. La définition 44 définira l'encodage d'un bit sur un de ces sous-graphes.

9.1 Famille de graphes $(G_N)_{N\in\mathbb{N}}$

Avant de montrer la construction générale d'un graphe G_N, nous commençons par l'exemple de $N = 4$. Cet exemple nous servira de fil conducteur et sera utilisé pour illustrer chaque étape de notre construction.

Le graphe G_4 est montré dans la figure 9.1. Les sommets représentés par des cercles sont utilisés pour stocker des bits, et donc un entier à eux tous. Dans cet exemple, l'entier stocké

est 3. Les sommets représentés par des carrés ont un usage technique et ils permettent de contrôler la valeur des bits. Leur usage précis est détaillé plus en avant. Les arêtes en gras représentent le couplage maximal précédemment calculé qui servira de base au calcul du couplage consistuant une $\frac{2}{3}$-approximation.

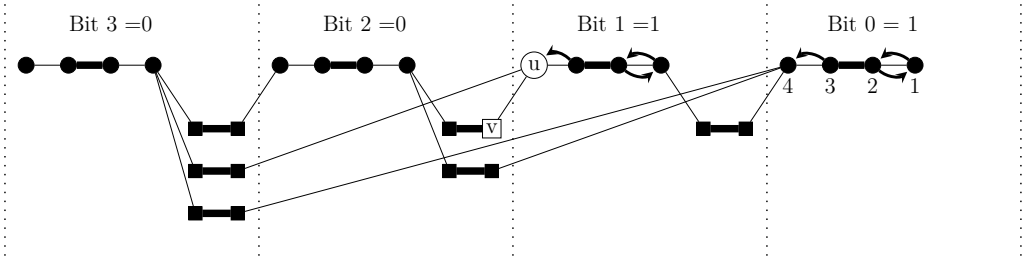


FIGURE 9.1 – Le graphe G_4 : les flèches représentent les variables locales p ; le graphe représente l'entier $3 = \overline{0011}^2$ (en utilisant la notation binaire) ; $u = B_4^1$; $v = R_2^{(2,1)}$; les arêtes épaisses sont celles qui appartiennent à M

Nous décrivons maintenant le graphe général $G_N = (V_N, E_N)$, pour $N \geq 1$, qui est formellement défini par :

1. $V_N = V_N^{\bullet} \cup V_N^{\blacksquare}$ où

$$\begin{aligned} V_N^{\bullet} &= \bigcup_{0 \leq i < N} \left\{ B_k^i \;\middle|\; k = 1,2,3,4 \right\} \\ V_N^{\blacksquare} &= \bigcup_{0 \leq j < i < N} \{R_1^{(i,j)}, R_2^{(i,j)}\} \end{aligned}$$

2. $E_N = E_N^{\bullet} \cup E_N^{\blacksquare}$ où

$$\begin{aligned} E_N^{\bullet} &= \bigcup_{0 \leq i < N} \left\{ (B_k^i, B_{k+1}^i) \;\middle|\; k = 1,2,3 \right\} \\ E_N^{\blacksquare} &= \bigcup_{0 \leq j < i < N} \{(B_4^j, R_2^{(i,j)}), (R_2^{(i,j)}, R_1^{(i,j)}), (R_1^{(i,j)}, B_1^i)\} \end{aligned}$$

Il existe une règle (détaillée plus loin) permettant de déterminer la valeur du bit i en fonction des variables locales des nœuds du sous-graphe correspondant.

Nous distinguons deux familles de sommets : ceux appartenant à $V_N^{\bullet}$ et ceux appartenant à $V_N^{\blacksquare}$. Pour chaque bit i, il y a quatre sommets de $V_N^{\bullet}$ qui y sont associés. Nous avons nommé ces sommets B_k^i, b pour "bit", i indique quel bit et k quel sommet dans ce bit. Par exemple, dans la figure 9.1, le sommet u est le quatrième sommet du bit 1, *i.e.* le sommet u est nommé B_4^1.

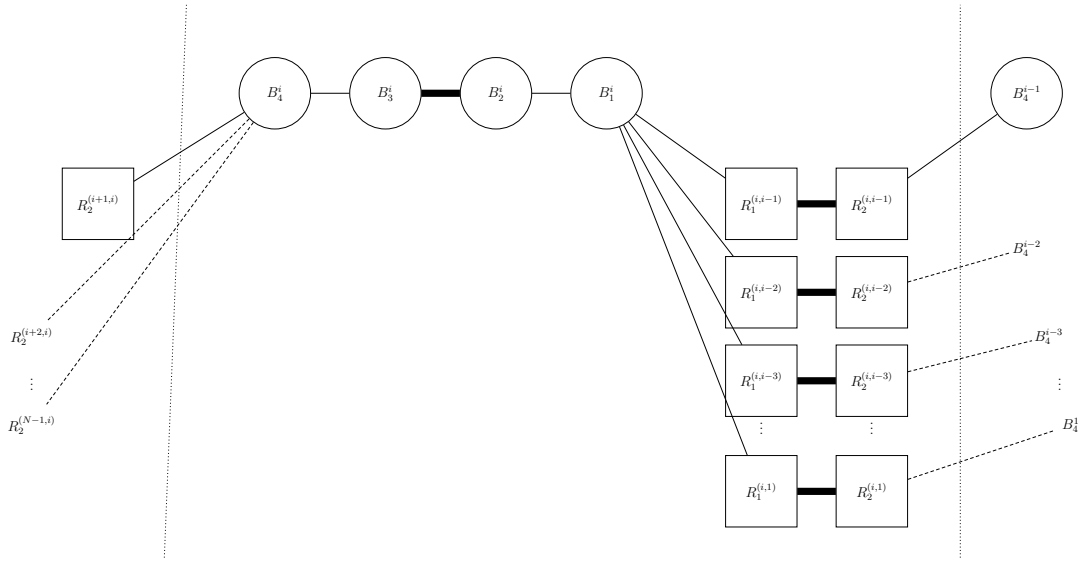


FIGURE 9.2 – Un bloc

Pour tous les bits i et j, avec $i > j$, nous introduisons une paire de sommets appartenant à $V_N^{\blacksquare}$. Ils servent de raccourcis pour que le bit de poids plus élevé i puisse communiquer avec le bit de poids plus faible j. En les utilisant, lorsque le bit i vaut 1, le bit j peut être remis à 0. Dans la suite, nous nous référerons à ces sommets en tant que *sommets reset*. Ces sommets seront nommés $R_1^{(i,j)}$ et $R_2^{(i,j)}$ (en suivant le chemin du bit de poids plus élevé au bit bit de poids plus faible) où r signifie "reset", i présente le bit de poids élevé et j le bit de poids faible. Par exemple, dans la figure 9.1, le sommet v est le second sommet sur le raccourci allant du bit 2 au bit 1, par suite le sommet v est nommé $R_2^{(2,1)}$.

Les identifiants des sommets sont choisis de manière à respecter les conditions suivantes :

$$B_k^i > B_{k'}^{i'} \text{ si } i < i' \text{ ou } i = i' \wedge k < k'$$

$$B_2^i > R_1^{(i,j)}$$

$$B_3^i < R_2^{(i,j)}$$

9.2 Description de l'éxécution

Notre exécution se base sur le couplage maximal M calculé par un algorithme de couplage maximal $\mathcal{M}$:

$$M = \left\{ (B_2^i, B_3^i) \;\middle|\; 0 \leq i < N \right\} \cup \left\{ (R_1^{(i,j)}, R_2^{(i,j)}) \;\middle|\; 0 \leq j < i < N \right\}$$

Cela signifie que chaque nœud x possède une variable m_x telle que : $m(B_2^i) = B_3^i, m(B_3^i) = B_2^i, m(R_1^{(i,j)}) = R_2^{(i,j)}$ and $m(R_2^{(i,j)}) = R_1^{(i,j)}$ Ce couplage est une $\frac{1}{2}$-approximation et l'algorithme va l'augmenter.

9.2.1 Exemple sur G_4

Nous décrivons maintenant une partie de l'exécution, en débutant avec la configuration représentée sur la figure 9.1. Le démon active les nœuds $R_2^{(2,1)}$ et $R_2^{(2,0)}$. Cela est possible puisque tous les nœuds célibataires du chemin 3-augmentant correspondant ont leur pointeur p initialisé à $\perp$. Ensuite le démon active les nœuds célibataires B_4^1 et B_4^0. Ces derniers choisissent respectivement $R_2^{(2,1)}$ et $R_2^{(2,0)}$. Puis le démon peut activer, et active, $R_1^{(2,1)}$ et $R_1^{(2,0)}$; ces derniers acceptent les propositions qui leur sont faites.

Les nœuds B_3^1 et B_3^0 doivent alors annuler leur proposition (puisque leurs élus ont choisi une autre proposition). Et cela conduit également à l'annulation des choix de B_2^1, B_2^0, B_1^1 et B_1^0. La configuration du graphe résultante est montrée dans la figure 9.5.

L'exécution continue avec les activations de B_2^3, B_1^3 et B_3^3. Elles sont possibles car les nœuds célibataires B_1^3 et B_4^3 n'ont pas encore fait de choix. Et ainsi le nœud B_1^3 a fait un choix (le nœud B_2^3), le démon peut alors activer les nœuds reset du second bit. Le graphe se retrouve finalement dans la configuration montrée dans la figure 9.7.

Grâce à ce morceau d'exécution, les bits ont changé de 0011 à 0100. L'exécution peut ensuite continuer et les configuration du graphe représenteront successivement $\overline{0101}^2$, puis $\overline{0110}^2$,$\overline{0111}^2$,$\overline{1000}^2$, et ainsi de suite.

En partant d'une configuration représentant $\overline{0000}^2$, nous pouvons obtenir de cette manière une exécution qui passe par toutes les configurations représentant les entiers entre $0 = \overline{0000}^2$ et $15 = \overline{1111}^2$.

Nous allons maintenant généraliser cette approche pour tous les graphe G_N.

9.2.2 Généralisation

Nous définissons et utiliserons la notion d'arêtes *On* ou *Off*. Puisque nous ne travaillerons que sur des graphes G_N durant tout le reste de cette section, nous pouvons supposer que tout nœud participant au couplage maximal a exactement un unique voisin célibataire.

Définition 43 (On/Off) *Soit $e = (u, v)$ une arête appartenant au couplage maximal M. Soit x (resp. y) le nœud célibataire voisin de u (resp. v). Sans perte de généralité, nous pouvons supposer que le nœud x a un identifiant plus petit que y.*

L'arête e est dite Off *si $p_u = \perp$, $p_v = \perp$, $p_x = \perp$ et $p_y = \perp$.*

De plus, l'arête e est dite On *si $p_u = x, p_x = u$ et $p_v = y$.*

La figure 9.3 illustre de la définition 43.

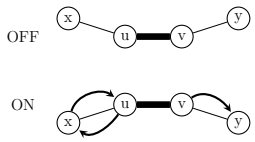


FIGURE 9.3 – Des arêtes On et Off ; les flèches représentent les variables locales p.

Lemme 9.2.1 (Turn On/Off) *Soit (u, v) une arête du couplage M. Soit x (resp. y) le voisin célibataire de u (resp. v). On suppose que $x < y$ et que $\nexists z < u.p_z = x$.*

Turn On : *Si l'arête (u, v) est Off, que $p_x = p_y = \perp$ et que $\alpha_v = \beta_u = \beta_v = \perp$, alors il existe une exécution finie telle que dans la configuration finale de cette exécution l'arête (u, v) est On, $\alpha_u = x$ et $\alpha_v = y$.*

Turn Off : *si l'arête (u, v) est On, que $p_y \notin \{v, \perp\}$, que $\alpha_u = x$ et que $\alpha_v = y$, alors il existe une exécution finie telle que dans la configuration finale de cette exécution l'arête (u, v) est Off et $\alpha_v = \beta_u = \beta_v = \perp$.*

Preuve.

Turn On : Supposons que l'arête (u, v) est Off, que $\alpha_v = \beta_u = \beta_v = \perp$ et que $p_x = p_y = \perp$. Le démon peut alors activer successivement les nœuds u, x et v.

1. Premièrement, la règle *Update* est appliquée par le nœud v. Sa garde est vérifiée puisque $BestRematch(v) = (y, \perp)$. Après application de la règle, nous avons $(\alpha_v, \beta_v) = (y, \perp)$ et $s_v = false$.
2. Deuxièmement, nous avons $AskFirst(u, v) = x \neq p_u$ et alors le nœud u vérifie la garde de la règle $MatchFirst$. En appliquant cette règle, nous obtenons $p_u = x$.
3. Troisièmement, nous avons $u = Lowest\{w \in N(x) \mid p_w = x\}$ et $p_x \neq u$. Par suite, le nœud x vérifie la garde de la règle *Single*, cela a pour effet de changer la valeur de p_x à u.
4. Quatrièmement, le nœud u vérifie la garde de la règle $MatchFirst$ puisque $false = s_u \neq (p_{p_u} = u) = true$. Après exécution de cette règle, nous avons $s_u = true$.
5. Cinquièmement, le nœud v vérifie la garde de la règle $MatchSecond$, en effet nous avons $AskSecond(v, u) = \beta_v \neq p_v$ et $s_u = true$. après application de cette règle, nous obtenons $p_v = y$.

À ce moment-là, l'arête (u, v) est en position On, $\alpha_u = x$ et $\alpha_v = y$. Cette séquence d'actions définit la sous-exécution qui sera notée TurnOn(u, v).

Turn Off : Supposons que l'arête (u, v) est On et que $p_y \notin \{v, \perp\}$. Le démon peut alors activer successivement les nœuds v, u et x.

1. Premièrement, la règle *update* est appliquée par le nœud v. La garde est en effet vérifiée par v : $BestRematch(v) = (\perp, \perp)$. Après application de la règle, nous avons $(\alpha_v, \beta_v) = (\perp, \perp)$, $p_v = \perp$ et $s_v = false$.
2. Deuxièmement, le nœud u vérifie la garde de la règle $ResetMatch$ puisqu'à présent $AskFirst(u, v) = AskSecond(u, v) = \perp$. En appliquant cette règle, nous obtenons alors $p_u := \perp$.
3. Troisièmement, nous avons $u \notin \{w \in N(x) \mid p_w = x\}$ et $p_x = u$. Par suite le nœud x vérifie la garde de la règle $Single$. Cela change la valeur du pointeur p_x sur $\perp$.

À ce moment-là, l'arête (u, v) est Off, $\alpha_v = \beta_u = \beta_v = \perp$ et $p_y = p_x = \perp$. Cette séquence d'actions définit la sous-exécution qui sera notée TurnOff(u, v).

■

Nous avons maintenant les outils nécessaires pour la description complète d'une exécution d'au moins 2^N pas.

Définition 44 (Bit associé à un bloc) *Soit $i < N$.*
Un bloc *de quatre sommets $\{B_1^i, B_2^i, B_3^i, B_4^i\}$ encode un bit. Sa valeur est* 1 *(resp* 0*) si l'arête (B_2^i, B_3^i) est On (resp. Off).*

Nous attirons l'attention sur le fait que la valeur du bit peut ne pas être bien définie, dans ce cas-là nous dirons que la valeur est non-définie.

Nous décrivons maintenant deux suites de pas qui commencent et finissent dans des configurations vérifiant les deux conditions décrites ci-après.

Raccourcis Off : Toutes les arêtes $(R_1^{(i,j)}, R_2^{(i,j)})$ sont Off, $\alpha_{R_2^{(i,j)}} = \perp$ et $\beta_{R_1^{(i,j)}} = \beta_{R_2^{(i,j)}} = \perp$

Consistence des Variables : Pour tout i, l'arête (B_2^i, B_3^i) est ou bien On, ou bien Off mais pas non-définie et $\alpha_{B_2^i} = B_1^i$, $\alpha_{B_3^i} = p_{B_3^i}$ et $\beta_{B_2^i} = \beta_{B_3^i} = \perp$.

Si une configuration d'un graphe G_N vérifie ces deux conditions, alors il est possible de lui associer un entier ω. Nous appellerons de telles configurations des ω-configurations. La figure 9.1 montre le graphe G_4 dans une 3-configuration, par exemple.

Le i^{eme} bit de l'entier ω correspond à la valeur codée par le i^{eme} bloc de sommets. Comme l'entier ω est écrit sur N bits, nous avons $0 \leq \omega < 2^N$.

Nous rappelons certains faits sur le graphe G_N qui seront beaucoup utilisés dans la construction de l'exécution.

— Tout sommet u qui appartient à une arête du couplage M a exactement un voisin célibataire.

— Nous pouvons donc en déduire que pour un tel sommet u, à tout instant nous avons $\beta_u = \perp$

Nous allons maintenant décrire une suite de pas qui transforme une ω-configuration en une $(\omega+1)$-configuration. Nous distinguons deux cas selon la valeur du bit de poids le plus faible.

Nous supposons dans un premier temps que le bit de poids le plus faible de ω est 0. Nous savons que $p_{B_k^0} = \perp$ pour $k = 1, 2, 3, 4$ et $\alpha_{B_3^0} = \perp$, il est donc possible d'appliquer le lemme 9.2.1 et donc le démon peut exécuter la suite de pas $\text{TurnOn}(B_2^0, B_3^0)$.

Après cette suite de pas, la configuration vérifie toujours les deux conditions énoncées plus haut, le seul bloc ayant été modifié est le bloc codant le bit de poids le plus faible et sa valeur a été changée de 0 en 1. Nous sommes donc dans une $(\omega + 1)$-configuration.

Dans un second temps, nous supposons que les $i - 1$ bits de poids le plus faible de ω sont 1 et que le i^{eme} bit est 0, pour $N \geq i \geq 2$. Dans ce cas-là, nous allons utiliser les raccourcis du bit i aux bits j pour tout $j < i$. Les arêtes $(R_1^{(i,j)}, R_2^{(i,j)})$ passent dans l'état On, ce qui permet de changer la valeur des bits j. Ensuite, la valeur du bit i peut changer. Finalement, les arêtes $(R_1^{(i,j)}, R_2^{(i,j)})$ peuvent repasser dans l'état Off. En faisant cela, le graphe est à nouveau dans une configuration qui respecte les conditions et l'entier encodé sur la configuration a été incrémenté de 1.

Nous allons utiliser le graphe de la figure 9.1 comme exemple pour bien suivre les différentes étapes de cette suite de pas. Et de manière plus précise, le démon exécute la suite de pas suivante :

1. Les arêtes $(R_1^{(i,j)}, R_2^{(i,j)})$ passent dans l'état On pour $0 \leq j < i$:
   $\text{TurnOn}(R_1^{(i,0)}, R_2^{(i,0)}); \ldots ; \text{TurnOn}(R_1^{(i,j)}, R_2^{(i,j)}); \ldots ;$
   $\text{TurnOn}(R_1^{(i,i-1)}, R_2^{(i,i-1)})$
   Le lemme 9.2.1 peut effectivement s'appliquer ici puisque $R_1^{(i,j)}, R_2^{(i,j)}, B_4^j$ et B_1^i pointent vers $\perp$ pour tout $0 \leq j < i$, et $\alpha_v = \perp$ pour $v \in \left\{ R_1^{(i,j)} \;\middle|\; 0 \leq j < i \right\}$.
   Le graphe de la figure 9.4 correspond au graphe de la figure 9.1 après cette étape.
2. Les bits $0 \leq j < i$ prennent la valeur 0 :
   $\text{TurnOff}(B_2^0, B_3^0); \ldots ; \text{TurnOff}(B_2^j, B_3^j); \ldots ;$
   $\text{TurnOff}(B_2^{i-1}, B_3^{i-1})$
   Cette suite de pas est possible puisque toutes les arêtes considérées sont On, que pour tout $0 \leq j < i$ nous avons $\alpha_{B_2^j} = B_1^j$, $\alpha_{B_3^j} = B_4^j$ et que $p_{B_4^j} = R_2^{(i,j)}$.
   Le graphe de la figure 9.5 correspond au graphe de la figure 9.1 après cette étape.
3. Le i^{eme} bit prend la valeur 1 :
   $\text{TurnOn}(B_2^i, B_3^i)$
   Nous avons $p_{B_k^i} = \perp$ pour $k = 1, 2, 3, 4$ et donc nous pouvons appliquer le lemme 9.2.1.
   Le graphe de la figure 9.6 correspond au graphe de la figure 9.1 après cette étape.

4. Les arêtes des raccourcis qui appartiennent à M passent dans l'état Off afin que la condition "Raccourcis Off" soit respectée :
   $\text{TurnOff}(R_1^{(i,0)}, R_2^{(i,0)}); \text{TurnOff}(R_1^{(i,1)}, R_2^{(i,1)}); \ldots;$
   $\text{TurnOff}(R_1^{(i,i-1)}, R_2^{(i,i-1)})$
   Cette suite de pas est possible parce que toutes les arêtes considérées sont On, que $\alpha_{R_1^{(i,j)}} = B_4^j$, $\alpha_{R_2^{(i,j)}} = B_1^i$ et que $pB_1^i = B_2^i$.
   Le graphe de la figure 9.7 correspond au graphe de la figure 9.1 après cette étape.

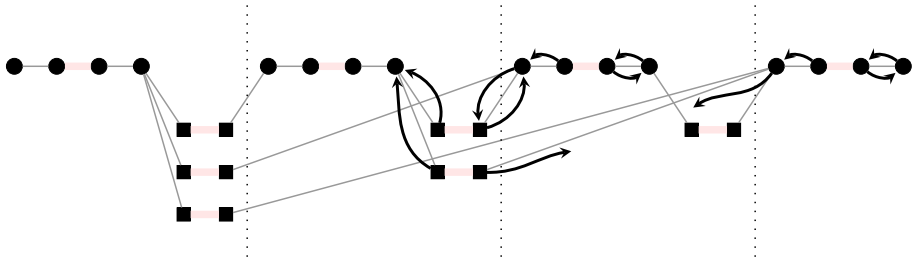


FIGURE 9.4 – Le même graphe G_4 que celui de la figure 9.1 après l'étape 1

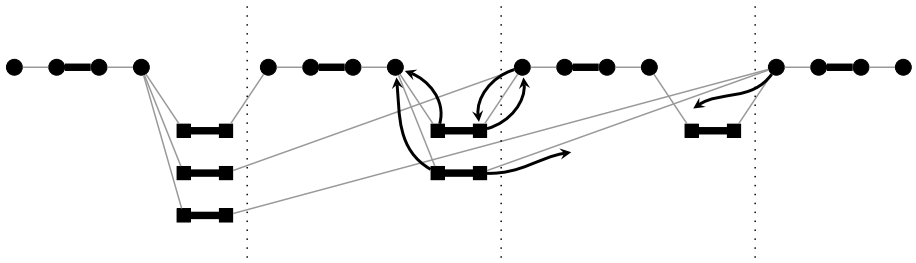


FIGURE 9.5 – Le même graphe G_4 que celui de la figure 9.1 après l'étape 2

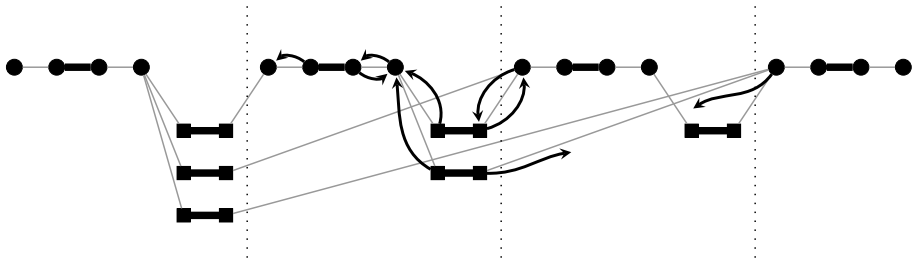


FIGURE 9.6 – Le même graphe G_4 que celui de la figure 9.1 après l'étape 3

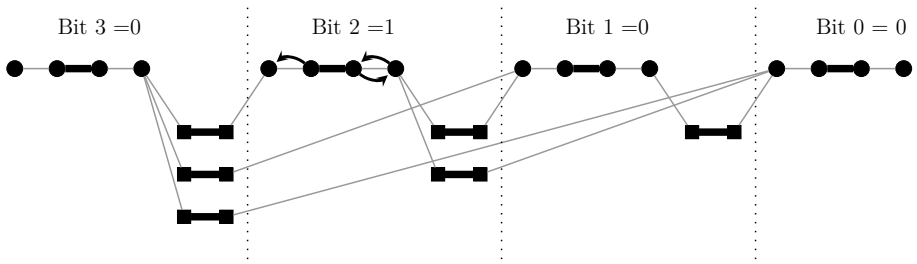


FIGURE 9.7 – Le même graphe G_4 que celui de la figure 9.1 après l'étape 4

À la fin de cette séquence de pas, la configuration vérifie les deux conditions et les i premiers bits de ω ont changé de valeur. Nous sommes donc passés d'une ω-configuration à une $\omega + 1$-configuration.

Dans tous les cas, nous obtenons une séquence de pas qui effectue l'opération

$$\omega := \omega + 1.$$

Nous définissons la configuration initiale de notre exécution C_0 en fixant, pour tout nœud u, la valeur des variables locales p_u à $\perp$, et β_u à $\perp$, et α_u à B_1^i si $u = B_2^i$ et $\perp$ sinon. La configuration C_0 est donc par définition une 0-configuration. En alternant les deux suites de pas décrites plus haut en fonction du dernier bit de l'entier encodé dans la configuration, l'exécution passe par une 1-configuration, une 2-configuration,. . .,une (2^N-1)-configuration. Chacune de ces suites de pas compte au moins 5 pas. Cette exécution comporte donc au moins $5 \cdot 2^N$ pas.

Si n est le nombre de sommets de G_N, alors, par construction du graphe, nous avons $n = 4N + 2\frac{N(N-1)}{2} = N^2 + 3N$. Nous avons donc :

$$n = \left(N + \frac{3}{2}\right)^2 - \frac{9}{4}$$

Et donc :

$$\sqrt{n} \leq N + \frac{3}{2}$$

Et ainsi :

$$2^{\sqrt{n}-\frac{3}{2}} \leq 2^N$$

Par suite, l'exécution comporte au moins $\frac{5}{2\sqrt{2}} \cdot 2^{\sqrt{n}}$ pas. L'algorithme a donc une complexité en $\Omega\left(2^{\sqrt{n}}\right)$ pas.

9.3 Résumé et conclusion

Dans ce chapitre, nous avons étudié la complexité d'un algorithme autostabilisant calculant une $\frac{2}{3}$-approximation d'un couplage maximum publié par Manne *et al.* en 2008.

Nous avons montré en détail la construction d'une exécution de taille $\Omega\left(2^{\sqrt{n}}\right)$ pas. Cette construction ne suffit pas à montrer que la borne supérieure sur la complexité de cet algorithme calculée par Manne *et al.* est optimale, cependant elle montre que l'agorithme en question n'est pas polynomial.

À l'aide de méthode probabiliste, nous avons commencé à développer une variante de cette algorithme dans le but d'obtenir une complexité polynomiale. Cet algorithme est composé de différentes couches qui exécutent chacune les mêmes algorithmes afin d'augmenter le couplage de la couche inférieure. Nous n'avons pas encore pu montrer de borne raisonnable sur la complexité de notre algorithme. Il est probable que son écriture évoluera encore et sa publication viendra alors compléter ce chapitre.

Chapitre 10

Conclusion et perspectives

Nous présentons dans cette thèse deux aspects de la notion de calcul distribué. Dans une première partie, nous développons une étude portant sur les protocoles de population. Cette partie traite de modèles de calcul distribué. Dans une seconde partie, nous nous sommes intéressés à des algorithmes distribués autostabilisants qui résolvent le problème du couplage dans un graphe.

Résumé

Le chapitre 2 est un chapitre bibliographique consacré aux protocoles de population. Ils ont été introduits en 2004 par Angluin *et al.* [AAD$^+$04]. Nous présentons le modèle original accompagné des principaux résultats à son sujet. Les langages reconnus par ce modèle correspondent exactement aux ensembles semi-linéaires. Nous présentons également quelques variantes présentées dans la littérature. La plupart de ces variantes ont été comparées à différentes classes de machines de Turing probabilistes ou non-déterministes. Pour d'autres variantes, portant sur le modèle de communication utilisée, nous donnons l'ensemble des prédicats reconnus. Nous relions également le modèle des protocoles de population à d'autres modèles de calcul moins courants que les machines de Turing, comme les réseaux stochastiques de réactions chimiques ou les systèmes d'addition de vecteurs.

Dans le chapitre 3, nous complétons ce premier chapitre bibliographique en rapportant la construction d'une machine de Turing sur la variante particulière des protocoles de population où le graphe d'interaction est réduit et possède un degré borné par une constante. Cette construction n'étant pas complètement satisfaisante, nous proposons une approche un peu différente. Notre approche s'appuie sur un autre formalisme que celui des protocoles de population. Ce nouveau formalisme permet une plus grand modularité. Il permet également de définir formellement des protocoles couramment utilisé, comme celui de l'élection d'un leader au sein de la population.

Dans le chapitre 4, le formalisme défini au chapitre précédent est réutilisé pour construire une hiérarchie entre le modèle des protocoles de population classiques — qui ne peuvent reconnaître que des langages semi-linéaires — et ceux travaillant sur un graphe à degré borné — qui ont une puissance de calcul équivalente aux machines de Turing non-déterministes utilisant un espace linéaire. Notre construction s'appuie sur deux constructions existantes de hiérarchies, et elle peut être vue comme une généralisation de chacune des deux. Cependant notre hiérarchie distingue plus de cas et notre construction permet de caractériser exactement la puissance des modèles dans des cas où les résultats précédents ne pouvaient que donner un encadrement.

Dans les chapitres 5 et 6, nous étudions des variantes du modèle des protocoles de population. Les résultats présentés dans ces chapitres sont tous des contributions originales.

Dans le chapitre 5, nous considérons des variantes où la forme des règles et l'acceptation sont modifiées. En s'inspirant sur des résultats d'Angluin *et al.*, qui ont montré que lire la sortie en consensus ou en leader ne change pas la puissance du modèle avec transitions classiques, nous avons regardé l'influence de ce changement quand par exemple un agent était détruit à chaque interaction. Nous montrons que le modèle est affaibli si, à chaque interaction, un agent est détruit. De plus, dans ces variantes dites temps réel, l'ensemble des prédicats reconnus dépend fortement de la façon dont est lue la sortie. Nous montrons en plus que, dans le cas où les transitions peuvent générer des agents, le modèle restait équivalent au modèle classique.

Dans le chapitre 6, nous nous focalisons sur une variante particulière du modèle classique des protocoles de population. Dans cette variante, les agents ont une opinion, et cette opinion est fonction de l'état de l'agent. En toute généralité, la sortie d'un agent dépend alors de l'opinion de l'agent, et plusieurs opinions peuvent correspondent à la même sortie. Notre étude ne porte cependant que sur le cas où l'opinion d'un agent est la sortie qui est associée à son état. La relation de transition est modifiée de manière à ce que les agents ne changent d'état — ou d'opinion selon la variante précisément considérée — qu'à la condition qu'ils interagissent entre agents ayant des opinions différentes. Les prédicats reconnus par chaque variation de ce modèle sont exactement les combinaisons booléennes de prédicats de seuil nul. Ces prédicats peuvent être réinterprétés de façon géométrique. Ils correspondent à des cônes dans l'espace $\mathbb{N}^d$ dont les faces, qui sont des cônes de dimension inférieure, sont incluses en partie, ou non, suivant un schéma récursif.

Dans une seconde partie, nous nous intéressons à des algorithmes calculant des approximations de couplages maximums. Dans le chapitre 7, nous posons les définitions, les outils et les concepts que nous utilisons dans la suite. Nous introduisons le problème du couplage et rappelons les principaux résultats. Nous définissons aussi les modèles d'algorithmique distribuée et le notions d'autostabilisation que nous utilisons dans la suite. Dans la fin du chapitre, nous présentons deux algorithmes calculant respectivement une $\frac{1}{2}$-approximation et une $\frac{2}{3}$-approximation d'un couplage maximum. Notre travail se base sur ces deux algorithmes. Nous avons essayé de les améliorer, en changeant les hypothèses sur le modèle ou en améliorant la complexité.

Le premier de ces deux algorithmes calcule un couplage maximal, c'est-à-dire une $\frac{1}{2}$-approximation d'un couplage maximum. Le chapitre 8 lui est consacré. Nous avons changer un choix déterministe effectué en comparant les identifiants des nœuds en un choix probabiliste. Cette modification permet à notre algorithme de fonctionner sous hypothèse d'anonymat tout en conservant une complexité en $O(n^3)$ où n est le nombre de nœuds. Pour un cas particulier, nous avons pu montrer que cette complexité pouvait être bornée en $O(n^2)$.

Dans le chapitre 9, nous étudions le second algorithme. Une borne supérieure exponentielle sur sa complexité avait été démontrée mais la question de savoir si cette borne est optimale est ouverte. Nous avons construit une exécution de cet algorithme dont la longueur est $O(2^N)$ sur un graphe ayant $O(N^2)$ sommets. Nous n'avons donc pas montré l'optimalité de la borne exponentielle mais cela prouve que cet algorithme n'est pas polynomial.

Perspectives

À propos des protocoles de population

Nous avons répertorié un certain nombre de variantes des protocoles de population dans le chapitre 5. Nous avons fait varié les formes des règles de transitions et le mode d'acceptation. La caractérisation exacte de la puissance de calcul n'a pas été montrée pour beaucoup de variantes. Une description géométrique des ensembles reconnus par chaque variantes pourrait permettre d'obtenir les caractérisations voulues.

Notre taxonomie de variantes serait enrichi par l'ajout des variantes probabilistes. La variante probabiliste du modèle classique est strictement plus puissance que le modèle classique. Je m'attends à des changements importants en terme de puissance de calcul pour les variantes probabilistes des variantes considérées dans le chapitre 5. Par exemple, je conjecture qu'avec des transitions génératrices il est possible de simuler une machine de Turing universelle. La taille du ruban de la machine de Turing simulée est limitée par la taille de la population, or avec des transitions génératrices il est possible de faire grossir la population autant que voulu. Cependant, pour les variantes temps réel, l'effet me semble plus dur à prédire : une exécution d'un protocole temps réel est très courte, et il est donc difficile de contrôler les erreurs produites au cour de l'exécution.

Pour notre étude des protocoles confiants, nous nous sommes limités aux protocoles confiants en leur sortie, c'est à dire où l'opinion d'un agent est la sortie qu'il renvoie par la fonction de sortie. Nous n'avons pas traité le cas général où plusieurs opinion différents peuvent correspondre à une même sortie. La condition de stabilité d'une configuration porte sur les sorties des agents. Ce fait me conduit à conjecturer que les protocoles confiants généraux ont exactement la même caractérisation que les protocoles confiants en leur sortie : ils reconnaissent exactement les combinaisons linéaires de prédicats de seuil nul.

À propos des algorithmes de couplage autostabilisants

Notre algorithme de couplage maximal distribué autostabilisant fonctionne sur un réseau anonyme non pondéré. Nous avons commencé à travaillé sur un algorithme autostabilisant calculant un couplage maximal sur un réseau anonyme et pondéré. Cependant je n'ai pas encore réussi à terminer le calcul de la complexité, le calcul de la complexité du cas non pondéré ne se transpose pas facilement au cas pondéré.

Nous avons commencé à construire une version probabiliste de l'algorithme calculant une $\frac{2}{3}$-approximation d'un couplage maximum étudié au chapitre 9. Notre algorithme, sous sa forme actuelle, est un algorithme qui est composé du succession de couches. La première couche calcule un couplage maximal, c'est-à-dire une $\frac{1}{2}$-approximation. Ensuite chaque couche élimine au moins un chemin augmentant de 3 arêtes, se faisant il augmente la taille du couplage. Nous avons pu prouver que lorsque notre algorithme se stabilise le couplage retourné par la couche la plus haute est une $\frac{2}{3}$-approximation d'un couplage maximum. Cependant, nous n'avons pas encore pu montré une borne supérieure polynomiale pour notre algorithme.

Annexe A

L'algorithme Link-name

A.1 Justification et notations

Nous nous plaçons sur un graphe avec numérotation des ports $G = (V, E, \pi)$ où
- V est l'ensemble des sommets et est de taille n.
- E est l'ensemble des arêtes et est de taille m.
- $\mathcal{P}$ est l'ensemble des ports du graphe.
- $\pi : V \times E \to \mathcal{P}$ est une fonction qui associe à un sommet et une arête le port correspondant.

Les nœuds de calculs sont placés sur, et identifiés aux, les sommets du graphes et les arêtes représentent les liens de communications qui sont branchés sur les nœuds en utilisant les ports. Par exemple, si u est un nœud et $e = (u, v)$ une arête alors le lien de comunication correspondant est branché à u sur le port $a = \pi(u, e)$.

Nous rappellons et définissons les notations suivantes :
- $N(u)$ est l'ensemble des voisins du nœud u.
- $\mathcal{P}(u)$ est l'ensemble des ports d'un nœud u.
- $proc(u, a)$ désigne le nœud— ou processus — avec lequel le nœud u peut communiquer via le port a.
- $port(u, a)$ désigne le port situé à l'autre extrémité du lien de communication branché sur le nœud u au port a.

L'*algorithme Link-Name* $\mathcal{A}_2$ presenté dans cette annexe utilise le modèle à numérotation des ports défini dans le chapitre 7.2.

L'algorithme $\mathcal{A}_1$ étudié au chapitre 8.1 utilise des conditions telles que " $\exists v \in N(u)$, $p_v = u$ " dans les gardes de ses règles afin de spécifier le nœud v qui pointe vers le nœud u. L'algorithme $\mathcal{A}_1$ est supposé fonctionner avec anonymat. Cette hypothèse rend ce type de test *a priori* impossible à effectuer : un nœud ne pas enregistrer l'identifiant d'un autre nœud, et donc un nœud ne peut pas lire son propre identifiant dans les variables de ses voisins. L'algorithme Link-Name est là pour résoudre le problème et rendre ce test possible.

Dans l'algorithme $\mathcal{A}_2$, chaque nœud u possède deux registres par port a (voir la figure A.1) : img_{ua} et $link_{ua}$. Le registre $link_{ua}$ contient le nom que le nœud u donne à son port a. Pour le nœud u, le contenu de $link_{ua}$ sera le nom utilisé par u pour désigner le nœud $proc(u, a)$, c'est-à-dire le nœud situé à l'autre bout du lien de communication branché sur le port a de u. Le registre img_{ua} contient le nom utilisé par le nœud $proc(u, a)$ pour désigner u. Du point de vue du nœud u, le registre $link_{vb}$ s'appelle $link_{proc(u,a)port(u,a)}$ (voir la règle (R_a) plus bas).

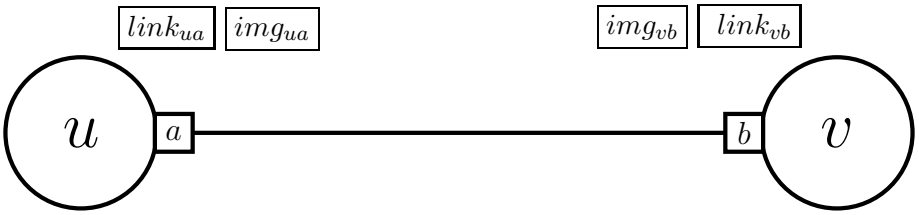


FIGURE A.1 – Une arête entre deux nœuds u et v et les quatres registres associés.

A.2 L'Algorithme Link-Name $\mathcal{A}_2$

A.2.1 Écriture de l'algorithme

L'algorithme Link-Name garantit que, à partir d'un certain moment, deux nœuds voisins u et v se seront mis d'accord sur les noms qu'ils utilisent pour s'appeller l'un l'autre. Plus formellement, nous définissons la spécification $\mathcal{LN}$ pour l'algorithme $\mathcal{A}_2$.

Définition 45 (Spécification $\mathcal{LN}$) *Soit $G = (V, E, \pi)$ un graphe.*
Nous posons $\mathcal{LN} = \mathcal{LN}_1 \wedge \mathcal{LN}_2$ avec :
$\mathcal{LN}_1$ *:* $\forall u \in V, \forall i \in \{1, \ldots, |\mathcal{P}(u)|\}, \exists a \in \mathcal{P}(u),\ link_{ua} = i$
$\mathcal{LN}_2$ *:* $\forall e = (u, v) \in E :\ (\pi(u, e) = a) \wedge (\pi(v, e) = b) \Rightarrow (img_{ua} = link_{vb}) \wedge (img_{vb} = link_{ua})$

La condition $\mathcal{LN}_1$ spécifie qu'un nœud a nommé chacun de ses ports de façon unique, et la condition $\mathcal{LN}_2$ spécifie que les nœuds de part et d'autre d'un lien de communication se sont mis d'accord sur les noms qu'ils emploient pour se nommer l'un l'autre.

Dans l'algorithme $\mathcal{A}_2$, un nœud u vérifie si chacun de ses ports a un nom unique pris entre 1 et $|\mathcal{P}(u)|$. Si ce n'est pas le cas, le nœud u les renomme tous, en utilisant la règle R_0. De plus, pour chaque port a, le nœud u vérifie si le nom utilisé par le nœud $proc(u, a)$ pour le désigner est bien celui contenu dans img_{ua}, au moyen de la règle R_a.

Algorithme 2 *L'algorithme Link-Name $\mathcal{A}_2$ pour le nœud u s'écrit :*

(R_0) $\neg(\forall i \in \{1,\ldots,|\mathcal{P}(u)|\}, \exists a \in \mathcal{P}(u),\ link_{ua} = i)$

$\rightarrow$ Rename all $link_{ua}$ from 1 to $|\mathcal{P}(u)|$

Pour tout $a \in \mathcal{P}(u)$ **:**

(R_a) $img_{ua} \neq link_{proc(u,a)port(u,a)} \rightarrow img_{ua} := link_{proc(u,a)port(u,a)}$

A.2.2 Complexité et correction

Cet algorithme afin d'avoir un intérêt pratique se doit d'avoir une complexité aussi faible que possible, et de fait il a une complexité linéaire en le nombre d'arêtes.

Lemme A.2.1 *Sous le démon distribué, avec l'atomicité lecture/écriture, toute exécution de l'algorithme Link-Name $\mathcal{A}_2$ atteint une configuration stable en au plus $20m$ pas.*

Preuve. Nous commençons par nous concentrer sur la règle R_0 exécutée par le nœud u. Le nœud u ne peut exécuter la règle R_0 au plus une fois. Pendant une exécution de la règle R_0, pour chaque port a, le nœud u effectue au plus deux pas : un pas de lecture pour tester si $link_{ua} = i$ et, éventuellement, un pas d'écriture pour mettre à jour $link_{ua}$. Le nombre de pas exécutés pour la règle R_0 par un nœud du réseau est donc d'au plus :

$$\sum_{u \in V} 2 \cdot |\mathcal{P}(u)| = 4m \ .$$

Ensuite, nous comptons le nombre de pas exécutés pour chaque règle R_a par un nœud u et un port a donnés. La règle R_a est exécutée par le nœud u au plus deux fois, puisque $link_{proc(u,a)port(u,a)}$ change de valeur au plus une fois. Durant chaque exécution de la règle R_a, le nœud u effectue au plus 4 pas : 2 pas de lecture pour comparer les registres img_{ua} et $link_{proc(u,a)port(u,a)}$, et, si la garde est vérifiée, 2 pas d'écritures pour mettre à jour les registres $link_{proc(u,a)port(u,a)}$ to img_{ua}. Par suite, le nombre total de pas effectués lors de l'exécution d'une règle R_a par un nœud du réseau pour tout port a est au plus de :

$$\sum_{u \in V} 2 \cdot 4 \cdot |\mathcal{P}(u)| = 16m \ .$$

En additionnant le nombre de pas effectués par tout les agents pour chaque règle, nous obtenons un nombre de total de pas inférieur, ou égal, à $20m$.

■

Lemme A.2.2 *Pour toute configuration stable de l'algorithme $\mathcal{A}_2$, la spécification $\mathcal{LN}$ est vérifiée.*

Preuve. Dans une configuration stable, la garde des règles R_0 est fausse pour tous les nœuds. Cela signifie que la condition $\mathcal{LN}_1$ est vérifiée.

Pour toute arrête $e = (u, v)$ dans le réseau, ni le nœud u ni le nœud v ne peuvent appliquer leur règle associée à e, c'est-à-dire les règles $R_{\pi(u,e)}$ et $R_{\pi(v,e)}$. Cela implique que $img_{ua} = link_{vb}$ et que $img_{vb} = link_{ua}$, où $a = \pi(u, e)$ et $b = \pi(v, e)$. Par suite, $\mathcal{LN}_2$ est vérifiée.

■

En combinant les lemmes A.2.1 et A.2.2, nous obtenons le théorème suivant.

Théoreme A.2.1 *Sous le démon distribué, avec l'atomicité lecture/écriture, l'algorithme Link-Name $\mathcal{A}_2$ est autostabilisant et silencieux pour la spécification $\mathcal{LN}$, et une configuration stable est atteinte en $O(m)$ pas.*

A.2.3 Réécriture de $\mathcal{A}_1$

Nous donnons ici une manière systématique pour réécrire l'algorithme $\mathcal{A}_1$ en utilisant les registres de $\mathcal{A}_2$ de façon à ce que les instructions de $\mathcal{A}_1$ respectent l'hypothèse d'anonymat. L'algorithme $\mathcal{A}_1$ possèdent deux sortes d'instructions qui, dans leur écriture orignale, violent cette hypothèse : une qui écrit une valeur non-nulle dans une variable p (*e.g.* l'instruction $p_u := v$) et une qui cherche pour une valeur non-nulle précise dans une variable p (*e.g.* le test $p_u \neq v$).

Par exemple, nous souhaitons réécrire le règle *Mariage*

$$(p_u = \perp) \wedge (\exists v \in N(u) : p_v = u) \rightarrow p_u := v$$

de la façon suivante :

$$(p_u = \perp) \wedge (\exists a \in \mathcal{P}(u) : p_{proc(u,a)} = img_{ua}) \rightarrow p_u := link_{ua}.$$

Nous donnons ci-après des règles gnériques qui permettent une telle réécriture. Ces règles génériques sont purement syntaxique, *i.e.* nous remplaçons certains caractères par d'autres. Dans la suite, le nœud u est le nœud exécutant l'algorithme et le nœud v est un autre nœud — *i.e.* $u \neq v$.

- L'ensemble $N(u)$ des identifiants des voisins du nœud u est remplacé par l'ensemble des ports $\mathcal{P}(u)$.
- Si "v" apparait ...
  1. ... dans un quantificateur, alors le nœud u utilise plutôt "a", le port qui le relie au nœud v. C'est-à-dire que l'expression "$\exists v \in N(u)$" est remplacée par "$\exists a \in \mathcal{P}(u)$" et l'expression "$\forall v \in N(u)$" par "$\forall a \in \mathcal{P}(u)$" ;
  2. ... en indice d'une variable (*e.g.* "p_v"), alors "v" désigne le propriétaire de la variable et il est possible de le remplacer par "$proc(u, a)$" ;

3. ... comme contenu d'une variable (*e.g.* "$p_u = v$"), alors "v" désigne le nœud en lui même et il est donc possible de le remplacer par "$link_{ua}$".

En appliquant les règles 2 et 3 ci-dessus, les tests "$p_u = v$" et "$p_v = \perp$" sont respectivement réécrits "$p_u = link_{ua}$" et "$p_{proc(u,a)} = \perp$".

— Si "u" (le nœud qui exécute la règle) apparaît...

1. ... en indice d'une variable (*e.g.* "p_u"), aucune réécriture n'est nécessaire.
2. ... comme contenu d'une variable (*e.g.* "$p_v = u$"), "u" désigne le nœud en lui-même apparraissant dans la variable d'un de ses voisins $v = proc(u, a)$ et il est possible de le remplacer par img_{ua}.

En appliquant ces règles de réécriture, le test "$p_v = u$" est réécrit "$p_{proc(u,a)} = img_{ua}$".

En appliquant ces règles de réécriture à l'algorithme $\mathcal{A}_1$, nous obtenons l'algorithme suivant.

Algorithme 3 ($\mathcal{A}_1^r$) *Le nœud u effectue une action selon l'une de ces règles :*

(Marriage) $(p_u = \perp) \wedge (\exists a \in \mathcal{P}(u)\,,\, p_{proc(u,a)} = img_{ua}) \rightarrow p_u := link_{ua}$

(Abandonment) $(\exists a \in \mathcal{P}(u)\,,\, p_u = link_{ua} \wedge p_{proc(u,a)} \neq img_{ua} \wedge p_{proc(u,a)} \neq \perp) \rightarrow p_u := \perp$

(Seduction) $(p_u = \perp) \wedge (\forall a \in \mathcal{P}(u)\,,\, p_{proc(u,a)} \neq img_{ua}) \wedge (\exists a \in \mathcal{P}(u)\,,\, p_{proc(u,a)} = \perp) \rightarrow$
Si $choose(\{0,1\}) = 1$
alors $p_u := choose(\left\{a \in \mathcal{P}(u) \;\middle|\; p_{proc(u,a)} = \perp\right\})$
sinon $p_u := \perp$

Dans ce qui suit, nous appellerons $\mathcal{A}_1^r$ l'algorithme $\mathcal{A}_1$ réécrit en utilisant les règles énoncées plus haut.

Ayant défini les algorithmes $\mathcal{A}_1^r$ et $\mathcal{A}_2$, nous voudrions les composer afin d'obtenir un algorithme auto-stabilisant. Cependant cela ne peut pas être fait aussi directement. En effet, les deux algorithmes sont définis dans des modèles de communication et des atomicités différents : l'algorithme $\mathcal{A}_1^r$ est écrit dans le modèl à état avec l'atomicité de la règle gardée tandis que l'algorithme $\mathcal{A}_2$ est écrit dans le modèle à numérotation des ports avec l'atomicité lecture/écriture. Pour cette composition, nous conserverons les deux modèles. Ainsi $\mathcal{A}_1^r$ et $\mathcal{A}_2$ seront éxécutés dans la même exécution, mais sous différents modèles.

Il n'est pas possible de directement appliquer le théorème de composition de Dolev *et al.* [DIM93] puisqu'ils supposent que les différents algorithmes utilisent le même modèle de communication. Cependant, il est possible de réutiliser les mêmes arguments :

1. L'algorithme $\mathcal{A}_2$ ne lit pas, ni n'écrit, dans les variables de l'algorithme $\mathcal{A}_1^r$, tandis que l'algorithme $\mathcal{A}_1^r$ lit, et n'écrit pas, dans les registres de l'algorithme $\mathcal{A}_2$.

2. L'algorithme $\mathcal{A}_2$ de stabilise indépendamment de l'algorithme $\mathcal{A}_1^r$.

La correction et la complexité de l'algorithme $\mathcal{A}_1^r$ ont été démontrées sous le modèle à états alors que l'algorithme utilise les registres de l'algorithme $\mathcal{A}_2$. Cependant, l'algorithme $\mathcal{A}_2$ est silencieux, cela signifie que la valeur de ces registres ne changera plus après un certain moment. De plus, un nœud exécutant l'algorithme $\mathcal{A}_1^r$ ne lit pas les registres de ses voisins mais seulement les siens. Ils peuvent donc être vus comme des variables internes. Il s'ensuit que l'algorithme $\mathcal{A}_1^r$ n'utilise que des variables locales et internes et, donc, les preuves pour l'algorithme $\mathcal{A}_1$ sont toujours correctes pour $\mathcal{A}_1^r$.

Par suite, une fois que l'algorithme $\mathcal{A}_2$ est stabilisé, c'est-à-dire une fois que l'algorithme $\mathcal{A}_2$ atteint une configuration stable, l'algorithme $\mathcal{A}_1^r$ finit par se stabiliser parès un certain temps sous le modèle à état avec l'atomicité règle gardée.

Bibliographie

[AAC+05] D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta. Stably computable properties of network graphs. In Viktor K. Prasanna, Sitharama Iyengar, Paul Spirakis, and Matt Welsh, editors, *Distributed Computing in Sensor Systems : First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USE, June/July, 2005, Proceedings*, volume 3560 of *Lecture Notes in Computer Science*, pages 63–74. Springer-Verlag, June 2005.

[AAD+04] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *ACM Symposium on Principles of Distributed Computing*, volume 23, pages 290–299. ACM Press, 2004.

[AAD+06] D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *ACM symposium on Principles of Distributed Computing*, 18(4) :235–253, 2006.

[AAE06a] D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. In *ACM symposium on Principles of Distributed Computing*, pages 61–75. Springer, 2006.

[AAE06b] D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear. In *ACM symposium on Principles of Distributed Computing*, volume 25, pages 292–299, New York, NY, USA, 2006. ACM Press.

[AAER07] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *ACM symposium on Principles of Distributed Computing*, 20(4) :279–304, 2007.

[AR07] J. Aspnes and E. Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93 :98–117, 2007.

[BBBD13] J. Beauquier, P. Blanchard, J. Burman, and S. Delaët. Tight complexity analysis of population protocols with cover times—the zebranet example. *Theoretical Computer Science*, 512 :15–27, 2013.

[BBRR12] J. Beauquier, J. Burman, L. Rosaz, and B. Rozoy. Non-deterministic population protocols. In *Principles of Distributed Systems*, pages 61–75. Springer, 2012.

[BCC+13] O. Bournez, J. Chalopin, J. Cohen, X. Koegler, and M. Rabie. Population protocols that correspond to symmetric games. *International Journal of Unconventional Computing*, 9(1-2) :5–36, 2013.

[BL13a] O. Bournez and J. Lefèvre. Population protocols on graphs : A hierarchy. In *Unconventional Computation and Natural Computation (UCNC)*, pages 31–42. Springer, 2013.

[BL13b] O. Bournez and J. Lefèvre. Population protocols on graphs : A hierarchy. In *Proceedings of the Spatial Computing Workshop (SCW) colocated with the conference on Automonous Agents and MultiAgent System (AAMAS)*, pages 17–22, 2013.

[BLR13] O. Bournez, J. Lefèvre, and M. Rabie. Trustful population protocols. In *International Symposium on DIStributed Computing (DISC)*, pages 447–461, 2013.

[Car35] C. Carathéodory. *Variationsrechnung und partielle Differentialgleichungen erster Ordnung*. Berlin, 1935.

[Car62] A. Carl. Petri. kommunikation mit automaten. *Bonn : Institut für Instrumentelle Mathematik, Schriften des IIM Nr*, 2, 1962.

[CHS02] Subhendu Chattopadhyay, Lisa Higham, and Karen Seyffarth. Dynamic and self-stabilizing distributed matching. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 290–297. ACM, 2002.

[CMN+10] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P.G. Spirakis. Passively mobile communicating logarithmic space machines. *Arxiv preprint arXiv :1004.3395*, 2010.

[CMN+11] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G Spirakis. Passively mobile communicating machines that use restricted space. *Theoretical Computer Science*, 412(46) :6469–6483, 2011.

[Con87] John H Conway. Fractran : A simple universal programming language for arithmetic. In *Open Problems in Communication and Computation*, pages 4–26. Springer, 1987.

[CSWB09] M. Cook, D. Soloveichik, E. Winfree, and J. Bruck. Programmability of chemical reaction networks. In *Algorithmic Bioprocesses*, pages 543–584. Springer, 2009.

[DIM93] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *ACM symposium on Principles of Distributed Computing*, 7(1) :3–16, 1993.

[FJ06] M. Fischer and H. Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *Principles of Distributed Systems*, pages 395–409. Springer, 2006.

[FMR68] P. C. Fischer, A. R. Meyer, and A. L. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2(3) :265–283, 1968.

[Gef03] V. Geffert. Space hierarchy theorem revised. *Theoretical Computer Science*, 295(1) :171–187, 2003.

[GHJS08] Wayne Goddard, Stephen T Hedetniemi, David Pokrass Jacobs, and Pradip K Srimani. Anonymous daemon conversion in self-stabilizing algorithms by randomization in constant space. In *Distributed Computing and Networking*, pages 182–190. Springer, 2008.

[GJ01] Maria Gradinariu and Colette Johnen. Self-stabilizing neighborhood unique naming under unfair scheduler. In *Euro-Par 2001 Parallel Processing*, pages 458–465. Springer, 2001.

[GR09] R. Guerraoui and E. Ruppert. Names trump malice : Tiny mobile agents can tolerate byzantine failures. pages 484–495, 2009.

[GS66] S. Ginsburg and E. H. Spanier. Semigroups, presburger formulas and languages. *Pacific journal of Mathematics*, 16(2) :285–296, 1966.

[HH92] Su-Chu Hsu and Shing-Tsaan Huang. A self-stabilizing algorithm for maximal matching. *Information processing letters*, 43(2) :77–81, 1992.

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(1) :326–336, 1952.

[HJS01] Stephen T Hedetniemi, David P Jacobs, and Pradip K Srimani. Maximal matching stabilizes in time o (m). *Information Processing Letters*, 80(5) :221–223, 2001.

[HK73] J. E. Hopcroft and R. M. Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4) :225–231, 1973.

[HOT11] J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis. Distributed anonymous discrete function computation. *Automatic Control, IEEE Transactions on*, 56(10) :2276–2289, 2011.

[IL94] G. Itkis and L. Levin. Fast and lean self-stabilizing asynchronous protocols. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 226–239. IEEE, 1994.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. In *Structure in Complexity Theory Conference, 1988. Proceedings., Third Annual*, pages 112–115. IEEE, 1988.

[KM69] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2) :147–195, 1969.

[Ler12] J. Leroux. Vector Addition Systems Reachability Problem (A Simpler Solution). In Andrei Voronkov, editor, *The Alan Turing Centenary Conference*,

volume 10 of *EPiC Series*, pages 214–228, Manchester, United Kingdom, June 2012. Andrei Voronkov.

[May81] E. Mayr. Persistence of vector replacement systems is decidable. *Acta Informatica*, 15(3) :309–318, 1981.

[Min67] M. L. Minsky. *Computation : finite and infinite machines*. Prentice-Hall, Inc., 1967.

[MM07] Fredrik Manne and Morten Mjelde. A self-stabilizing weighted matching algorithm. In *Stabilization, Safety, and Security of Distributed Systems*, pages 383–393. Springer, 2007.

[MMPT08] F. Manne, M. Mjelde, L Pilard, and S. Tixeuil. A self-stabilizing $\frac{2}{3}$-approximation algorithm for the maximum matching problem. In *Stabilization, Safety, and Security of Distributed Systems*, pages 94–108. Springer, 2008.

[MMPT09] F. Manne, M. Mjelde, L. Pilard, and S Tixeuil. A new self-stabilizing maximal matching algorithm. *Theoretical Computer Science*, 410(14) :1336–1345, 2009.

[Par61] R. J. Parikh. Language generating devices. *Quarterly Progress Report*, 60 :199–212, 1961.

[Par66] R. J. Parikh. On context-free languages. *Journal of the ACM (JACM)*, 13(4) :570–581, 1966.

[Pil05] L. Pilard. *Observer la stabilisation*. PhD thesis, Université Paris XI, Orsay, 2005.

[Pre30] M. Presburger. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. sprawozdanie z 1 kongresu matematyków krajow slowianskich, ksiaznica atlas. 1930.

[Pre99] R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *Symposium on Theoretical Computer Science (STACS'99)*, volume 1563 of *Lecture Notes in Computer Science*, pages 259–269. Springer, 1999.

[Sak96] M. Saks. Randomization and derandomization in space-bounded computation. In *Computational Complexity, 1996. Proceedings., Eleventh Annual IEEE Conference on*, pages 128–149. IEEE, 1996.

[TH11] Volker Turau and Bernd Hauck. A new analysis of a self-stabilizing maximum weight matching algorithm with approximation ratio 2. *Theoretical Computer Science*, 412(40) :5527–5540, 2011.

[Tur36] A. Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58 :345–363, 1936.

Les protocoles de population sont une modélisation mathématique de systèmes composés d'une multitude d'agents se déplaçant sans contrôle et pouvant communiquer ensemble, deux à deux, quand ils sont assez proches. Dans cette thèse, nous avons déterminé quelles formules peuvent être calculées, et quels problèmes peuvent être résolus, par un tel système en fonction d'une série de contraintes imposées : seules certaines paires d'agents peuvent communiquer, chaque agent peut faire plus ou moins de calcul tout seul, *etc.*. D'un autre côté, nous avons étudié le problème du couplage distribué : comment faire pour qu'un groupe de personnes s'organise de lui même par paire ? Chacun ne peut appartenir qu'à au plus une paire, toutes les paires ne sont pas possibles initialement et on voudrait qu'il y ait le plus de paires possibles formées. Par ailleurs, personne ne connaît la liste totale des paires possibles, ni même le nombre de personnes présentes. Nous avons proposé une solution sous certaines contraintes et étudié le problème sous d'autres contraintes.

Population protocols are a mathematical model of systems composed by a lot of mobile agents. The agents are equipped of sensors and can communicate pairwise when close enough. We determined what sort of formula, or problem, could be solved by those systems under different sets of constraints such as : only some pairs of agents can interact, each agent can do some local computation, *etc.*. On the other side, we studied the distributed matching problem : how can a group of people organize themselves into pairs ? Each person can be in at most one pair, but not all pairs are possible. Furthermore, nobody know neither the list of every possible pairs, nor the size of the group. We proposed a solution under some constraints and studied the problem under some others.