Laboratoire Spécification et Vérification

ENS CACHAN

Mémoire de DEA :

Configurations récurrentes pour les anneaux de processus

—

Application à l'auto-stabilisation

Marie Duflot
Sous la direction de Laurent Fribourg

Table des matières

Introduction

Le sujet de ce stage a été l'étude de systèmes distribués particuliers que sont les chaînes circulaires de machines d'états finis, chacune étant vue comme un processus. Plus précisément, nous nous intéresserons à leur comportement limite.
Les systèmes distribués ont la particularité de ne pas avoir de contrôle global. En effet, chaque machine ne possède qu'une vision locale du système. Elle n'a accès qu'a son état et celui de ses voisins immédiats. De plus, la taille du système (*i.e.* le nombre de machines) n'est pas fixée, mais est un paramètre. Ces deux facteurs font qu'il est difficile de vérifier une propriété globale du système (comme par exemple qu'un et un seul des processus est dans un état particulier).

On peut représenter les comportements possibles de ce système par les chemins d'un graphe, dont les états sont les configurations du système et les arêtes les transitions. L'étude de l'évolution du système se ramène alors à la construction d'un graphe et l'analyse de certaines de ses propriétés. Classiquement, on teste des propriétés d'accessibilité (*i.e.* savoir dans quels états on peut arriver en partant de certains états particuliers appelés états initiaux et en empruntant des arêtes). Cela a été fait par exemple dans [JN00] pour vérifier des propriétés de sûreté (montrer qu'il est impossible de visiter un certain ensemble d'états).

On va s'intéresser ici à des propriétés de vivacité. Pour cela, on sera amené à chercher les états du graphe qui sont infiniment souvent visités (le long d'un chemin infini).

Notre méthode permet de résoudre des problèmes connus comme celui de l'auto-stabilisation. Informellement, un système est dit auto-stabilisant si son comportement lui permet, à partir d'une configuration quelconque, de se ramener en un nombre fini de transitions à un ensemble de configurations particulières dites légitimes. Nous illustrerons notre méthode en prouvant la correction d'algorithmes auto-stabilisants d'exclusion mutuelle, dans les parties 3 et 4.

En résumé, nous présentons dans ce mémoire une nouvelle méthode, basée sur la notion d'états récurrents permettant de vérifier des propriétés de vivacité sur les anneaux de processus (en particulier l'auto-stabilisation).

Travaux afférents

Une bonne présentation du domaine de l'auto-stabilisation ainsi que de son application à l'exclusion mutuelle peut être trouvée dans [Del95]. Les algorithmes présentés dans la suite ont été présentés par Dijkstra [Dij74] et Herman [Her92].
Récemment, des méthodes semi-automatiques d'étude de systèmes de réécriture

ou de chaînes de processus ont été exposées. L'article de [LHR97] porte sur la vérification de propriétés de sûreté. L'article [BBF99] prouve des propriétés de vivacité sur l'auto-stabilisation. Il utilise des résultats poussés de réécriture pour donner une méthode automatisée de preuve d'auto-stabilisation. Cependant, le cadre d'application est plus limité et la méthode donnée ne permet pas de traiter certains exemples comme l'algorithme de Dijkstra a K états.

Plan

Le chapitre 1 présente le cadre de l'auto-stabilisation, avec une présentation du problème ainsi qu'un exemple et une méthode classique de résolution.
Dans le chapitre 2, on trouve une présentation de notre méthode basée sur une notion nouvelle : celle d'états récurrents.
Le traitement d'un exemple classique qu'est l'algorithme de Dijkstra est fait dans le troisième chapitre. Y sont données deux preuves : une classique et la deuxième en utilisant notre méthode.
La partie 4 contient pour sa part une extension de la notion d'états récurrents ainsi que de la façon de les calculer dans le cas d'un algorithme probabiliste, qui est illustrée sur l'algorithme de Herman.

Chapitre 1

Le cadre de l'auto-stabilisation

1.1 Le problème

On s'intéresse à une chaîne de N processus identiques (N étant un paramètre). Chacun d'entre eux est une machine d'états finis qui peut de plus interagir avec ses voisins directs. Cela signifie qu'à chaque étape, un processus peut changer d'état en fonction de son état et de celui de ses voisins immédiats. Les deux processus des extrémités communiquent entre eux, éventuellement au moyen de règles particulières (c.f. FIG 1.1).

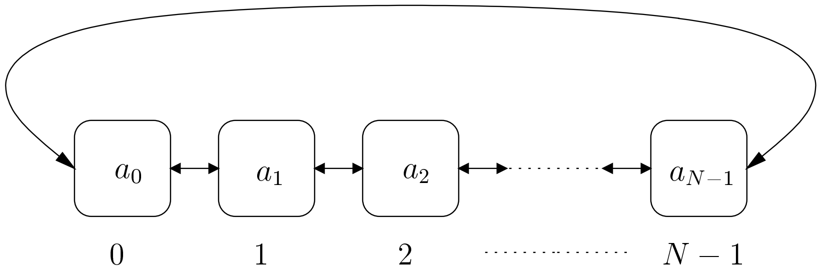

FIG. 1.1 – Une chaîne de processus

Pour plus de clarté, on va numéroter les états possibles des machines, par exemple $\{0, 1, ..., K-1\}$ pour des machines à K états. Les variables $x_0, x_1, ..., x_N$ désigneront l'état des N machines à un instant donné. Une configuration du système se décrit alors par la valeur de chacune de ces variables, c'est à dire par une donnée du type : $x_0 = a_0, x_1 = a_1, ..., x_{N-1} = a_{N-1}$ où les a_i prennent leur valeur dans $\Sigma = \{0, 1, ..., K-1\}$.

Pour des raisons de concision, on notera souvent une configuration plus simplement par la suite des états des machines, *i.e.* par le mot $a_0a_1...a_{N-1}$. L'étude de l'évolution du système revient alors à l'étude d'un système de réécriture (*i.e.* un ensemble de règles de réécriture) de mots.

Les règles d'évolution du système sont du genre :

$$\begin{array}{lrcl} 0 < i < N-1 : & x_{i-1} = a,\, x_i = b \text{ et } x_{i+1} = c & \Longrightarrow & x_i := b' \\ & x_0 = a,\, x_{N-2} = b \text{ et } x_{N-1} = c & \Longrightarrow & x_{N-1} := c' \\ & x_0 = a,\, x_1 = b \text{ et } x_{N-1} = c & \Longrightarrow & x_0 := a' \end{array} \tag{1.1}$$

L'exemple présente ici des règles mettant en jeu 3 positions consécutives, mais on peut aussi en avoir à moins, si le processus n'évolue qu'en fonction d'un de ses voisins et de son état propre.

On va s'intéresser au comportement de ce genre de systèmes, et plus précisément à son comportement limite. En effet, on va étudier ce qui se passe après une certaine phase transitoire, quand le comportement devient plus ordonné et répétitif.

Définition : Un *modèle* est un couple $\mathcal{M} = (C_\mathcal{M}, \rightsquigarrow_\mathcal{M})$, où $C_\mathcal{M}$ est un ensemble de configurations et $\rightsquigarrow_\mathcal{M}$ une relation de transition sur $C_\mathcal{M}$ (*i.e.* un sous ensemble de $C_\mathcal{M} \times C_\mathcal{M}$). Une *exécution* est une suite maximale (i.e. infinie ou non prolongeable) $u_0, u_1, u_2, ...$ telle que pour tout $i \geqslant 0$, $u_i \rightsquigarrow_\mathcal{M} u_{i+1}$ (qui est une notation plus agréable pour $(u_i, u_{i+1}) \in \rightsquigarrow_\mathcal{M}$).

Dans la suite, et tant qu'il n'y aura pas d'ambiguïté, on omettra de noter $\mathcal{M}$ en indice. D'autre part, les configurations seront des mots sur un alphabet Σ, et donc on aura $C = \Sigma^*$ (ou plutôt $\Sigma\Sigma^+$ car on considère des mots d'au moins deux lettres).

On peut voir le modèle $\mathcal{M}$ comme un graphe généralement infini dont les états sont les configurations de C et dont les arêtes sont les couples d'états en relation par $\rightsquigarrow$.
Le graphe est en général infini (car la longueur des mots n'est pas bornée) mais il est à branchement fini (chaque état a un nombre fini d'arêtes sortantes).
Soit $\mathcal{P}$ un sous ensemble (généralement infini) de C.

Définition : Un modèle $\mathcal{M}$ est *auto-stabilisant* pour $\mathcal{P}$ s'il existe un ensemble $\mathcal{L} \subseteq C$ de configurations appelées **légitimes** vérifiant les trois propriétés suivantes :

(i) **Correction :** toute configuration de $\mathcal{L}$ est dans $\mathcal{P}$ ($\mathcal{L} \subseteq \mathcal{P}$)

(ii) **Convergence :** toute exécution infinie atteint une configuration de $\mathcal{L}$ (*i.e.* le long de toute exécution infinie $u_0, u_1, u_2, ...$, il existe un indice i tel que $u_i \in \mathcal{L}$)

(iii) **Clôture :** pour tout couple (u, u') tel que $u \rightsquigarrow u'$ et $u \in \mathcal{L}$, on a $u' \in \mathcal{L}$.

Cette définition est inspirée de [Tel94] mais légèrement différente en ce sens que l'on impose ici la clôture et l'on s'intéresse à un sous-ensemble $\mathcal{P}$ de C sans préciser la façon dont il est défini.
La méthode que l'on va employer pour démontrer l'auto-stabilisation va être de construire un ensemble particulier que l'on appelle récurrent. Cet ensemble va contenir tous les mots infiniment souvent répétés le long d'exécutions infinies (pour une définition formelle, voir la partie 2.1) et va vérifier la propriété de convergence (*ii*) par construction. L'ensemble récurrent sera légitime si et seulement si il est inclus dans $\mathcal{P}$ (*i*) et vérifie la propriété de clôture (*iii*), ce qui sera vérifié en pratique sur les exemples considérés.

1.2 Un exemple

1.2.1 Description de l'algorithme de Dijkstra

On s'intéresse à l'algorithme de Dijkstra à K états (donc à valeurs dans $\{0, 1..., K-1\}$) et à N machines (c.f. [Dij74]). Les règles d'évolution du système sont les suivantes :

$$\begin{array}{rrcl} 0 < i < N : & x_{i-1} \neq x_i & \Longrightarrow & x_i := x_{i-1} \\ & x_{N-1} = x_0 & \Longrightarrow & x_0 := x_0 + 1 \end{array} \qquad (1.2)$$

Cet algorithme peut servir pour réaliser un protocole d'exclusion mutuelle. Supposons que plusieurs processus partagent une ressource, et qu'on veut vérifier qu'au bout d'un moment, jamais deux processus n'accèdent en même temps à la ressource (sûreté).

Pour cela on considère qu'il existe des jetons, et que le fait de posséder un jeton signifie pour un processus être autorisé à utiliser la ressource. L'exclusion mutuelle signifie donc qu'il n'existe qu'un seul jeton pour l'ensemble des processus.

Un processus possède un jeton et peut accéder à la ressource lorsque :
- soit c'est le premier de la chaîne et il est dans le même état que le dernier ($i.e.$ $i = 0$ et $x_i = x_{N-1}$)
- soit il n'est pas dans le même état que son voisin de gauche ($0 < i < N$ et $x_i \neq x_{i-1}$)

On va alors donner l'ensemble de configurations $\mathcal{P}$ correspondant a l'exclusion mutuelle.
Une configuration appartient à $\mathcal{P}$ si elle possède un et un seul jeton.
Comme on peut associer une configuration à un mot sur l'alphabet des états, l'ensemble $\mathcal{P}$ est alors formé des mots du type aa^+ ou a^+b^+, où a et b sont des états ($i.e.$ des lettres de Σ) distincts.

Remarque : On peut noter que, à toute configuration possible on peut appliquer au moins une règle (on n'a donc pas de blocage). En effet, soit il existe $0 < i < N$ tel que $x_i \neq x_{i-1}$ auquel cas on peut appliquer la première règle de 1.2, soit ils sont tous dans le même état, auquel cas $x_{N-1} = x_0$ et donc on peut appliquer la deuxième règle de 1.2.

Remarque : De plus, dans cet exemple, les processus possédant des jetons sont exactement ceux à qui une règle peut être appliquée. Le nombre de règles pouvant être appliquées à une configuration est donc égal au nombre de jetons qu'elle possède. La remarque précédente implique donc que toute configuration possède au moins un jeton.

1.2.2 Fonctionnement normal

On s'intéresse ici à savoir si, en partant d'une configuration de $\mathcal{P}$ et en effectuant une exécution quelconque on conserve la propriété d'exclusion mutuelle.

Lemme 1. *Si à un moment on a un seul jeton, alors en exécutant l'algorithme autant de fois que l'on veut, cette propriété est conservée.*

DÉMONSTRATION : En effet, à chaque pas, le processus qui le possède "passe le jeton" au suivant : on passe de $a^{N-l}b^l$ ($l > 0$) à $a^{N-l+1}b^{l-1}$ ou de a^N à $(a+1)a^{N-1}$. Il perd son jeton au profit de son voisin de droite.
On a ainsi préservation de la propriété de sûreté car l'ensemble des configurations à un jeton est clos par application des règles d'évolution.

■

1.2.3 Fonctionnement à partir d'un état dégradé

Le but est maintenant de savoir si cet algorithme s'auto-stabilise relativement à l'ensemble $\mathcal{P}$, c'est à dire si, en partant d'une configuration dégradée (qui possède donc plusieurs jetons car on a vu qu'il était impossible de n'en avoir aucun), on arrive en appliquant l'algorithme 1.2, à l'exclusion mutuelle en un temps fini.

Dijkstra a montré (c.f. [Dij74]) que pour un nombre d'états $K \geqslant N$, l'ensemble $\mathcal{P}$ des configurations possédant un seul jeton est légitime (une démonstration de ce résultat sera donnée dans la partie 3.1). On a donc auto-stabilisation relativement à l'ensemble $\mathcal{P}$. Il a fallu attendre 1999 pour qu'Hoepman [Hoe99] montre que ce résultat est encore vrai pour $K = N - 1$.

On va dans le chapitre 3 s'intéresser au cas général (on ne précise pas le domaine de valeurs de K) et montrer une propriété de convergence, en étudiant un ensemble particulier dans lequel passent toutes les exécutions (infinies).
Pour $K + 1 \geqslant N$ on va pouvoir retrouver le résultat de Dijkstra, mais on va aussi s'intéresser à savoir vers quoi converge le système lorsqu'on n'a pas auto-stabilisation pour $\mathcal{P}$.

1.3 Méthode classique de preuve d'auto-stabilisation

Une méthode souvent utilisée pour des preuves d'auto-stabilisation est la suivante, décrite dans [Tel94]. Considérons un sous-ensemble $\mathcal{P}$ de C et un ensemble $\mathcal{L}$ qu'on veut prouver légitime relativement à $\mathcal{P}$.
Le point clef de la démonstration est la preuve de convergence : en effet, la clôture et la correction se font de manière classique, en vérifiant des inclusions d'ensembles : Il suffit de montrer que $\mathcal{L} \subseteq \mathcal{P}$ et que l'ensemble image de $\mathcal{L}$ par l'application d'une étape de l'algorithme est inclus dans $\mathcal{L}$.

Pour prouver la convergence vers $\mathcal{L}$, on a un ensemble de conditions qui est suffisant :

- Il n'y a pas de blocage pour l'algorithme considéré (c'est à dire de configuration pour laquelle aucune règle ne peut s'appliquer), sauf peut-être dans $\mathcal{L}$.
- il existe une fonction (que l'on appelle norme) $f : C \mapsto \mathbb{N}$ telle que pour chaque transition $\gamma \rightsquigarrow \delta$, on ait $f(\gamma) > f(\delta)$ ou $\gamma \in \mathcal{L}$

alors $\mathcal{L}$ vérifie la propriété de convergence.

Dans l'article [Gho93] Ghosh utilise ce genre de raisonnement pour prouver l'auto-stabilisation de son algorithme, à la différence près que la mesure qu'il utilise ne décroît pas à chaque étape mais toutes les au plus trois étapes.

La méthode de preuve utilisée dans la suite est différente de celle-ci, et basée sur la notion d'états "récurrents".

Chapitre 2

Configurations récurrentes

Cette section explique comment traiter de manière originale le problème de l'auto-stabilisation en utilisant la notion de configurations récurrentes.

2.1 Définitions et notations

Dans la suite, Σ désignera un alphabet fini, et dans les exemples ce sera l'ensemble des états des machines considérées. u, v, w et leurs variantes (u', v_1 etc ...) désigneront des mots de Σ^* et a, b, c et leurs variantes des lettres de Σ. On définit également un nouveau caractère _ qui n'est pas dans Σ et un nouvel alphabet $\Gamma = \Sigma \cup \{\_\}$. Les lettres $\mathcal{U}$ et $\mathcal{V}$ désigneront des mots de Γ^*

Une *règle de réécriture* se note de la manière suivante : $R :\ \langle \mathcal{U}, \mathcal{V} \rangle$, où $\mathcal{U}$ et $\mathcal{V}$ sont des mots sur $\_\Sigma^*\_ \cup \Sigma^*\_\Sigma^*$ (ils sont du même type parmi les 2). Intuitivement, le caractère _ signifie qu'on va devoir mettre à la place un sous mot quelconque de Σ^*.
$\mathcal{U}$ est appelé *membre gauche* de R, et $\mathcal{V}$ est le *membre droit.*
On appelle *système de réécriture* un ensemble de règles de réécriture, qui peuvent par exemple servir à décrir un algorithme.

Pour coller avec le modèle de processus qui ne communique qu'avec ses deux voisins immédiats, on va dans la suite considérer des règles qui ne font intervenir que deux ou trois lettres de Σ, et qui conservent la longueur, par exemple :

$$R_1 :\ \langle \_abc\_, \_ab'c\_ \rangle, \text{ ou } R_2 :\ \langle a\_b, a'\_b \rangle$$

Définition : Un mot u de Σ^* est *réductible* via un système de réécriture $\mathcal{S}$ si :

- Soit u peut se découper en $u'.v.u''$ tels qu'il existe une règle $\langle \mathcal{U}, \mathcal{V} \rangle$ dans $\mathcal{S}$ avec $\mathcal{U} = \_v\_$
- Soit u peut se découper en $v_1.u'.v_2$ tels qu'il existe une règle $\langle \mathcal{U}, \mathcal{V} \rangle$ dans $\mathcal{S}$ avec $\mathcal{U} = v_1\_v_2$.

Une *réduction* est alors l'aplication d'une règle de réécriture à un mot (qui doit être réductible). Ainsi, sur les exemples, si $u = u'.abc.u''$ on peut lui appliquer la règle R_1, et on obtient le mot $v = u'.ab'c.u''$ qui est appelé *forme réduite*

de u (par R_1).
De la même manière, si $u = a.u'.b$, on peut lui appliquer la règle R_2, et on obtient le mot $v = a'.u'.b$.
On note $u \rightarrow_{\mathcal{S}} v$ pour dire que u se réduit à v par l'application d'une règle de $\mathcal{S}$, ou simplement $u \rightarrow v$. On note $u \rightarrow^n_{\mathcal{S}} v$ si u se réduit à v au moyen d'une suite de n réductions consécutives de $\mathcal{S}$, c'est à dire il existe des mots
$u_0, u_1, ..., u_n$ tels que $u = u_0 \rightarrow_{\mathcal{S}} u_1 \rightarrow_{\mathcal{S}} ... \rightarrow_{\mathcal{S}} u_n = v$ (par convention si $n = 0$ on a $u = v$ et $u \rightarrow^0 u$). On note $u \rightarrow^*_{\mathcal{S}} v$ (resp. $u \rightarrow^+_{\mathcal{S}} v$) si il existe un $n \geqslant 0$ (resp $n > 0$) tel que $u \rightarrow^n_{\mathcal{S}} v$.
Par abus de langage, une règle $R : \langle \mathcal{U}, \mathcal{V} \rangle$ sera parfois notée $R : \mathcal{U} \rightarrow \mathcal{V}$.

Il est souvent possible d'appliquer plusieurs règles à un même mot, à la même position ou à des positions différentes. Par exemple, le mot *adabcdb* peut être réduit via R_1 en $adab'cdb$, ou via R_2 en $a'dabcdb$. Lorsqu'une telle situation se présente, on choisit arbitrairement la réduction qu'on va effectuer. On n'effectue jamais deux réductions simultanément. Si le long d'une exécution on passe plusieurs fois (resp. une infinité de fois) par une même configuration, on peut très bien toujours choisir d'effectuer la même réduction (ce qui ne sera plus le cas dans le cadre probabiliste).

On a ainsi un modèle comme défini à la partie 1.1, pour lequel l'ensemble C des configurations est Σ^* et la relation de transition sur Σ^* est $\rightarrow$, la relation de réduction.

Définition : Lors d'une réduction du type
$a_0a_1...a_{N-1} \rightarrow b_0b_1..b_{N-1}$, on dit que la position i est *affectée* si $a_i \neq b_i$.

Remarque : Dans cette définition, la notion de position a un sens car les réductions conservent la longueur des mots.

Dans toute la suite, on ne considère que des règles dont les réductions associées n'affectent qu'une seule position à la fois.
Une réduction est dite *top* si elle affecte la dernière lettre du mot (*i.e.* la position $N-1$), elle est dite *bottom* si elle affecte la première lettre du mot (*i.e.* la position 0), et *middle* sinon.
Par extension, on peut définir ces trois notions pour les règles. Une règle $\langle \mathcal{U}, \mathcal{V} \rangle$ est top (resp. bottom) si la dernière lettre (resp. la première lettre) de $\mathcal{U}$ et celle de $\mathcal{V}$ sont différentes. Elle est middle si $\mathcal{U}$ et $\mathcal{V}$ sont dans $\_\Sigma^*\_$ ou si la règle affecte une lettre qui n'est ni la première ni la dernière de $\mathcal{U}$.

Remarque : A priori rien n'oblige une règle à appartenir à une seule de ces catégories. Par exemple $\langle \_abc\_, \_abc'\_ \rangle$ est à la fois top et middle. Cependant dans la suite chaque règle sera d'un seul type. En effet, les règles considérées dans les démonstrations seront toujours à trois lettres et les règles middle affecteront toujours la lettre du milieu (comme pour la règle R_1 décrite précédemment).
De plus, les règles à deux lettres peuvent se généraliser en règles à trois lettres. Seulement, si on veut une description précise, une règle à deux lettres se généralise en $|\Sigma|$ règles à trois lettres. Par exemple, la règle R_2 se généralise en $\langle ac\_b, a'c\_b \rangle$, mais il faut écrire une règle pour chaque valeur possible de c.

Définition : Un système de réécriture est *équitable* si le long d'une suite infinie quelconque de réductions, toute position est affectée une infinité de fois.

Soit Λ une suite infinie (resp. finie) de réductions $u_0 \rightarrow u_1 \rightarrow ... \rightarrow u_n \rightarrow ...$

(resp. $u_0 \to u_1 \to ... \to u_n$). Un *préfixe fini* de Λ est une suite de réductions de la forme $u_0 \to u_1 \to ... \to u_k$ avec k entier quelconque (resp. $k \leqslant n$).

La propriété d'équité signifie que le long d'une suite infinie de réductions suivant un système de réécriture équitable, si on enlève un préfixe fini quelconque, alors on peut être sûr que dans la suite restante, toute position sera encore affectée. C'est cette caractérisation de l'équité que l'on va utiliser dans la suite.

Enfin, on va présenter l'élément majeur de notre méthode : la notion d'états récurrents.

Définition : Un mot u est dit *récurrent* pour un système de réécriture $\mathcal{S}$ s'il existe une suite (infinie) de réductions suivant $\mathcal{S}$ le long de laquelle u est répété une infinité de fois, c'est à dire une suite $w_0 \to_{\mathcal{S}} w_1 \to_{\mathcal{S}} w_2 \to_{\mathcal{S}} ...$ telle qu'on ait une infinité d'indices i vérifiant $w_i = u$.

Le langage des mots récurrents sera noté $\mathcal{R}$.
On peut déjà remarquer que la définition même du langage récurrent entraîne, lorsqu'il n'y a pas de blocages, la convergence (définie à la partie 1.1) vers $\mathcal{R}$. En effet, comme les règles considérées conservent le nombre de lettres des mots, tous les mots le long d'une exécution infinie ont la même longueur. Or il y a un nombre fini de mots d'une longueur donnée vu que l'alphabet Σ est fini. Cela veut dire qu'il existe un ou plusieurs mots infiniment souvent répétés le long de cette exécution, et ceux-cis sont dans $\mathcal{R}$.

On va chercher dans la suite à déterminer l'ensemble des mots récurrents pour les différents systèmes étudiés.

2.2 Notre Méthode

Cette partie a pour but de décrire le plan de la méthode, en donnant les étapes majeures de la construction du langage récurrent.

2.2.1 Langages rationnels

On ne sépare pas dans notre étude les différentes valeurs possibles de N, mais au contraire on fait une union sur toutes les valeurs de N ce qui nous permet de considérer des langages rationnels au lieu de mots de longueur donnée. Ainsi,

$$\mathcal{R} = \bigcup_{N \geqslant 2} \{\text{mots récurrents de taille N}\}$$

Si l'on veut s'intéresser à un système précis dont on connait le nombre de processus, il suffit alors de prendre l'intersection des langages avec Σ^N, l'ensemble des mots de taille N. Ainsi l'ensemble des mots récurrents pour un système à N processus est $\mathcal{R} \cap \Sigma^N$.

On considère dans la suite des mots d'au moins deux lettres (en effet, c'est la taille des plus petites règles que l'on considère). Par exemple, au lieu du mot $a^{N-1}b$, on considèrera le langage $\bigcup_{N>1} a^{N-1}b = a^+b$.

2.2.2 Graphe

Pour représenter le comportement du système, on va construire un graphe décrivant les évolutions possibles. Les nœuds sont appelés des *boîtes* et étiquetés par des langages rationnels et les arêtes représentent de manière synthétisée les réductions qui peuvent s'appliquer (comme on le verra par la suite).

De manière générale, le graphe peut se séparer en deux parties (cf FIG 2.1).

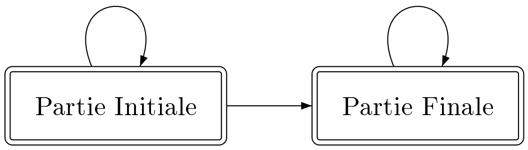

FIG. 2.1 – Schéma général de la méthode

- La première partie décrit le comportement initial du système. C'est une sorte de phase transitoire et une partie du travail qui sera fait sur ce graphe consistera à montrer que l'on ne peut pas rester indéfiniment dans cette partie. Comme le graphe doit pouvoir décrire tous les comportements possibles suivant le système de réécriture considéré, on peut donc vouloir décrire une exécution partant de n'importe quelle configuration. Chaque mot de Σ^* doit donc être au moins inclus dans un des états de la partie initiale.[1]
- La deuxième partie décrit ce qui se passe après cette phase transitoire. Elle est donc stable par application d'une suite quelconque de réductions, car on ne doit pas pouvoir revenir dans l'autre partie. L'ensemble des mots contenus dans les états de ce sous-graphe est plus restreint.

Le graphe doit simuler le système de réécriture considéré ($\mathcal{S}$), c'est à dire qu'à toute exécution suivant $\mathcal{S}$ doit correspondre un chemin dans le graphe qui lui corresponde. Par exemple si on prend l'exécution $u_0, u_1, ..., u_n, ...$, il doit exister un chemin dans le graphe $e_0, e_1, ..., e_n, ...$ (où les e_i sont des états du graphe) tel que : pour tout i, le mot u_i est inclus dans le langage qui étiquette l'état e_i.

Remarque : On ne détaille pas toutes les réductions possibles à partir de chaque état, on les regroupe. Cela se fait souvent en étiquetant les arrêtes suivant le type de réduction qu'elles représentent (middle, top ou bottom). On a parfois besoin de détailler un peu plus (comme on le verra à la FIG. *3.3*).

Pour déterminer l'ensemble des mots récurrents, on va ensuite chercher les cycles dans la partie finale. Les mots récurrents, comme ils doivent être infi-

[1] Dans la théorie, on peut en effet partir de n'importe quelle configuration mais il arrive que l'on ne considère que certaines configurations qui vérifient une propriété particulière stable par réécriture. Dans l'algorithme de Ghosh [Gho93], les processus 0 et $N-1$ doivent satisfaire certaines contraintes particulières (le processus 0 à pour états possibles $\{1,3\}$ et le processus $N-1$ à pour états $\{0,2\}$ alors que tous les autres ont pour états $\{0,1,2,3\}$)).

niment souvent répétés le long d'au moins une exécution, sont inclus dans les états de ces cycles.

Le travail consiste donc à regrouper les mots en langages pour former des nœuds du graphe, et ceci de manière adéquate pour décrire le système avec un degré de précision bien choisi. En effet le graphe doit être d'une part assez synthétique pour que sa compréhension soit relativement facile, et d'autre part assez détaillé pour pouvoir le séparer en deux parties vérifiant les conditions ci-dessus. Ce découpage permettra alors de faciliter la recherche des mots récurrents.

2.2.3 Schéma de la procédure de preuve

Cette partie récapitule les différentes étapes de notre méthode et l'ordre dans lequel les effectuer.
Dans la suite , on va noter $\mathcal{F}$ le langage composé de tous les mots qui sont contenus dans les étiquettes des états de la partie finale.

La méthode consiste donc en trois étapes :

1. Tout d'abord on construit le graphe, et on vérifie qu'il simule le système de réécriture $\mathcal{S}$.
2. Ensuite on montre que la partie finale est close par application de réductions
3. Puis on montre que l'on ne peut pas boucler dans la partie initiale. Cela implique que comme les mots récurrents sont infiniments souvent répétés le long d'au moins une exécution, alors ils sont inclus dans les états de la partie finale et donc $\mathcal{R} \subseteq \mathcal{F}$. Dans la pratique, cette propriété se montre généralement en utilisant des arguments portant sur l'équité du système.
4. Enfin, on construit le langage récurrent $\mathcal{R}$ par raffinement (si nécessaire) du langage $\mathcal{F}$. Lorsqu'il n'y a pas besoin de raffiner on a déjà une inclusion ($\mathcal{R} \subseteq \mathcal{F}$). Dans tous les cas, l'inclusion de $\mathcal{R}$ dans un sous-ensemble $\mathcal{E}$ de C se montre en exhibant pour tout mot de $\mathcal{E}$ une exécution le long de laquelle ce mot est infiniment souvent répété. En général on exhibe un cycle contenant le mot.

Une présentation plus fine des liens entre langage récurrent et ensemble légitime est donné dans l'annexe A. L'annexe B présente une condition suffisante pour que le langage récurrent soit légitime.

Dans le paragraphe suivant, on va considérer des hypothèses qui vont permettre de simplifier les preuves. De plus ces hypothèses seront valables dans les exemples traités ultérieurement.

2.3 Hypothèses simplificatrices

2.3.1 Pas de blocage

Dans toute la suite on ne va considérer que des systèmes de réécriture tels qu'à tout mot, une règle au moins peut être appliquée.
Par conséquent, comme une exécution est une suite maximale de réductions consécutives, elle est forcément infinie.

2.3.2 Règles conservant une mesure

Pour simplifier les preuves, on peut considérer une mesure φ sur les mots, à valeurs dans $\mathbb{N}$. On la choisit telle qu'aucune règle de $\mathcal{S}$ ne la fasse augmenter. Ainsi le long d'une séquence, φ ne peut que décroître ou rester constante.
Si on a une suite infinie $w_0 \to_{\mathcal{S}} w_1 \to_{\mathcal{S}} ... \to_{\mathcal{S}} w_n \to_{\mathcal{S}} ...$, la suite des $\varphi(w_i)$ est décroissante dans $\mathbb{N}$ (et minorée par 0) donc elle converge. Il existe donc un indice i_0 tel que $\forall\ i \geqslant i_0$, $\varphi(w_i) = \varphi(w_{i+1})$. La suite $\varphi(w_j)$ est stationnaire à partir de $j = i_0$.

On va alors considérer un sous-ensemble de $\mathcal{S}$, noté $\mathcal{S}^\varphi$ et défini par :

$$u \to_{\mathcal{S}^\varphi} v \iff u \to_{\mathcal{S}} v \text{ et } \varphi(u) = \varphi(v)$$

Les règles de $\mathcal{S}^\varphi$ ne sont pas forcément des règles de $\mathcal{S}$, mais plus souvent des sous-cas de règles de $\mathcal{S}$. Par exemple, comme nous le verrons dans la section 3, la règle middle conservant la mesure α est : $\langle \_abb\_, \_aab\_ \rangle$, pour $a \neq b$, et non plus $\langle \_ab\_, \_aa\_ \rangle$. On peut donc avoir des règles à trois lettres au lieu de deux comme ici, mais elles n'affectent toujours qu'une seule lettre.
Si par contre on a généralisé les règles à deux lettres en règles à trois lettres, alors les règles de $\mathcal{S}^\varphi$ sont des règles de $\mathcal{S}$.

Proposition 1. *L'ensemble des mots récurrents pour $\mathcal{S}$ est le même que pour $\mathcal{S}^\varphi$.*

Démonstration : Comme les règles de $\mathcal{S}^\varphi$ sont des sous-cas des règles de $\mathcal{S}$, toute suite de réductions de $\mathcal{S}^\varphi$ est une suite de réductions de $\mathcal{S}$, donc tout mot récurrent de $\mathcal{S}^\varphi$ est un mot récurrent pour $\mathcal{S}$. Réciproquement, si u est un mot récurrent pour $\mathcal{S}$, il existe une suite de réductions le long de laquelle u est répété une infinité de fois. Quitte à enlever un préfixe fini, on a une suite infinie de réductions de $\mathcal{S}$ qui conservent φ (d'où une suite infinie de réductions de $\mathcal{S}^\varphi$) qui contient une infinité d'occurences de u. Le mot u est donc récurrent pour $\mathcal{S}^\varphi$.

■

Comme on restreint l'ensemble des règles à considérer, il est possible que l'on fasse apparaître des blocages "artificiels", c'est à dire qui ne correspondent pas à des réels blocages du système. Cependant ils ne posent pas de problème, car lorsqu'on est bloqué pour le système de réécriture $\mathcal{S}^\varphi$, c'est que dans le système complet on peut (et donc on va) effectuer une réduction qui fait strictement diminuer φ. Or on a supposé qu'on ne s'intéressait qu'à la partie finale des exécutions, *i.e.* celle qui ne fait plus varier φ, donc les réductions avec blocage ne sont pas à considérer.

D'autre part, et contrairement à la méthode classique de preuve énoncée dans la partie 1.3, l'utilisation d'une mesure n'est pas ici un argument clef de notre méthode. Elle sert juste à ne considérer que certaines réductions, et ceci afin de faciliter la preuve.

Chapitre 3

Retour sur l'algorithme de Dijkstra à K états

Revenons à l'exemple : on veut trouver l'ensemble des mots récurrents pour le problème de Dijkstra à K états. La preuve donnée ci-dessous est nouvelle et illustre la méthode décrite dans la partie précédente. Elle retrouve le résultat dû à Hoepman [Hoe99] et donne les mots récurrents dans les deux cas : $K+1 \geqslant N$ et $K+1 < N$.

On peut écrire les règles d'évolution du système (c.f. 1.2) sous forme de règles de réécriture. Cela donne :

$$\mathcal{S}_{Dij} = \begin{cases} \langle \_ab\_, \_aa\_\rangle & \text{(top et middle)} \\ \langle a\_a, (a+1)\_a\rangle & \text{(bottom)} \end{cases}$$

où $a \neq b$, ou si l'on transforme les règles à deux lettres en règles à 3 lettres :

$$\begin{array}{cc} \langle \_abc\_, \_aac\_\rangle & \text{(middle)} \\ \langle c\_ab, c\_aa\rangle & \text{(top)} \\ \langle ac\_a, (a+1)c\_a\rangle & \text{(bottom)} \end{array}$$

où $a \neq b$ et c est quelconque

3.1 Une preuve classique

La preuve ci-dessous montre le résultat de Dijkstra à savoir que l'ensemble des configurations à un jeton est légitime, mais uniquement dans les cas où $K \geqslant N$. Elle est inspirée de [Tel94].
Pour démontrer l'autostabilisation, on va montrer que l'ensemble $\mathcal{P}$ défini à la partie 1.2 est légitime. Pour cela on va montrer les trois propriétés qui permettront d'affirmer que $\mathcal{P}$ est un ensemble légitime.

Tout d'abord, l'ensemble $\mathcal{P}$ est inclus dans lui-même et vérifie donc la propriété de correction.

Lemme 2. *L'algorithme de Dijkstra pour $K \geqslant N$ converge vers $\mathcal{P}$.*

Démonstration : On va ici utiliser une mesure qui décroît le long de certaines suites de réductions, mais ce sera pour montrer qu'elles ne peuvent pas être infinies, et non pour montrer directement la propriété de convergence.

On définit la mesure :

$$F = \sum_{i \in J} (N - 1 - i)$$

où $J = \{i \geqslant 1 |$ le $i^{\text{ème}}$ processus possède un jeton$\}$

Toute réduction qui n'affecte pas le processus 0 fait décroître strictement la mesure F, étant donné que le jeton passe du processus i au processus $i+1$, ou éventuellement du processus $N-1$ au processus 0 (qui n'est jamais dans S). Ainsi il est impossible d'avoir une exécution (forcément infinie car il n'y a pas de blocage pour cet algorithme) qui n'affecte pas infiniment le processus 0.

D'autre part, comme on a au moins autant d'états que de machines, soit on a un état qui n'est pas présent dans la configuration initiale que l'on appelle alors u_0, soit toutes les machines sont dans des états différents. Dans le deuxième cas, on considère la configuration obtenue après la première réduction. Comme une des machines a changé d'état, il y a alors un état qui n'est pas présent dans cette configuration que l'on va noter u_0, et deux machines qui sont dans le même état.

Comme le processus 0 augmente son état d'un (modulo K) à chaque fois qu'il évolue, il va atteindre un état, mettons a qui n'est pas dans u_0 en au plus $N-1$ changements d'état. Comme la seule chose que peuvent faire les autres processus est de copier leur voisin de gauche, lorsque le processus 0 prend l'état a, il est le seul dans cet état.
Il faut alors, pour pouvoir à nouveau appliquer une règle bottom que le dernier processus passe lui aussi dans l'état a et pour cela que tous les processus soient dans l'état a, ce qui est une configuration de $\mathcal{P}$.
Comme on a vu que l'état du processus 0 était infiniment modifié, on va donc obligatoirement passer par une configuration de $\mathcal{P}$.

■

Lemme 3. *L'ensemble $\mathcal{P}$ est clos par réduction suivant le système de réécriture $\mathcal{S}_{Dij}$.*

Démonstration : On a vu précédemment (c.f. 1.2.2) que le fait d'appliquer une réduction à une configuration à un seul jeton donne une autre configuration à un seul jeton. Cela assure la propriété de clôture.

■

Corollaire 1. *L'ensemble $\mathcal{P}$ des configurations à un seul jeton est légitime pour l'algorithme de Dijkstra, dans le cas où le nombre K d'états de chaque machine est supérieur au nombre N de machines.*

La preuve de Hoepman [Hoe99] (*i.e.* pour $K+1 = N$) utilise un raisonnement similaire en précisant un peu plus ce raisonnement pour montrer qu'à un moment on arrive forcément dans une configuration où au moins trois machines sont dans le même état, et ainsi un état qui n'est pas dans la configuration considérée. La suite de la preuve de convergence est identique à celle-ci, ainsi que la preuve de clôture.

3.2 Notre preuve

3.2.1 Règles préservant le nombre de sites actifs

Comme annoncé à la partie 2.2, on va introduire une mesure sur les mots : on dit qu'une position d'un mot est *active* pour un système de réécriture $\mathcal{S}_{Dij}$ s'il existe une réduction suivant $\mathcal{S}_{Dij}$ qui affecte cette position. Pour un mot u on notera $\alpha(u)_{\mathcal{S}_{Dij}}$ le nombre de positions actives de u suivant $\mathcal{S}_{Dij}$, ou plus simplement $\alpha(u)$ s'il n'y a pas de risque de confusion. On note que dans l'exemple, l'application d'une réduction suivant $\mathcal{S}_{Dij}$ fait disparaitre au moins une position active (celle qui est affectée) et en fait apparaître au plus une nouvelle (celle qui la suit). Cela implique que α ne peut pas croître le long d'une suite de réductions, et on peut prendre α comme mesure.

Plus précisément, si l'on considère chacun des cas de réductions possibles et leur influence sur α, cela donne :

- $\underset{a\neq b}{u.abc.v} \longrightarrow u.aac.v$ si $b = c$, α reste stable
  si $b \neq c$, $\alpha := \alpha - 1$
- $\underset{a\neq b}{c.u.ab} \longrightarrow c.u.aa$ si $b = c$, $\alpha := \alpha - 2$
  si $b \neq c$ et $a = c$, α reste stable
  si $b \neq c$ et $a \neq c$, $\alpha := \alpha - 1$
- $cb.u.c \longrightarrow (c+1)b.u.c$ si $b = c$, α reste stable
  si $c \neq b \neq (c+1)$, $\alpha := \alpha - 1$
  si $b = c+1$, $\alpha := \alpha - 2$

où u et v sont des mots quelconques sur Σ^*.

D'après la Proposition 1, on peut ne considérer que des règles qui conservent α. On obtient ainsi le système de réécriture $\mathcal{S}^{\alpha}_{Dij}$ suivant :

$$\begin{array}{rcll} abb & \longrightarrow & aab & \text{middle} \\ aa\_a & \longrightarrow & (a+1)a\_a & \text{bottom} \\ a\_ab & \longrightarrow & a\_aa & \text{top} \end{array} \quad (3.1)$$

3.2.2 Equité

Proposition 2. *Dans l'algorithme de Dijkstra à K états, le système $\mathcal{S}^{\alpha}_{Dij}$ des règles conservant le nombre α est équitable.*

DÉMONSTRATION : Raisonnons par l'absurde : supposons qu'il existe une position du mot de départ, mettons i, qui ne soit affectée qu'un nombre fini de fois. Il existe donc un moment le long de la suite à partir duquel x_i ne change plus jamais sa valeur, que nous appellerons a. On a donc un mot de la forme $x_1x_2...x_{i-1}ax_{i+1}...x_{N-1}$ duquel est issue une suite infinie de réductions le long de laquelle ce a n'est jamais changé.

Alors x_{i+1} peut changer au plus une fois de valeur (si au départ il est $\neq a$ il peut changer une fois et devenir égal à a, sinon il ne peut pas changer), de même comme x_{i+1} change au plus une fois, x_{i+2} changera au plus 2 fois ....etc.. Lorsque $i + m = N$ on applique la règle à x_0 et on a la même propriété : x_0 peut changer de valeur au plus une fois de plus que x_{N-1}, ce qui fait que sans changer létat de x_i, on peut effectuer au plus $\frac{N(N-1)}{2}$ réductions, mais en aucun cas une infinité.

■

Le système est donc équitable et on va pouvoir utiliser cette propriété dans le cœur de la preuve.

3.2.3 Construction de l'ensemble récurrent

On va à présent déterminer l'ensemble des états récurrents. Pour cela, et comme on le verra dans le graphe 3.3 on va mettre en évidence le fait que l'application d'une suite de réductions fait que le mot gagne en "régularité" (en un sens que l'on verra plus précisément par la suite), jusqu'à obtenir en un temps fini un mot récurrent.

On note $H_a = (a-1)^+(a-2)^+...(a-K+1)^+a^+$. Sachant que l'addition est définie modulo K, cela revient à dire que $a-K+1 = a+1$ et donc
$H_a = (a+K-1)^+(a+K-2)^+...(a+1)^+a^+$.

On définit le langage $\mathcal{Q}$ par $\mathcal{Q}_1 \cup \mathcal{Q}_2$ avec :

$$\mathcal{Q}_1 = (\bigcup_{a\in\Sigma} a^+H_a^*) \cap (\bigcup_{b\in\Sigma} \Sigma^*bb\Sigma^*)$$

$$\mathcal{Q}_2 = \bigcup_{a\in\Sigma}(a+1)^+a^+H_a^*$$

On a donc dans $\mathcal{Q}$ des mots pour lesquels une lettre est toujours soit égale à sa voisine de gauche, soit inférieure de 1. De plus,

- soit la première et la dernière lettre sont égales et le mot possède de plus un redoublement (deux lettres consécutives identiques) qui sera nécessaire pour appliquer les réductions middle et ou bottom. La configuration est alors dans $\mathcal{Q}_1$,
- soit la première lettre excède de 1 la dernière auquel cas, s'il n'y a pas redoublement on peut effectuer une réduction top et sinon une réduction middle. La configuration est alors dans $\mathcal{Q}_2$

On va voir dans la suite l'intérêt et le rôle de ces configurations particulières.

Théorème 1. *L'ensemble $\mathcal{R}$ des configurations récurrentes pour l'algorithme de Dijkstra à K états est $\mathcal{Q}$.*

Remarque : Ce théorème implique que si on a $K+1 \geqslant N$, les seuls mots récurrents de taille N sont $(a+1)^la^{N-l}$ (avec $1 \leqslant l \leqslant N$ et $a \in \Sigma$). En effet, le langage H_a^* est réduit à ε qui est le mot vide.
On retrouve donc l'analogue du résultat de Hoepman [Hoe99] mais ici pour le langage récurrent. On donne également les mots récurrents dans le cas $K+1 < N$.

L'idée directrice de la preuve est que l'application répétée de réductions bottom induit une certaine "régularité" sur la partie gauche des mots, régularité que les réductions middle et bottom ne peuvent que conserver ou augmenter. On va alors montrer, en se servant de l'équité du système montrée dans la partie 3.2.2 que cette partie régulière va augmenter en taille jusqu'à recouvrir tout le mot, puis que les mots réguliers obtenus forment l'ensemble des mots récurrents.

On va découper notre preuve en plusieurs étapes : Tout d'abord on va s'intéresser à cette notion de régularité et à son augmentation le long des exécutions. On va ensuite construire le graphe correspondant qui met en évidence cette régularité, et vérifier sa correction. Puis on va démontrer que l'on ne peut pas boucler dans la partie initiale. Enfin, par une double inclusion, on va prouver que le langage récurrent $\mathcal{R}$ est égal au langage $\mathcal{Q}$ défini précédemment.

Partie régulière, langages $M(b, a)$

Considérons le langage suivant :

$$M(b,a) = \bigcup_{b-m=a \bmod K} b^+(b-1)^+...(b-m)^+$$

On définit alors le langage

$$M = \bigcup_{a\in\Sigma} \bigcup_{b\in\Sigma} M(a,b)$$

On peut noter que $M(a,a) = a^+H_a^*$ et $M(a+1,a) = (a+1)^+a^+H_a^*$ ce qui permet de caractériser d'une autre manière les langages $\mathcal{Q}_1$ et $\mathcal{Q}_2$ par

$$\mathcal{Q}_1 = (\bigcup_{a\in\Sigma} M(a,a)) \cap (\bigcup_{b\in\Sigma} \Sigma^* bb\Sigma^*)$$

$$\text{et } \mathcal{Q}_2 = \bigcup_{a\in\Sigma} (a+1)^+M(a,a)$$

Tout mot w de Σ^+ se décompose en $u.v$ où u et v sont des facteurs, tels que $u \in M$ et $v \in \Sigma^*$ Par exemple si $w = a.v$ où a est une lettre de Σ, on a $u = a \in M$ et $v \in \Sigma^*$. Comme on considère des mots de plus de deux lettres, on peut même dire que tout mot de $\Sigma.\Sigma^+$ se décompose en $u.v$ où $u \in M$ et $v \in \Sigma^+$.

Si w est l'origine d'une suite de réductions infinie $w = w_0, w_1, ...w_i, ...$, on va décomposer les $w_i (i \in \mathbb{N})$ en $u_i.v_i$, avec les u_i dans M et v_i dans Σ^*. Pour cela on va partir de $w_0 = u_0.v_0$ avec $u_0 \in \Sigma$ et choisir pas à pas les u_i et v_i. On va considérer les différents cas de réductions qui peuvent s'appliquer à $w_i = u_i.v_i = (a+m)^{\lambda_{i,m}}(a+m-1)^{\lambda_{i,m-1}}...(a+1)^{\lambda_{i,1}}a^{\lambda_{i,0}}.v_i \in M(a+m,a)$, illustrés FIG. 3.1 :

Les règles 1 à 4 représentent les différents cas de réductions middle, les règles 5 et 5' des réductions bottom, et les règles 6 et 6' décrivent pour leur part les deux cas de réductions top.
Pour bien fixer les idées, dans la partie gauche, on souligne la lettre qui est affectée par la réduction.

On va tout d'abord montrer l'influence des réductions middle sur la régularité du mot réduit.

Lemme 4. *Pour tous a et b dans Σ, le langage $M(b,a)\Sigma^*$ est clos par application des règles* middle *de $\mathcal{S}_{Dij}^{\alpha}$.*

Cas 1 :	$\llcorner\ b\underline{c}c\ \lrcorner\ \llcorner\ \lrcorner$	$\xrightarrow{\text{middle}}$	$\llcorner\ bbc\ \lrcorner\ \llcorner\ \lrcorner$
Cas 2 :	$\llcorner\ b\underline{c}\lrcorner\ \llcorner c\ \lrcorner$	$\xrightarrow{\text{middle}}$	$\llcorner\ bbc\lrcorner\ \llcorner\ \lrcorner$
Cas 3 :	$\llcorner\ b\lrcorner\ \llcorner\underline{c}c\ \lrcorner$	$\xrightarrow{\text{middle}}$	$\llcorner\ bb\lrcorner\ \llcorner c\ \lrcorner$
Cas 4 :	$\llcorner\ \lrcorner\ \llcorner\ b\underline{c}c\ \lrcorner$	$\xrightarrow{\text{middle}}$	$\llcorner\ \lrcorner\ \llcorner\ bbc\ \lrcorner$
Cas 5 :	$\llcorner\underline{b}b\ \lrcorner\ \llcorner\ b\lrcorner$	$\xrightarrow{\text{bottom}}$	$\llcorner(b+1)b\ \lrcorner\ \llcorner\ b\lrcorner$
Cas 5' :	$\llcorner\underline{b}\lrcorner\ \llcorner b\ b\lrcorner$	$\xrightarrow{\text{bottom}}$	$\llcorner(b+1)b\lrcorner\ \llcorner\ b\lrcorner$
Cas 6 :	$\llcorner c\ \lrcorner\ \llcorner\ c\underline{b}\lrcorner$	$\xrightarrow{\text{top}}$	$\llcorner c\ \lrcorner\ \llcorner\ cc\lrcorner$
Cas 6' :	$\llcorner c\ c\lrcorner\ \llcorner\underline{b}\lrcorner$	$\xrightarrow{\text{top}}$	$\llcorner c\ c\lrcorner\ \llcorner c\lrcorner$

FIG. 3.1 – Les différentes réductions possibles

DÉMONSTRATION : On a ici besoin de considérer uniquement les cas 1 à 4 (c.f. FIG. 3.1) qui décrivent les changements effectués sur u_i et v_i selon la position de w_i qui va être affectée. On va montrer que si w_i est dans $M(a+m,a)$ alors w_{i+1} aussi.

cas 1 : $u_i = u'_i.(c+1)\underline{c}c.u''_i$ avec $c = a+l$, alors on pose
$u_{i+1} = u'_i.(c+1)(c+1)c.u''_i$ et $v_{i+1} = v_i$ et on a :
$u_{i+1} = (a+m)^{\lambda_{i,m}}...(a+l+1)^{\lambda_{i,l+1}+1}(a+l)^{\lambda_{i,l}-1}...(a+1)^{\lambda_{i,1}}a^{\lambda_{i,0}}$, avec $\lambda_{i,l-1} - 1 > 0$ (car on avait dans u_i deux c d'affilée avec $c = (a+l)$) donc $u_{i+1} \in M(a+m,a)$.

cas 2 : Si on a pu appliquer la règle c'est que $u_i = u'_i.\underline{(a+1)}a$, $v_i = a.v''_i$ alors on pose $u_{i+1} = u'_i.(a+1)(a+1)a$, et $v_{i+1} = v''_i$. On a donc
$u_{i+1} = (a+m)^{\lambda_{i,m}}(a+m-1)^{\lambda_{i,m-1}}...(a+1)^{\lambda_{i,1}+1}a \in M(a+m,a)$.

cas 3 : On a $u_i = u'_i.a$ et $v_i = \underline{c}c.v''_i$, et on pose $u_{i+1} = u'_i.aa$, $v_{i+1} = c.v''_i$. On a alors $u_{i+1} = (a+m)^{\lambda_{i,m}}(a+m-1)^{\lambda_{i,m-1}}...(a+1)^{\lambda_{i,1}}a^{\lambda_{i,0}+1} \in M(a+m,a)$

cas 4 : On a $v_i = v'_i.b\underline{c}c.v''i$. On pose $u_{i+1} = u_i \in M(a+m,a)$ et
$v_{i+1} = v'_i.bbc.v''i$

■

De plus, les cas 2 et 3 augmentent de un la longueur de la partie régulière (qui correspond aux u_i) ce qui tend, conformément à ce qui avait été annoncé dans l'idée de la preuve, à faire gagner au système de la régularité. Le cas 5' augmente aussi la longueur de la partie régulière, mais seulement lorsque celle-ci est composée d'une seule lettre.

Graphe et preuve de sa correction

Pour construire le graphe correspondant à ce système de réécriture, on a besoin de distinguer certaines des réductions middle que nous appellerons middle' :

ce sont celles qui correspondent au cas 2 lorsque le mot v_i est composé de la seule lettre c. Alors v_{i+1} est le mot vide (c.f. FIG 3.2).

Cas 2' : $\sqcup\_\_\_\_\_\_ \underline{bc}\sqcup \sqcup \underline{c}\sqcup \xrightarrow{\text{middle'}} \sqcup\_\_\_\_\_\_ \underline{bbc}\sqcup$

FIG. 3.2 – La réduction middle'

On va à présent construire le graphe correspondant aux suites de réductions possibles suivant les règles de $\mathcal{S}^{\alpha}_{Dij}$.

Le graphe FIG. 3.3 ne décrit pas tous les comportements possibles. En fait il faudrait faire une copie de la partie initiale pour chaque lettre $a \in \Sigma$ mais comme elles sont identiques à a près, on n'en met qu'une, par souci de lisibilité. De plus on a, pour toute boîte du type $M(l,a)\Sigma^+$ en fait une seule transition sortante étiquetée par middle'qui dépend des valeurs relatives de a et l :
- si $a = l$ on arrive dans une boîte du type $M(l,l)$
- si $a = l - 1$ on arrive dans une boîte du type $M(l, l - 1)$
- sinon on arrive dans la boîte étiquetée par $M(c,a)$ avec $c \neq a \neq c - 1$

D'après le Lemme 4, tous les cas possibles de réduction middle vérifient la propriété de clôture de $M(b,a)\Sigma^*$. Si de plus on ne considère pas les réductions middle', on reste bien dans $M(b,a)\Sigma^+$ comme décrit sur le graphe.

Lemme 5. *L'application d'une règle bottom à un mot de $M(b,a)\Sigma^+$ en fait un mot de $M(b+1,a)\Sigma^+$.*

DÉMONSTRATION : En effet, on est dans le cas 5 ou 5' de la FIG. 3.1. et on passe de $u_i = (a+m)^{\lambda_{i,m}}(a+m-1)^{\lambda_{i,m-1}}...(a+1)^{\lambda_{i,1}}a^{\lambda_{i,0}}$ avec $\lambda_{i,m} \geqslant 2$ et $a+m = b \mod k$
à $u_{i+1} = (a+m+1)(a+m)^{\lambda_{i,m}}(a+m-1)^{\lambda_{i,m-1}}...(a+1)^{\lambda_{i,1}}a^{\lambda_{i,0}} \in M(b+1,a)$, ou bien dans le cas 5' de $u_i = b$ 'a $u_{i+1} = (b+1)b$.

■

Pour les mêmes raisons, l'application d'une règle bottom à un mot de $M(c,c)$ en fait un mot de $M(c+1,c)$.

Quand à l'application des règles top, c'est à dire les cas 6 et 6' de la figure 3.1, on voit qu'elles n'affectent en rien la partie régulière d'un mot de $M(b,a)\Sigma^+$ car elles ne changent que la dernière lettre. Par contre, si on a un mot de $M(c+1,c)$, il devient après application d'une règle top un mot de $M(c+1,c+1)$, ce qui se montre facilement.

Des raisonnements qui précèdent, on en déduit le

Lemme 6. *Le graphe donné à la figure 3.3 est correct pour la description des exécutions associées au système de réécriture $\mathcal{S}_{Dij}$*

Pas de bouclage dans la partie initiale

Maintenant qu'on a vérifié la conformité du graphe au modèle, il reste à justifier la validité du "découpage" en deux parties.

Lemme 7. *On ne peut pas rester infiniment dans la partie initiale*

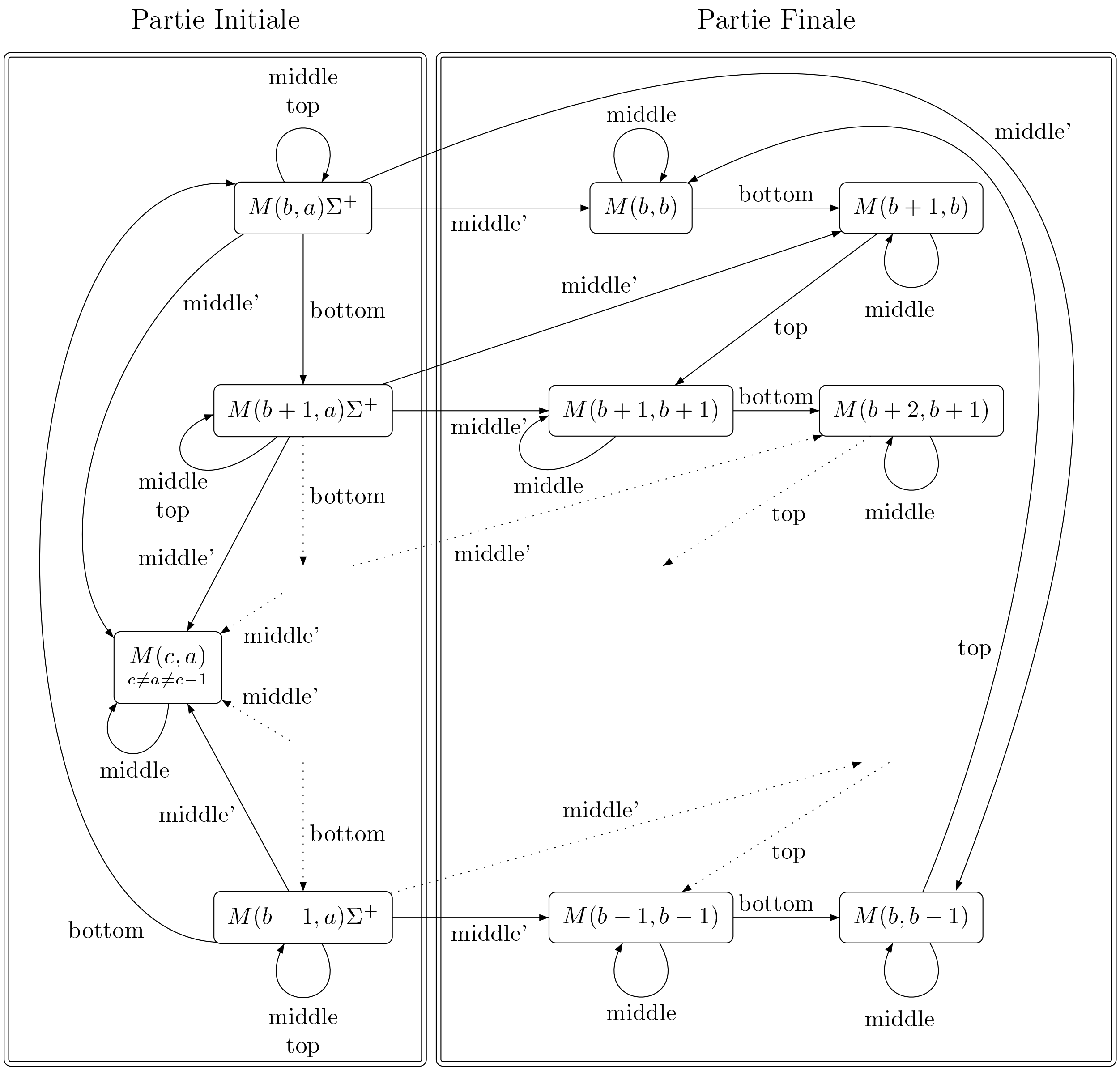

FIG. 3.3 – Un graphe de boîtes pour l'agorithme de Dijkstra à K états

DÉMONSTRATION : Pour des raisons d'équité, si u_i est de longueur $l_i < N$, la position l_i va être réduite par la suite, mettons que la première fois sera lors du passage de $u_{i'}$ à $u_{i'+1}$. Alors,

- soit entre temps la longueur de la partie régulière a augmenté (par exemple en appliquant une règle du cas 3), auquel cas si $l_{i'} = N$ on a fini, sinon on recommence le raisonnement pour $u_{i'}$,
- soit la position l_i est la dernière lettre de $u_{i'}$. Dans ce cas elle ne peut être réduite que par une règle décrite dans les cas 2 ou 5', ce qui fait que l'application de cette règle va augmenter la taille de $u_{i'}$.

Il va donc exister un indice $j > i$ tel que la longueur de u_j soit strictement supérieure a celle de u_i. En itérant ce raisonnement autant de fois que nécessaire (toujours en nombre fini), on montre qu'il existe un indice i_0 tel que $|u_{i_0 - 1}| < N$ et $|u_i| = N$.
On arrive soit dans un état du genre $M(c, c)$ ou $M(c+1, c)$ qui est dans la partie finale, soit dans l'état $M(c, a)$ avec $c \neq a \neq c - 1$. Si on est dans la partie finale c'est réglé, sinon, il faut montrer qu'il n'existe pas de suite infinie de réductions qui bouclent dans $M(c, a)$.
Soit $u = c.v.ba \in M(c, a)$. Si on applique middle, on est forcément dans le cas 1 de la Fig. 3.1 et comme on a vu dans le Lemme 4 la forme réduite est aussi dans $M(c, a)$. De plus, comme u est dans $M(c, a)$, on a $b = a$ ou $b = a + 1$ d'où $b \neq c$ et on ne peut pas appliquer de règle top. De plus comme $a \neq c$ on ne peut pas appliquer bottom. On reste donc dans $M(c, a)$ sans appliquer de réduction ni top ni bottom. Par équité, on ne peut donc pas effectuer de suite infinie de réductions.

■

Le fait qu'on reste dans la partie finale une fois qu'on y est entré se voit de manière évidente sur le graphe. Il reste à trouver les mots récurrents parmi tous ceux contenus dans les boîtes de la partie finale. Certains mots des boîtes de la partie finale ne sont en effet atteints qu'un nombre fini de fois.

Détermination du langage récurrent (preuve de $\mathcal{R} = \mathcal{Q}$)

Il reste à présent à montrer que le langage $\mathcal{Q}$ défini au début de cette partie est le langage récurrent $\mathcal{R}$.
On voit tout d'abord que toutes les boîtes font partie d'un même cycle. De plus pour des raisons d'équité, on va devoir effectuer une infinité de réductions bottom et donc ne pas boucler sur un seul état.

Lemme 8. $\mathcal{R} \subseteq \mathcal{Q}$

Démonstration : Soit u un mot de $\mathcal{R}$, alors il existe une exécution infinie le long de laquelle u est infiniment souvent répété. Comme on a montré qu'on ne pouvait pas rester dans la partie initiale, on va donc ne considérer que la fin de l'exécution. u est donc un mot de $M(b, b)$ ou $M(b + 1, b)$ pour un certain b.
Seulement, pour pouvoir appliquer bottom à un mot de $M(b, b)$, il faut que ce mot soit de la forme $u = bb.v$ avec $v \in H_b^n$ pour un certain entier n, et donc que la longueur du mot soit $|u| = N \geqslant nK + 2$. Or le long de l'exécution infinie, on va devoir appliquer bottom, et la longueur des mots ne change pas.
De plus les $b^+H_b^n$ et $(b+1)^+b^+H_b^n$ sont stables par application de règles middle, ce qui fait que u est soit dans $(b + 1)^+b^+H_b^n$ (donc dans $\mathcal{Q}_2$), soit dans $b^+H_b^n$ avec $|u| \geqslant nK + 2$, ce qui signifie que u n'est pas égal à $b((b-1)(b-2)...(b+1)b)^n$ et donc possède deux lettre consécutives égales (et appartient à $\mathcal{Q}_1$).

■

Remarque : Si $N \leqslant K + 1$, on a forcément $n = 0$ et $u \in b^+$. Si on applique ensuite une suite de reductions à ce mot, on obtient le langage récurrent qui est $\mathcal{Q} = \bigcup_{b \in \Sigma}(b^+ \cup (b + 1)^+b^*)$.

On vient donc de montrer que l'ensemble $\mathcal{R}$ des mots récurrents est inclus dans $\mathcal{Q}$. Pour achever la démonstration du Théorème 1, reste à montrer l'inclusion réciproque.

Lemme 9. $\mathcal{Q} \subseteq \mathcal{R}$

DÉMONSTRATION : Prenons un mot u de $\mathcal{Q}$ et construisons une suite de réductions qui passe une infinité de fois par ce mot. Pour cela on doit distinguer deux cas :

- Si $u \in \mathcal{Q}_1$, alors $u = b^{\lambda_0}(b-1)^{\lambda_1}...(b-m)^{\lambda_m}$ avec $m = 0 \bmod K$, tous les $\lambda_j \geqslant 1$ et tel qu'il existe $0 \leqslant i \leqslant m$ tel que $\lambda_i \geqslant 2$. On procède alors par étapes, que l'on va suivre en parallèle sur un exemple :

  1. On peut appliquer à u la règle $\langle \_(a-i+1)(a-i)(a-i)\_, \_(a-i+1)(a-i+1)(a-i)\_\rangle$ puis $\langle \_(a-i+2)(a-i+1)(a-i+1)\_, \_(a-i+2)(a-i+2)(a-i+1)\_\rangle$ et ainsi de suite. On fait ainsi se déplacer le coefficient $\geqslant 2$ jusqu'au début du mot.
     On obtient donc le mot $b^{\lambda_0}b(b-1)^{\lambda_1}...(b-i)^{\lambda_i-1}...(b-m)^{\lambda_m}$.

     $$03\underline{2}2100 \longrightarrow 0\underline{3}32100 \longrightarrow 0032100$$

  2. Puis, par une réduction bottom on obtient le mot $v = (b+1)b^{\lambda_0}(b-1)^{\lambda_1}...(b-i)^{\lambda_i-1}...b^{\lambda_m}$ (car on a $b-m = b \bmod K$)

     $$\underline{0}032100 \longrightarrow 1032100$$

  3. On effectue à présent des réductions associées à la règle $\langle \_(b+1)bb\_, \_(b+1)(b+1)b\_\rangle$ autant de fois que nécessaire pour n'avoir qu'un seul b à la fin du mot (il est possible qu'on n'ait pas à en faire du tout).
     On obtient le mot $(b+1)b^{\lambda_0}(b-1)^{\lambda_1}...(b-i)^{\lambda_i-1}...(b+1)^{\lambda_{m-1}+\lambda_m-1}b$

     $$1032\underline{1}\underline{0}0 \longrightarrow 1032110$$

  4. Ensuite on applique top ce qui donne $(b+1)b^{\lambda_0}(b-1)^{\lambda_1}...(b-i)^{\lambda_i-1}...(b+2)^{\lambda_{m-2}}(b+1)^{\lambda_{m-1}+\lambda_m}$

     $$103211\underline{0} \longrightarrow 1032111$$

  5. Enfin, on va appliquer la règle $\langle \_(b+2)(b+1)(b+1)\_, \_(b+2)(b+2)(b+1)\_\rangle$ jusqu'à obtenir le mot $(b+1)b^{\lambda_0}(b-1)^{\lambda_1}...(b-i)^{\lambda_i-1}...(b+2)^{\lambda_{m-2}+\lambda_{m-1}}(b+1)^{\lambda_m}$ et réitérer le processus avec la règle $\langle \_(b+3)(b+2)(b+2)\_, \_(b+3)(b+3)(b+2)\_\rangle$ et ainsi de suite, afin d'obtenir le mot $(b+1)^{\lambda_0}b^{\lambda_1}...(b-m+1)^{\lambda_m}$

     $$1032\underline{1}11 \longrightarrow 103\underline{2}211 \longrightarrow 1033211$$

  On obtient donc un mot qui est l'image de u par changement de chaque lettre par sa suivante dans l'alphabet Σ. Il suffit donc de réitérer cette opération K fois et on retombe sur le mot u. On a ainsi construit un cycle de réductions passant par le mot u et en l'itérant une infinité de fois, on met en évidence que u est récurrent.

- Si $u \in \mathcal{Q}_2$, on construit un cycle de manière similaire, mais en commençant par l'étape 3, puis on applique top comme en 4, on décale le doublement de lettre pour l'amener en première position ce qui serait une sorte de 5', puis on applique top comme en 1 et par des r'eductions middle on forme le mot image de celui de départ par un décalage dans l'alphabet (2'). Sur un exemple cela donne :

$$11021\underline{0}00 \xrightarrow{3} 110211\underline{0}0 \xrightarrow{3} 1102111\underline{0} \xrightarrow{4} 11021\underline{1}11 \xrightarrow{5'} 110\underline{2}2111$$

$$\xrightarrow{5'} 11\underline{0}02111 \xrightarrow{5'} \underline{1}1102111 \xrightarrow{1} 2\underline{1}102111 \xrightarrow{2'} 22102111$$

■

On a ainsi montré que l'ensemble des mots récurrents était exactement $\mathcal{Q}$, ce qui achève la démonstration du Théorème 1.

Pour démontrer qu'il y a auto-stabilisation, il suffit de montrer la clôture de $\mathcal{R}$ et le fait qu'il est inclus dans $\mathcal{P}$. Pour la clôture, cela se fait à partir de la clôture de $\bigcup_{b\in\Sigma}(M(b,b) \cup M(b+1,b))$ et de la conservation de la longueur des mots. La correction n'est vraie que quand on a $K+1 \geqslant N$, ce qui retrouve les résultats de Dijkstra et Hoepman.

Cette preuve permet ici de donner un ensemble légitime dans le cas où il y a auto-stabilisation relativement à l'ensemble $\mathcal{P}$ (*i.e.* dans le cas où $K+1 \geqslant N$), et de donner les états récurrents dans tous les cas.
On peut remarquer que pour $K+1 \geqslant N$, les ensembles

$$\mathcal{R} = (\bigcup_{a\in\Sigma}(a+1)^+a^+) \cup (\bigcup_{a\in\Sigma} aa^+) \text{ et}$$

$$\mathcal{P} = (\bigcup_{a\in\Sigma} aa^+) \cup (\bigcup_{a\in\Sigma}\bigcup_{\substack{b\in\Sigma\\ b\neq a}} a^+b^+)$$

sont légitimes. Pour $\mathcal{R}$ on vient de le montrer, et pour $\mathcal{P}$, c'est un sur-ensemble de $\mathcal{R}$ (inclus dans $\mathcal{P}$) et qui vérifie la propriété de clôture.

Chapitre 4

Version probabiliste de notre méthode

4.1 Adaptation du cadre précédent

4.1.1 Systèmes de réécriture probabilistes

On va maintenant s'intéresser à des systèmes de réécriture probabilistes. Ce qui va changer est qu'on va introduire dans les algorithmes des règles probabilistes. Ces règles sont du genre :

$$R_3 = \ \_u\_ \rightarrow \begin{cases} \_u_1\_ \ (p_1) \\ \_u_2\_ \ (p_2) \\ \vdots \\ \_u_n\_ \ (p_n) \end{cases}$$

où les p_i sont tous $0 < p_i < 1$ et leur somme est égale à 1.
Cela signifie qu'un mot du type $v'.u.v''$ peut être réduit suivant cette règle, et que la forme réduite a une probabilité p_1 d'être $v'.u_1.v''$, une probabilité p_2 d'être $v'.u_2.v''$ *etc.* Comme dans les parties précédentes, on considère principalement des règles à 3 lettres et qui n'en affectent qu'une. Si la lettre affectée dans tous les membres droits est la même, on peut écrire la règle de la manière habituelle, par exemple $\langle a\_bc, a\_b\gamma\rangle$. γ est alors un générateur aléatoire qui tire la lettre c_1 avec une probabilité p_1, la lettre c_2 avec une probabilité p_2, ... la lettre c_j avec une probabilité p_j.

Cette nouvelle définition implique que l'on ne peut pas choisir arbitrairement toutes les réductions que l'on va appliquer. Bien sûr entre deux réductions qui ne sont pas issues de la meme règle probabiliste, le choix est toujours arbitraire, mais au sein d'une même règle probabiliste, le choix est régi par les probabilités de chacun des sous-cas. Ainsi, si l'on applique une infinité de fois la règle R_3, la probabilité que l'on réduise toujours le mot de la même façon (par exemple $\_u_2\_$) est nulle. Les probabilités respectives de chaque membre droit vont imposer une certaine régularité dans le choix des réductions.

Si certaines notions définies dans la partie 2.1 s'appliquent directement au cadre probabiliste, d'autres par contre nécessitent quelques modifications.

4.1.2 Ce qui change

On va tout d'abord introduire la notion de probabilité sur les ensembles d'exécutions.
Soit Π un ensemble d'exécutions (infinies car il n'y a pas de blocages). La *probabilité* de l'ensemble Π est la probabilité, parmi toutes les exécutions, d'en emprunter une quelconque de Π.

La différence entre les deux cadres vient du fait que dans le cadre non probabiliste, la notion de convergence est absolue : un modèle converge vers un sous-ensemble de configurations si et seulement si toute exécution (infinie) atteint une configuration de ce sous-ensemble.
Dans le cadre probabiliste, la notion de convergence est différente : les "mauvaises" exécutions (celles qui ne vérifient pas les propriétés voulues) ne sont pas inexistantes, mais on a une probabilité nulle d'emprunter l'une d'elles. Cela s'exprime plus formellement dans la définition suivante :

Définition : On dit qu'un système de réécriture probabiliste *converge* vers un ensemble de configurations $\mathcal{C}$ s'il existe un ensemble d'exécutions de probabilité 1 tel que toute exécutions de cet ensemble atteint une configuration de $\mathcal{C}$.

Pour s'adapter à cette nouvelle forme de convergence, on introduit une nouvelle définition des mots récurrents :

Définition : Pour un système de réécriture probabiliste, un mot u est *récurrent* s'il existe un ensemble d'exécutions $\mathcal{C}$, de probabilité non nulle et tel que le long de toute exécution de $\mathcal{C}$, u est répété infiniment souvent.

4.2 Un algorithme probabiliste d'exclusion mutuelle : L'algorithme de Herman

4.2.1 Le cadre

Dans un cadre déterministe, on n'a auto-stabilisation dans l'algorithme de Dijkstra et pour $K = 3$ que si le nombre de processus est $N \leqslant 4$, cet exemple utilise les probabilités pour obtenir une propriété d'auto-stabilisation pour un N quelconque. Pour le cas étudié, les processus considérés sont numérotés de 0 à $N-1$. Ces processus possèdent trois états possibles : 0, 1 et 2.
Une configuration se représente alors par un mot de N lettres sur l'alphabet $\{0, 1, 2\}$.
Les règles d'évolution du système sont les suivantes :

$$\begin{array}{rrcl} 0 < i < N : & x_{i-1} \neq x_i & \Longrightarrow & x_i := x_{i-1} \\ & x_{N-1} = x_0 & \Longrightarrow & x_0 := \gamma \end{array} \tag{4.1}$$

ce qui correspond au système de réécriture :

$$\mathcal{S}_{Her} = \left\{ \begin{array}{cc} \langle \_ab\_, \_aa\_ \rangle & \text{(middle et top)} \\ \langle a\_a, \gamma\_a \rangle & \text{(bottom)} \end{array} \right.$$

où γ est un générateur aléatoire de nombres qui tire de façon équiprobable les valeurs 0, 1 ou 2.

On garde dans cet exemple le même ensemble $\mathcal{P}$ que pour l'algorithme de Dijkstra classique, à savoir qu'une configuration u est dans $\mathcal{P}$ ssi elle ne possède qu'un seul jeton. L'ensemble $\mathcal{P}$ est alors formé des mots du type aa^+ ou a^+b^+, où a et b sont des lettres distinctes.

On veut donc montrer que l'algorithme ci-dessus est auto-stabilisant, mais cette fois-ci de manière probabiliste. C'est à dire qu'il existe un ensemble $\mathcal{L}$ tel que toute suite infinie quelconque de réductions mène avec une probabilité 1 et en un temps fini à une configuration légitime, et qu'ensuite on reste dans l'ensemble $\mathcal{L}$.

Une preuve de ce résultat a été donnée par Herman dans [Her92], preuve qui va être brièvement décrite dans la section suivante. La preuve de la partie 4.2.3 aborde le problème sous un autre angle et utilise la méthode décrite dans la partie 2.2 pour calculer le langage récurrent.

On va donc procéder comme annoncé dans la partie 2.2 et découper le graphe en deux parties. La différence est qu'on ne va pas prouver que l'on ne peut pas rester indéfiniment dans la partie initiale mais que la probabilité de le faire est nulle.

Avant cela, on va donner une ébauche de la preuve faite par Herman.

4.2.2 Une idée de la preuve de Herman

Dans sa preuve, donnée dans [Her92], Herman commence tout d'abord par montrer que l'ensemble des configurations à un seul jeton ($\mathcal{P}$) est clos pour l'application de réductions et qu'il ne peut y avoir de blocages.

Ensuite et en utilisant une mesure décroissante le long de toute suite de réductions n'affectant pas le processus 0, il montre (comme dans la preuve de Dijkstra non probabiliste présentée par [Tel94]) que la réduction bottom est appliquée une infinité de fois.
Puis, il introduit une mesure du nombre de fois où une position est affectée le long d'une suite finie de réductions, ainsi que le lien entre le nombre d'affectations d'une position et celui des positions voisines, avec la particularité que dans cet algorithme, il est possible d'effectuer une réduction sans affecter aucune position [1].

Il continue avec un raisonnement probabiliste montrant qu'avec une probabilité de 1, on va arriver en un temps fini à une configuration où aucun processus n'est dans l'état 2 (le choix de 2 étant arbitraire).

Enfin, la fin de la preuve est quasiment la même que pour celle décrite dans la partie 1.3, à savoir que, avec une probabilité 1, le processus 0 va revenir en un temps fini dans l'état 2, et ne va pouvoir le quitter que lorsque tous les processus seront dans le même état que lui. Or comme le processus 0 change infiniment souvent d'état, cette configuration va être effectivement atteinte.

Comme la configuration 2^N est dans $\mathcal{P}$, on a montré la convergence (probabiliste). Comme la correction et la clôture sont aussi vérifiées, on a bien auto-stabilisation.

[1] Ceci arrive lorsque l'on effectue une réduction bottom suivant la règle $a\_a \rightarrow \gamma\_a$ et que γ prend la valeur a

4.2.3 Notre preuve

Comme dans la partie 1.2, on peut noter qu'il n'existe pas de "deadlocks" (ou blocages), c'est à dire de configurations pour lesquelles aucune transition n'est tirable. On peut, pour le montrer, utiliser exactement les mêmes arguments car les membres gauches des règles sont les mêmes dans les deux cas.

Le nombre α de sites actifs est donc toujours strictement positif. On peut ainsi construire une suite infinie de réductions partant de n'importe quelle configuration et donc prolonger toute suite finie en une suite infinie.

Règles conservant le nombre de sites actifs

On va à présent étudier l'évolution du nombre α de sites actifs le long d'une suite de transitions. Tout d'abord pour une réduction, plusieurs cas peuvent se présenter :

$$\begin{array}{clcll}
\bullet & \underset{a\neq b}{u.abc.v} & \longrightarrow & u.aac.v & \text{si } b=c,\ \alpha \text{ reste stable} \\
 & & & & \text{si } b\neq c,\ \alpha := \alpha-1 \\
\bullet & \underset{a\neq b}{c.u.ab} & \longrightarrow & c.u.aa & \text{si } b=c,\ \alpha := \alpha-2 \\
 & & & & \text{si } b\neq c \text{ et } a=c,\ \alpha \text{ reste stable} \\
 & & & & \text{si } b\neq c \text{ et } a\neq c,\ \alpha := \alpha-1 \\
\bullet & cb.u.c & \longrightarrow & \gamma b.u.c & \text{si } \gamma=c,\ \alpha \text{ reste stable} \\
 & & & & \text{si } b=c \text{ et } \gamma\neq b,\ \alpha \text{ reste stable} \\
 & & & & \text{si } b\neq c \text{ et } c\neq\gamma\neq b,\ \alpha := \alpha-1 \\
 & & & & \text{si } b\neq c \text{ et } \gamma=b,\ \alpha := \alpha-2
\end{array}$$

On a donc α qui ne peut pas croître lors d'une réduction, donc lors d'une suite de réductions.
D'après la Proposition 1, on peut s'intéresser uniquement aux réductions qui conservent α et qui sont :

$$\begin{array}{rcll}
aa.u.a & \longrightarrow & ba.u.a & \left.\begin{array}{l}\text{bottom}_1\\ \text{bottom}_2\end{array}\right\}\text{bottom} \\
ac.u.a & \longrightarrow & ac.u.a & \\
u.abb.v & \longrightarrow & u.aab.v & \text{middle} \\
a.uab & \longrightarrow & a.u.aa & \text{top}
\end{array} \tag{4.2}$$

Où a, b et c sont des lettres de Σ avec $a \neq b$ et u, v sont des mots de Σ^*.

Graphe et explications

Les configurations qui ne sont pas dans $\mathcal{P}$ sont celles qui ont au moins deux sites actifs (car on a vu qu'il était impossible de ne pas en avoir du tout). Elles entrent dans l'un des types suivants :

$$\delta\Sigma^* \underset{\neq\delta}{\eta}\, \delta^*\delta,\ \rho\Sigma^* \underset{\substack{\neq\delta\\ \neq\rho}}{\eta}\, \delta^*\delta \text{ ou } \delta\Sigma^* \underset{\neq\delta}{\eta}\, \delta^* \underset{\neq\beta}{\delta}\, \beta^*\beta$$

En effet : soit les sites actifs sont le numéro 0 et un autre, auquel cas on entre dans la deuxième catégorie, soit ce sont deux sites différents de 0 auquel cas on est dans la première ou la troisième catégorie.

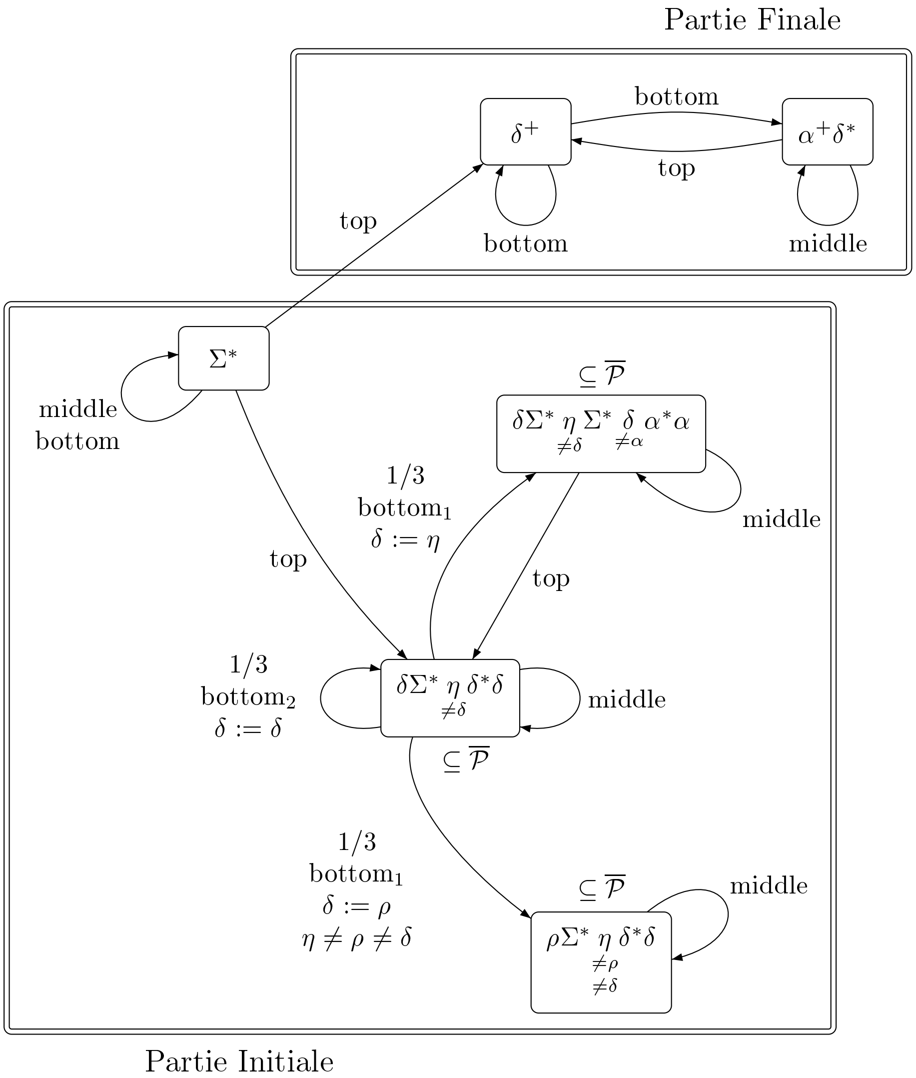

FIG. 4.1 – Graphe pour l'algorithme de Herman

On peut construire un graphe représentant les transitions possibles FIG. 4.1.

Toute suite de réductions part d'un mot de Σ^*, et ce qui nous intéresse c'est la première fois où l'on applique top. On a alors un mot du type $u = \delta.v.\delta\delta$ et on peut distinguer deux cas :

- Si $v \in \delta^*$ alors u est dans le langage de la partie finale ($=\mathcal{P}$).
- S'il existe une lettre $\eta \in \Sigma$ qui est différente de δ et qui apparaît dans v, alors on reste dans la partie initiale et on entre dans le sous-graphe qui contient les mots qui ne sont pas dans $\mathcal{P}$.

Ensuite, à partir d'un mot de $\overline{\mathcal{P}}$ (complémentaire de $\mathcal{P}$) et en appliquant une réduction conservant α, on reste dans un des états de $\overline{\mathcal{P}}$.
Ce fait est assez facile à vérifier sur le graphe, sauf peut-être pour la réduction

étiquetée par $\delta := \eta$, mais on tombe bien dans la boîte étiquetée par $\delta\Sigma^*\eta\Sigma^*\delta\alpha^*\alpha$ car cette réduction n'est en fait réalisable que si l'on part d'une configuration du type $\delta\mu\Sigma^*\eta\delta^*\delta$, avec η et μ différents de δ.
On arrive donc dans la configuration $\eta\mu\Sigma^*\eta\delta^*\delta$ qui est bien dans l'état désiré.

Clôture de la partie finale

On montre que l'ensemble $\mathcal{P}$ est stable pour le système de transitions considéré (*i.e.* stable par application d'une réduction associée à une règle de $\mathcal{S}_{Her}$). En effet, les mots de $\mathcal{P}$ sont de deux types : soit a^N (le site actif est alors celui d'indice 0), soit $a^p b^{N-p}$ avec $a \neq b$ et $0 < p < N$ (auquel cas le site actif est le $p^{\text{ème}}$). Dans les deux cas, en tirant une transition, on obtient un mot de $\mathcal{P}$ ($a^{p+1}b^{N-p-1}$ ou γa^{N-1}).

Non bouclage dans la partie initiale

On doit montrer qu'il n'existe pas de suite de réductions infinie qui reste dans la partie initiale. Par l'absurde, supposons qu'il existe un chemin infini qui ne visite jamais un état de $\mathcal{P}$. Il se termine par un suffixe infini qui n'utilise que des réductions conservant α, donc que des réductions du sous-graphe de $\overline{\mathcal{P}}$ que l'on notera G'.

On peut montrer, grâce à un raisonnement similaire à celui de la Proposition 2, que l'ensemble de règles conservant α est équitable.

Si on a un chemin infini dans G', cela signifie que :

- Soit on visite à un moment l'état étiqueté par $\rho\Sigma^* \underset{\substack{\neq\delta \\ \neq\rho}}{\eta}\, \delta^*\delta$, auquel cas on y reste infiniment, ce qui est impossible étant donné qu'on ne peut tirer qu'un nombre fini de fois de rang une transition middle (à cause de l'équité).
- Soit on boucle sur les deux premiers états. Comme on ne peut pas rester bloqué dans le premier (toujours par équité), on passe une infinité de fois par le deuxième, et on tire une infinité de fois une transition probabiliste sans jamais tirer la transition étiquetée par $\delta := \rho$ qui a pourtant une probabilité $1/3$. La probabilité de réaliser cela est nulle.

L'ensemble des exécutions infinies qui arrivent dans une configuration de la partie finale a une probabilité 1, ce qui achève la démonstration de non bouclage dans la partie initiale.

Démonstration de $\mathcal{R} = \mathcal{P}$

Comme on a montré le non bouclage dans la partie initiale, on a donc déjà $\mathcal{R} \subseteq \mathcal{P}$. Reste à prouver l'inclusion réciproque.

On pourrait croire que cela se fait très simplement car tout mot de $\mathcal{P}$ de longueur N est du type $a^l b^{N-l}$ (avec $0 \leqslant l < N$ et $a \neq b$) et donc récurrent car infiniment répété le long du cycle décrit à la figure 4.2.

Seulement, pour un algorithme probabiliste la définition de mot récurrent est plus restrictive, et c'est heureux car sinon on pourrait montrer que des mots du type $\delta\Sigma^* \underset{\neq\delta}{\eta}\, \delta^*\delta$ sont récurrents alors qu'ils ne sont même pas dans l'ensemble

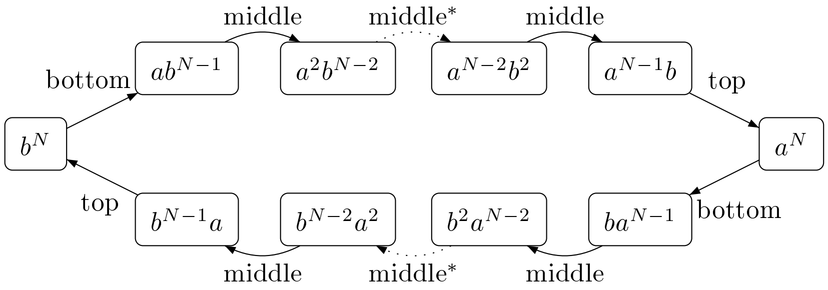

FIG. 4.2 – Une suite de réductions infinie

légitime. On va devoir montrer une propriété plus forte, c'est à dire que tout mot de $\mathcal{P}$ a une probabilité non nulle d'être infiniment répété le long d'une exécution.

Proposition 3. *Pour cet algorithme, tout mot de $\mathcal{P}$ a une probabilité 1 d'être infiniment répété le long d'une exécution.*

DÉMONSTRATION : Pour le montrer, on va détailler la partie finale du graphe 4.1. On va montrer plus précisément que tout mot de longueur N vérifie la proposition. En faisant varier N, on aura donc montré ce que l'on voulait.

Pour que le graphe soit lisible, on va regrouper certains mots dans un seul état. On va en effet désigner par $I(a,b)$ le sous graphe (que l'on peut voir d'ailleurs dans la FIG. 4.2) qui décrit la suite de réductions
$ab^{N-1} \to a^2b^{N-2} \to ... \to a^{N-2}b^2 \to a^{N-1}b$, suite de $N-2$ réductions middle consécutives. On peut alors construire le graphe présenté à la figure 4.3

Lorsqu'une exécution visite un sous-graphe du type $I(a,b)$, elle passe par chacun des mots qui composent les états de $I(a,b)$. Il suffit donc de montrer que toute exécution infinie qui se termine par une exécution de ce graphe (*i.e.* toutes les exécutions possibles sauf un ensemble de probabilité 0) visite infiniment souvent chacun des états de ce graphe.
On peut remarquer tout d'abord que si tous les états de la forme $I(a,b)$ sont infiniments souvent visités, alors les autres (de la forme a^N) le sont aussi.

D'autre part, la probabilité que (par exemple) $I(0,1)$ soit visité un nombre fini de fois sachant que 1^N est infiniment souvent visité est nulle, car cela reviendrait a ne jamais appliquer la réduction pour $\gamma = 0$ alors qu'elle a une probabilité 1/3 d'être réalisée. Donc avec une probabilité 1 soit ils sont tous les deux visités infiniment souvent, soit tous les deux finiment souvent.

On va raisonner par l'absurde :
Si on est dans le deuxième cas, alors on est sûr que l'état $I(1,2)$ est visité un nombre fini de fois (car la seule transition sortante de $I(1,2)$ mène dans 1^N et qu'on ne peut appliquer que $N-2$ réductions au sein de $I(1,2)$).
Alors, par le même raisonnement que ci-dessus, avec une probabilité 1, l'état 2^N n'est visité qu'un nombre fini de fois, ce qui entraîne que l'état $I(2,0)$ aussi. En appliquant une dernière fois le même raisonnement, on en déduit que l'état 0^N est lui aussi visité un nombre fini de fois avec une probabilité 1. On en déduit donc qu'avec une probabilité 1, aucun des états du type a^N n'est infiniment

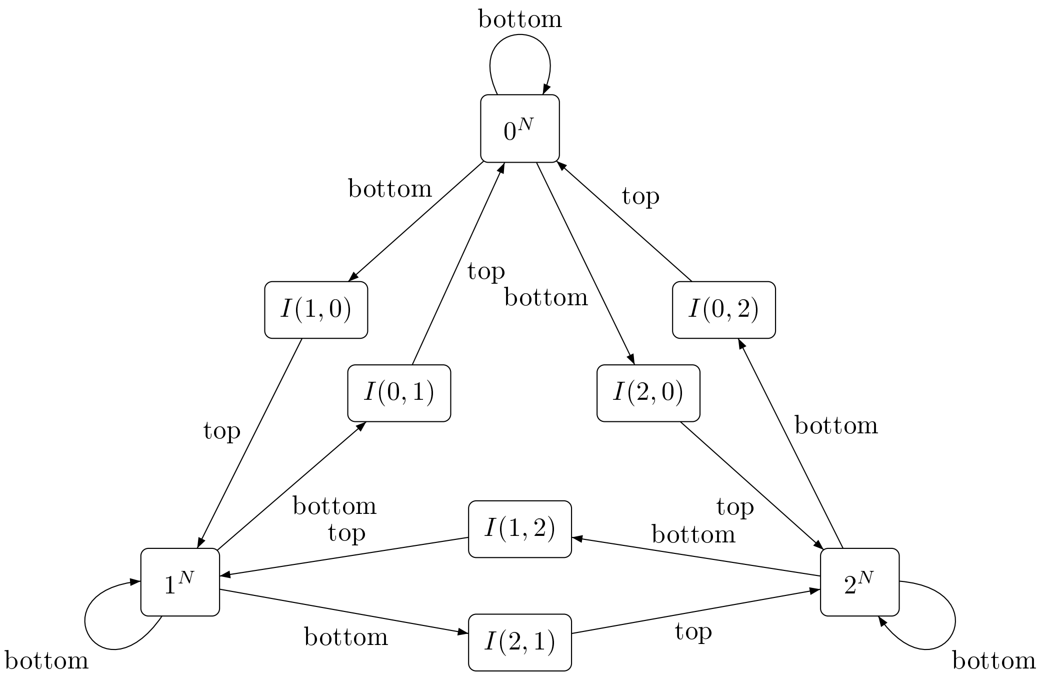

FIG. 4.3 – Les exécutions de mots de N lettres récurrents

visité, ce qui est impossible car on ne peut rester infiniment dans un état du type $I(b, c)$.

Cela entraîne que l'état $I(0, 1)$ est infiniment souvent visité. Comme tous les états du type $I(b, c)$ jouent un rôle identique, ils sont tous infiniment souvent répétés et les a^N aussi, et ceci avec une probabilité de 1, ce qui achève la démonstration de la proposition.

■

Tous les mots de $\mathcal{P}$ sont donc dans $\mathcal{R}$ ce qui nous donne donc la deuxième inclusion et on a $\mathcal{P} = \mathcal{R}$, ce qui achève la détermination du langage récurrent.

Pour prouver l'auto-stabilisation, il reste à prouver la correction (qui est évidente car $\mathcal{R} =\mathcal{P}$) et la clôture (qui a déja été démontrée car $\mathcal{R}$ est le langage des étiquettes des états de la partie finale). L'algorithme de Herman est auto-satabilisant relativement à l'ensemble $\mathcal{P}$.

Cet algorithme illustre l'apport de l'ajout de probabilités dans les algorithmes distribués. En effet l'algorithme initial n'a été que légèrement modifié, en ajoutant des probabilités sur une des règles. Ce simple ajout a permis d'avoir convergence vers un système d'exclusion mutuelle, et ceci pour seulement trois états et un nombre quelconque N de processus. L'analogue non probabiliste nécessitait pour sa part au moins $N-1$ états pour obtenir le même résultat.

Conclusion

La méthode présentée dans ce mémoire introduit, grâce à la notion d'états récurrents, un angle nouveau pour l'étude des comportements de chaînes de processus.
Elle permet de démontrer la correction d'algorithmes classiques d'auto-stabilisation comme l'algorithme de Dijkstra à K états, et ce de manière originale. D'autre part, elle s'avère assez flexible pour traiter sans trop de changements le cas probabiliste, comme on a pu le voir dans le chapitre 4.
L'utilisation de graphes la rend également plus visuelle que les preuves habituelles d'auto-stabilisation qui utilisent des calculs sur une norme.

Perspectives

La suite de ce travail aura pour but d'appliquer cette méthode à des algorithmes non seulement d'auto-stabilisation, mais aussi de détection de terminaison comme celui de [DFvG83]. Pour certains algorithmes, il est simple de détecter que l'on est dans une configuration bloquante, par exemple si les processus sont dans un état terminal. Cependant dans d'autres cas il est nécessaire d'écrire un nouvel algorithme qui permette de la faire (pour détecter par exemple que tous les processus sont en attente d'un changement de leur voisin qui n'arrivera pas).
On dit qu'une configuration vérifie *term* si aucune règle d'évolution ne peut lui être appliquée. Un algorithme de détection de terminaison doit alors vérifier les trois propriétés suivantes :

(i) Il ne doit pas interférer avec l'algorithme initial

(ii) Si *term* est vérifiée, alors l'algorithme doit le détecter et mettre les processus dans un état terminal en un nombre fini de pas (vivacité)

(iii) S'il s'arrête, alors c'est que *term* est vérifié (sûreté)

On veut s'intéresser à la propriété de vivacité, qui peut être exprimée en terme de langage récurrent. En effet, le fait que toute exécution qui satisfie la propriété *term* à un moment est finie est équivalent à ce qu'aucun état vérifiant term ne soit dans le langage récurrent.
Ce lien nous permettra d'étudier grâce à notre méthode l'algorithme de [DFvG83].

D'autre part, il conviendrait d'étudier plus en profondeur le lien avec les méthodes classiques d'auto-stabilisation. Cela permettrait de déterminer dans quelle mesure notre méthode en est différente, et si la nouveauté réside dans le fond de la preuve ou dans une présentation nouvelle et plus simple à partir des mêmes arguments de base (notamment l'équité).

Bibliographie

[BBF99] J. Beauquier, B. Berard, and L. Fribourg. A new rewrite method for convergence of self-stabilizing systems. In *Proc. of the 13th International Conference on Distributed Computing (DISC'99)*, volume 1693 of *LNCS*, pages 240–253, 1999.

[Del95] Sylvie Delaët. *Auto-stabilisation : Modèle et applications à l'exclusion mutuelle*. PhD thesis, Université de Paris Sud, 1995.

[DFvG83] E.W. Dijkstra, W.H.J. Feijen, and A.J.M. van Gasteren. Derivation of a termination detection algorithm for distributed computations. *Information Processing Letters*, 16 :217–219, 1983.

[Dij74] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17 :643–644, 1974.

[Gho93] Sukumar Ghosh. An alternative solution to a problem on self-stabilization. *TOPLAS*, 15 :735–742, 1993.

[Her92] Ted Herman. Self stabilization : Randomness to reduce space. *Distributed Computing*, 6 :95–98, 1992.

[Hoe99] Jaap-Henk Hoepman. Self-stabilizing mutual exclusion on a ring, even if $K = N^*$. *cs.DC/9909013*, 1999.

[JN00] Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite state systems. In *Proc. of the Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1785 of *LNCS*, pages 220–234, 2000.

[LHR97] David Lesens, Nicolas Halbwachs, and Pascal Raymond. Automatic verification of parameterized linear networks of processes. In *ACM*, pages 346–357, 1997.

[Tel94] Gerard Tel. *Introduction to distributed algorithms*. Cambridge University Press, 1994.

Annexe A

Langages récurrents - Ensembles légitimes

Le langage récurrent est, contrairement à l'ensemble légitime, déterminé de manière unique : en effet il se compose de tous les mots récurrents et uniquement de ceux-là. Au contraire, on peut avoir plusieurs ensembles légitimes pour un même système, comme on peut le voir dans la partie 3.
En effet, pour un même sous-ensemble $\mathcal{P}$ de configurations, on peut avoir plusieurs ensembles légitimes. Par exemple, si on en trouve un, mettons $\mathcal{L}$ et qu'un sur-ensemble $\mathcal{L}$' de $\mathcal{L}$ vérifie et la spécification et la propriété de clôture, alors $\mathcal{L}$' est lui aussi légitime.

Il y a cependant un lien entre les deux. En effet :

Proposition 4. *Le langage récurrent est inclus dans tout ensemble légitime, dès lors qu'un tel ensemble légitime existe.*

DÉMONSTRATION : Dans le cas non probabiliste, si un mot u est récurrent, cela signifie qu'il existe une suite de réductions $\Lambda = w_0 \to w_1 \to ...w_n \to ...$ le long de laquelle le mot u est infiniment répété. Or si $\mathcal{L}$ est un ensemble légitime, alors par définition il existe un indice i tel que pour tout $j \geqslant i$ on ait $w_j \in \mathcal{L}$. Comme u est répété une infinité de fois, on a bien un indice $i_0 \geqslant i$ tel que $w_{i_0} = u$ et donc $u \in \mathcal{L}$.
Dans le cas probabiliste, on impose de plus que l'ensemble des chemins le long desquels u est répété ait une probabilité non nulle d'être emprunté. Ainsi la probabilité qu'un chemin converge vers $\mathcal{L}$ tout en répétant infiniment le mot u est non nulle, et on est sûr qu'un tel chemin existe. La suite du raisonnement est identique à celle du cas non probabiliste.

■

Si on affaiblissait la notion de récurrent dans le cas probabiliste, on aurait la possibilité d'avoir une exécution (de probabilité nulle) qui répète infiniment un mot qui ne soit pas dans un ensemble légitime, comme on peut le voir sur le graphe 4.1 de la partie 4.2.3.

En particulier, si l'algorithme est auto-stabilisant, cela implique que le langage récurrent est inclus dans l'ensemble $\mathcal{P}$, ce qui nous donne une condition nécessaire pour l'auto-stabilisation.

Proposition 5. *Si le langage récurrent n'est pas inclus dans $\mathcal{P}$, alors le système n'est pas auto-stabilisant pour $\mathcal{P}$.*

Pour obtenir un ensemble légitime à partir du langage récurrent, il suffit de trouver un sur-ensemble de $\mathcal{R}$ inclus dans $\mathcal{P}$ et qui soit clos pour le système de réécriture $\mathcal{S}$.

Par contre, le langage des mots récurrents n'est pas forcément un ensemble légitime. Il vérifie la propriété de convergence mais pas toujours la propriété de clôture (c.f FIG. A.1).

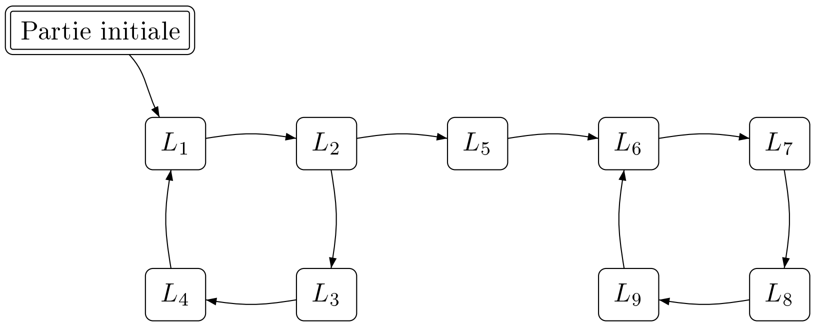

FIG. A.1 – Langage récurrent non clos

En effet, dans ce graphe, si les états 1 à 9 représentent des configurations (donc chacune un mot), $(1, 2, 3, 4)$ et $(6, 7, 8, 9)$ sont des cycles, donc peuvent faire partie du langage récurrent. Dans la suite on va supposer que c'est le cas. La configuration 5 ne fait quant à elle pas partie du langage récurrent car elle est atteinte au plus une fois le long d'une exécution infinie. Le langage récurrent ne peut donc pas être légitime dans ce cas car il n'est pas clos.

Annexe B

Une condition suffisante pour que $\mathcal{R}$ soit légitime : la non ambiguïté

On dit qu'un ensemble de règles est *ambigu* pour un ensemble de configurations $\mathcal{Q}$ s'in existe une configuration de $\mathcal{Q}$ à laquelle on peut appliquer au moins deux règles de réécriture distinctes.

Même si la non ambiguïté n'est pas nécessaire pour utiliser notre méthode, on la décrit ici car elle est vérifiée dans les exemples considérés dans le mémoire ainsi que dans d'autres exemples de la littérature, notamment des exemples d'exclusion mutuelle. D'autre part, cette hypothèse assure que le langage récurrent est légitime dès lors qu'il est inclus dans $\mathcal{P}$.

Non ambiguïté dans le cas non probabiliste

Si l'on empêche que deux réductions distinctes puissent s'effectuer sur un même mot récurrent, alors on voit bien que l'on ne peut plus avoir le cas de la figure A.1, où deux réductions au choix peuvent êtres faites sur L_2.
La non ambiguïté est ici une notion très forte, car elle suppos e à la fois que tout mot récurrent ne possède qu'un seul site atif, et que l'on ne peut pas appliquer deux règles différentes au même site actif.

Si l'on suppose la non ambiguïté de $\mathcal{S}$ pour $\mathcal{R}$, on va pouvoir montrer que le langage $\mathcal{R}$ des mots récurrents est clos par réduction.
En effet, si un mot u appartient à $\mathcal{R}$, alors il existe une exécution infinie $\Lambda = w_0, w_1, ..., w_n, ...$ telle qu'il existe une infinité d'indices $i_0, i_1, ...i_k, ...$ vérifiant $w_{i_k} = u$. Alors, comme on ne peut effectuer qu'une seule réduction sur u, mettons $u \to v$, on a pour chaque indice $i_k + 1$ $w_{i_k+1} = v$ et ainsi v est dans $\mathcal{R}$ qui est donc clos.

Il est vrai que cette hypothèse est très réductrice, elle permet cependant de trouver plus facilement un langage légitime car $\mathcal{R}$ en est un si et seulement si le système est auto-stabilisant, et le système est auto-stabilisant si et seulement si toute configuration de $\mathcal{R}$ est également dans $\mathcal{P}$.

Par exemple, pour l'algorithme de Dijkstra, la non ambiguïté est vérifiée pour $K + 1 \geqslant N$, et elle garantit que le langage $\mathcal{R}$ qui est alors $(\bigcup_{a\in\Sigma}(a+1)^+a^+) \cup (\bigcup_{a\in\Sigma} aa^+)$ est légitime, vu qu'il est inclus dans $\mathcal{P}$.

Non ambiguïté probabiliste

Dans le cas probabiliste, il y a un problème supplémentaire, car une réduction probabiliste introduit à elle seule de l'ambiguïté (comme dans le système 4.1).

Seulement, si ces règles sont la seule source d'ambiguïté, le langage récurrent est encore clos par application de réductions. En effet, si un mot u est récurrent et a une probabilité p d'être infiniment répété le long d'une exécution ($p > 0$), et si on a une règle probabiliste qui fait qu'on va effectuer la réduction $u \to v$ avec une probabilité p' non nulle, alors v aura une probabilité non nulle d'être infiniment répété.

Dans ce cas, on parle de non ambiguïté probabiliste.