Thèse

présentée à

L'Université Paris 7 Denis Diderot

pour obtenir le titre de

Docteur de l'Université Paris 7 Denis Diderot

Spécialité

Informatique fondamentale

par

Olivier Heen

Sujet de la thèse

Économie de ressources sur automates cellulaires

Devant être soutenue devant le jury composé de :

Jean Éric	Pin	Président
Serge	Grigorieff	Directeur
Jacques	Mazoyer	Rapporteur
Jean Paul	Allouche	Rapporteur
Maurice	Margenstern	Rapporteur
Danièle	Beauquier	

À Valérie

Je tiens à remercier tout particulièrement

- M Jean Éric Pin pour avoir accepté de présider ce jury et pour l'intérêt qu'il a porté à mon travail en diverses occasions.
- M Serge Grigorieff pour « plus de raisons qu'il n'est possible d'en donner ici, » citons néanmoins son efficacité en tant que de directeur de thèse et les cours brillants qu'il m'a dispensé dans le cadre du D.E.A. d'informatique fondamentale.
- M Jacques Mazoyer pour avoir accepté de raporter ce travail et pour sa bienveillance dès mes débuts dans le domaine.
- M Jean Paul Allouche pour avoir accepté de raporter ce travail avec célérité et pour son enthousiasme communicatif.
- M Maurice Margenstern pour avoir accepté de raporter ce travail. Ses remarques ont contribué à une meilleure rédaction.
- Mme Danièle Beauquier pour avoir accepté de participer au jury et pour m'avoir donné la parole en plusieurs occasions au séminaire d'informatique fondamentale de l'université Paris-Val-de-Marne.

Table des matières

Chapitre 1

Introduction

Imaginés dès 1950 par J. von Neumann, les automates cellulaires forment une classe originale de machines parallèles :

– machines abstraites, utiles pour l'étude de problèmes parallèles et des fondements du parallélisme ;

– machines concrètes, dites massivement parallèles, pour la modélisation de problèmes physiques et le calcul intensif.

Une grande partie du succès des automates cellulaires provient probablement de leur aptitude à modéliser naturellement des problèmes réputés complexes (écoulement fluide, traitement d'image, calculs réversibles, sociétés d'insectes) ...

D'un point de vue plus informatique, un certain nombre d'algorithmes séquentiels simples (mais inefficaces) deviennent optimaux sur automates cellulaires (multiplication « naïve » d'entiers, tri à bulles, génération de nombres premiers par le crible d'Eratosthène).

De plus, les automates cellulaires facilitent la compréhension de certains problèmes du parallélisme en mettant en évidence les déplacements d'information dans l'espace et dans le temps, qui peuvent alors être interprétés géométriquement.

Il est à noter qu'un grand nombre de questions sur automates cellulaires quelconques s'expriment directement ou se réduisent en termes de questions sur automates cellulaires unidimensionnels. C'est le cas pour les problèmes de synchronisation, dont la plupart des solutions sont des extensions du cas à une dimension. Certains problèmes échappent toutefois à cette règle et sont qualitativement plus difficiles quand la dimension augmente (la réversibilité par exemple n'est décidable qu'en dimension 1). Les problèmes résolus dans cette thèse le sont en dimension un et se généralisent souvent facilement aux dimensions supérieures.

Comme il existe une théorie des machines de Turing, il existe un début de théorie des automates cellulaires. En effet, si les premiers résultats de complexité ont d'abord été établis par simulation ou réduction des résultats classiques sur machines de Turing, on en trouve aujourd'hui des preuves directes, c'est-à-dire impliquant *seulement* des automates cellulaires, ce qui fait apparaître les automates cellulaires comme un véritable modèle du calcul parallèle (en particulier, l'universalité des automates cellulaires est une propriété intrinsèque). Une grande partie du travail de cette thèse vise l'obtention de preuves intrinsèques et efficaces des théorèmes d'accélération (essentiels puisqu'ils permettent d'obtenir plus rapidement le résultat d'un calcul). On trouve déjà cette préoccupation dans les travaux de J. Mazoyer et N. Reimen par exemple.

Une liste de références couvrant cet aspect important est proposée à la fin de cette introduction.

1 Motivations

1.1 Économie

Comme le titre de la thèse l'indique, la motivation initiale est l'économie de ressources. Par *ressources* nous entendons principalement :

- le temps nécessaire à la réalisation d'une tâche,

- le nombre d'états nécessaires à la réalisation d'une tâche.

Dans ce cadre, il est naturel de se préoccuper d'accélération (d'une constante ou linéaire) et, si possible, de reformuler les théorèmes déjà connus dans ce domaine en minimisant le nombre d'états requis (et d'autres paramètres secondaires comme la densité ou la présence d'une synchronisation). En particulier, le théorème d'accélération d'une constante pour les calculateurs n'existait pas sous une forme exploitable en pratique ; nous remédions à ce problème dans les chapitres 4 et surtout 5.

Toujours dans l'optique de minimiser les ressources nécessaires à la réalisation d'une tâche précise nous présentons une étude systématique des signaux élémentaires (se déplaçant d'une case toutes les p étapes de calcul), briques de base de beaucoup de constructions sur lignes d'automates. Nous caractérisons le nombre minimal d'états requis pour programmer certains signaux. L'étude est presque close pour les signaux élémentaires mais reste à poursuivre pour les signaux de pente rationnelle. Une motivation secondaire à l'étude de tels signaux est la création d'une solution synchronisante de la famille de celle de Minsky fonctionnant avec 6 états seulement. Bien que n'ayant pas encore trouvé une telle solution nous pensons que certains signaux très économes peuvent s'avérer utiles. Nous donnons des indices dans ce sens dans la section 4 du chapitre 8. En particulier, nous présentons une solutions à 6 états fonctionnant pour une infinité de longueurs de lignes (mais pas toutes) et utilisant un signal économe, et même optimal en nombre d'états.

Notons que d'autres notions de ressources existent, comme par exemple le nombre de communications nécessaires à la réalisation d'une tâche (cette mesure est très importante quand les communications ont un coût ou dans le cadre de la tolérance aux pannes). Des références traitant de ces aspects sont citées à la fin de cette introduction. Ce point n'est pas abordé dans la thèse mais une étude du nombre de communications dans certaines solutions synchronisantes à été réalisée lors de mon stage de D. E. A. [Hee92].

1.2 Regroupement

Une nouvelle technique de simulation d'automates cellulaires s'est avérée utile à plusieurs reprises. Il s'agit de la méthode de regroupement que nous décrivons au chapitre 3 et que nous utilisons en filigrane tout au long des chapitres concernant l'accélération. En particulier, les théorèmes d'accélération d'une constante pour reconnaisseurs (chapitre 4, théorème 4.1, page 52) et d'accélération linéaire pour reconnaisseurs (chapitre 6, théorème 6.1, page 87) sont obtenus à l'aide de regroupements *purs* (c'est-à-dire sans faire appel au produit traditionnel d'automates).

Il est intéressant de noter que les résultats obtenus par regroupement dans ces deux cas sont meilleurs, du point de vue des ressources, que ceux obtenus par les méthodes traditionnelles. Dans d'autres cas comme le ralentissement d'une constante des reconnaisseurs, un regroupement fonctionne moins bien que certaines techniques optimisées mais permet néanmoins d'aboutir au résultat. Dans le cas des calculateurs, enfin, la plupart des solutions efficaces que nous proposons sont basées sur des regroupements (sans être des regroupements purs cependant).

Notre sentiment est que la méthode de regroupement est un nouvel outil puissant qui peut être appliqué à une grande variété de problèmes, outre les problèmes d'accélération déjà évoqués ; citons en particulier la composition des calculateurs et la réversibilité des simulations. Notons que, dans le cadre de la composition des calculateurs, cette méthode semble avoir des liens avec les grilles de J. Mazoyer.

Les deux remarques suivantes permettent de considérer la méthode de regroupement dans un contexte plus large :

1. la simulation par regroupement n'est pas la seule méthode d'accélération envisageable ; il existe d'autres approches, en particulier celle de C. Moore qui prend en compte les caractéristiques algébriques de certains automates cellulaires (les *automates treillis*) pour lesquels la loi de transition, binaire, peut posséder certaines propriétés facilitant l'accélération.

   Nous proposons à la fin de cette introduction une liste de références concernant l'aspect algébrique bien que ce ne soit pas l'approche que nous retenons dans cette thèse.

   En effet, notre choix est de fournir des méthodes d'accélération, et plus généralement de simulation, indépendantes des propriétés initiales de l'automate cellulaire. En fait, pour la méthode de regroupement, la fonction de transition initiale n'est pas *reprogrammée* ; en particulier, connaissant la méthode de regroupement utilisée on peut retrouver la fonction initiale à partir de la version simulée ;

2. depuis longtemps on connaît des liens étroits entre automates cellulaires et pavages. Cette voie a déjà fourni des résultats importants, notamment à propos du problème de la réversibilité des automates cellulaires. Notre méthode de regroupement permet d'interpréter le calcul de certains automates cellulaires particuliers comme une déformation du demi-plan discret $\mathbb{Z} \times \mathbb{N}$. Certaines conditions techniques (décrites au chapitre 3) garantissent la connexité horizontale des lignes déformées. L'effet du regroupeur peut alors être interprété comme un pavage du demi-plan (avec ou sans recouvrement).

1.3 Accélération ou échange temps/états?

Nous émettons ici une remarque concernant les méthodes d'accélération qui dépasse le cadre des automates cellulaires: de nombreuses techniques d'accélération de machines utilisent des produits d'états impliquant un accroissement polynomial de la taille des alphabets ([IJ89, IKM85, MR92]) ou des fonctions de prédiction des états futurs avec accroissement exponentiel ([CČ83, Gef93]). La machine obtenue effectue moins de pas de calcul (elle « va plus vite ») mais elle est aussi plus complexe. En particulier, les règles de transition de la machine, automate cellulaire ou machine de Turing par exemple, sont plus nombreuses et traitent des objets plus complexes. Il faut donc plus de temps pour sélectionner chaque règle. Si l'on n'utilise aucune astuce de compression on d'indexation rapide de la table des transitions, le temps de sélection d'une règle est exactement multiplié par le gain d'étapes de calcul. Il y a moins d'étapes mais elles sont plus longues et donc on ne gagne pas de temps !

En pratique l'accroissement du temps de recherche d'une règle est souvent plus faible que le gain en nombre de pas de calcul et les astuces de compression évoquées plus haut peuvent être mises en place (notamment dans le cas d'un accroissement polynomial seulement du nombre d'états). Mais d'un point de vue théorique il serait plus correct de parler d'échange temps/états (nous gardons néanmoins dans cette thèse le terme usuel « d'accélération »).

2 Organisation du document

Note sur le vocabulaire : par la suite nous parlons fréquemment de théorèmes d'accélération en version *faible* ou *forte*. Un théorème en version forte s'applique à toutes les machines du type concerné et quelle que soit la longueur de l'entrée. Un théorème en version faible peut ne s'appliquer qu'à des machines « suffisamment lentes » où à certaines longueurs de lignes particulières.

Chapitre 1 Cette introduction. Une liste de références bibliographiques est proposée au dernier paragraphe.

Chapitre 2 Nous introduisons les notions essentielles, ainsi que les notations et quelques lemmes techniques utilisés dans les chapitres suivants. Le dernier paragraphe propose un bref survol du problème de la synchronisation d'une ligne de fusiliers, essentiel par la suite (surtout au chapitre 7).

Chapitre 3 Nous donnons les définitions nécessaires à la mise au point de la notion d'automate regroupeur et les lemmes qui l'accompagnent, puis nous illustrons la méthode sur un exemple simple : le ralentissement d'un pas des calculateurs.

Chapitre 4 C. Choffrut et K. Čulik ont donné dans [CČ83] une preuve directe du théorème d'accélération d'une constante pour reconnaisseurs au prix d'un accroissement exponentiel du nombre d'états. Or, plus tard, d'autres auteurs [IKM85, MR92] ont proposé des preuves du théorème d'accélération linéaire en version forte avec accroissement polynomial (seulement) du nombre d'états.

Dans le cadre des reconnaisseurs, tout d'abord, nous indiquons en quoi le premier théorème est un corollaire du second (et fournissons ainsi une preuve du théorème d'accélération d'une constante pour reconnaisseurs ne nécessitant plus qu'un accroissement polynomial des ressources). Puis nous en formulons une nouvelle preuve par la méthode de groupage, qui minimise la densité d'utilisation des nouvelles ressources.

Chapitre 5 La preuve initiale de Choffrut et Čulik fonctionne également dans le cas, plus difficile, des calculateurs (mais toujours avec un accroissement exponentiel du nombre d'états). Nous donnons une nouvelle preuve du théorème d'accélération d'une constante pour calculateurs en gardant un accroissement modéré et un faible taux d'utilisation des ressources.

Chapitre 6 Nous proposons une nouvelle preuve du théorème d'accélération linéaire pour reconnaisseurs, précédemment étudié par Cole, Smith, Ibarra *et al.*, Mazoyer et Reimen [Col69, Smi71b, IKM85, MR92]. Cette nouvelle preuve, directe, n'utilise pas de solution synchronisante ; là encore, la méthode de regroupement est employée avec succès.

Chapitre 7 Un « défaut » de la théorie des automates cellulaires est l'absence de théorème d'accélération linéaire fonctionnant pour tous les calculateurs (en version forte donc). Nous en donnons néanmoins une version adaptée à la sous classe, importante en pratique, des synchroniseurs. C'est, à notre connaissance, la première incursion de l'accélération linéaire en version forte hors du domaine des reconnaisseurs.

Comme application directe des résultats précédents nous établissons la clôture de la classe des temps de synchronisation par composition avec les polynômes à coefficients rationnels positifs ou négatifs, améliorant ainsi un résultat de J. Mazoyer concernant seulement les polynômes à coefficients entiers et positifs.

Note : dans tous les cas, les polynômes considérés doivent excéder un certain minimum.

Chapitre 8 Enfin nous étudions en tant qu'objets les signaux élémentaires, utilisés comme composants dans les chapitres précédents (et dans la littérature, notamment pour la fabrication de solutions synchronisantes). Nous donnons une borne supérieure du nombre d'états nécessaires à la réalisation de chaque signal. En outre une recherche systématique (par programme) permet d'exhiber quelques codages optimaux ; nous les utilisons pour mettre au point des solutions synchronisantes partielles utilisant peu d'états.

Annexes Pour mener à bien une partie des recherches, nous avons mis au point certains programmes utilitaires en langage Pascal. Une documentation succincte ainsi que des exemples de fonctionnement sont fourni en annexe B. Nous donnons également le code source d'un des programme de recherche de signaux élémentaires. Par ailleurs, le code source (en Prolog) d'un programme de validation des automates regroupeurs est fourni en annexe A.

3 Quelques références

Nous indiquons ici quelques références permettant d'aborder plus en détail certains aspects importants évoqués ci-avant.

- Simulation de phénomènes par automates cellulaires: c'est peut être l'aspect le plus immédiatement séduisant des automates cellulaires et il existe un grand nombre de références s'y rapportant, citons par exemple S. Wolfram [Wol84a], A. G. Barto [Bar75], N. Margolus [Mar84], G. Vishniac [Vis84] et H. Gutowitz [Gut91, Gut].

- À propos de la réversibilité, T. Toffoli et N. Margolus proposent un article de synthèse [TM90], J. Kari prouve l'indécidabilité du problème de la réversibilité en deux dimensions [Kar90, Dur93]. Sutner [Sut95] étudie d'autres résultats de complexité liés à la dimension. Un instrument usuel de la réversibilité, les *automates cellulaires à partition*, présente certaines analogies (sinon formelles, du moins dans l'esprit) avec la méthode de regroupement que nous développons au chapitre 3. Les travaux de K. Morita [Mor95] s'y rapportent. Note : il existe une autre variété d'automates cellulaires à partition développée par N. Margolus [Mar84] (voir également les travaux de J. O. Durand-Lose [Dur96a]).

- L'aspect intrinsèque de certaines propriétés que nous évoquions au début de l'introduction apparaît par exemple grâce aux travaux de K. Lindgren et M. Nordahl

[LN88, LN90], J. Albert et K. Čulik [AČ87], E.Goles *et al.* [GMM93] et B. Martin [Mar94] pour l'universalité. Pour l'accélération et les classes de complexité, nous citerons J. Mazoyer et N. Reimen [MR92], C. Choffrut et K. Čulik [CČ84, CČ83], C. Papadimitriou et J. Ullman [PU87], K. Sutner [Sut95], O. H. Ibarra, M. A. Palis et S. M. Kim [IPK85], A. R. Smith [Smi71a] et enfin V. Terrier [Ter91, Ter95, Ter96] pour l'aspect temps réel.

– À propos de l'approche algébrique des problèmes d'accélération : C. Moore étudie des règles de transition conférant une structure particulière à l'ensemble des configurations (groupes, semi-groupes) [Moo95, Moo96], on citera également A. D. Robinson [Rob87], T. Boykett [Boy95] et J. Pederson [Ped82] pour une approche algébrique d'autres types de problèmes.

– Le problème de la synchronisation d'une ligne d'automates est abondamment traité par J. Mazoyer [Maz87, Maz90]. Voir également les travaux de R. Balzer [Bal67b] et J. B. Yunès [Yun93, Yun94] et par exemple « l'école italienne » pour quelques variations [Rom76, Gra75].

  L'aspect tolérance aux pannes à été étudié par H. Nishio et Y. Kobuchi, M. Harao et S. Noguchi [HN75], et plus récemment par H. Umeo [Ume93], J. B. Yunès [Yun96], M. Kutrib et R. Vollmar [KV95].

– Enfin, des liens entre pavages et automates cellulaires ont souvent été mis en évidence (en dehors de l'aspect réversibilité, déjà évoqué). Citons par exemple J. Mazoyer, J. Kari [Kar90] avec K. Čulik [ČK95] et P. Aigrain avec D. Beauquier [AB95].

Chapitre 2

Définitions et propriétés

1 Conventions et notations

Nous notons $\mathbb{Z}$ l'ensemble des entiers et $\mathbb{N}$ l'ensemble des entiers positifs ou nuls.

Dans la suite du document, les lettres majuscules calligraphiées $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ désignent généralement des ensembles finis. Les lettres majuscules droites (A, B, M) désignent des noms de machines (la lettre M étant plutôt réservée aux regroupeurs) ou de fonctions réalisées par des machines. Les lettres minuscules désignent des états (a, b, c), des entiers (i, j, k) ou des mots (u, v, w).

Le produit de concaténation est noté « $\cdot$ » quand il n'est pas implicite. Si $\mathcal{A}$ est un alphabet, on note $\mathcal{A}^*$ l'ensemble des suites finies sur $\mathcal{A}$, $\mathcal{A}^\omega$ l'ensemble des suites semi-infinies sur $\mathcal{A}$ et ${}^\omega\mathcal{A}^\omega$ l'ensemble des suites bi-infinies sur $\mathcal{A}$.

À propos des états, nous utilisons si nécessaire la convention suivante : les lettres du début de l'alphabet sont des variables contenant des états nécessairement non quiescents, les lettres de la fin de l'alphabet sont des variables désignant des états potentiellement quiescents. Ainsi une règle de bord gauche, par exemple, s'écrit de la façon suivante : $\delta(\sharp, a, z) = \ldots$ ce qui insiste sur le fait que le deuxième argument ne doit pas être l'état quiescent.

Dans les diagrammes espace-temps présentés en exemple tout au long de la thèse nous appliquons les conventions suivantes : les cellules sont représentées de gauche à droite par ordre de numéro croissant à partir de 0 et le temps va de **haut en bas** à partir du temps 0. Cette convention n'est pas celle qu'utilise J. Mazoyer par exemple mais elle est compatible avec l'affichage des diagrammes espace-temps produits par des programmes de simulation et avec le sens de l'écriture.

À partir du chapitre 3 nous utiliserons des translations de portions P du plan suivant un vecteur v, le resultat est simplement noté $P + v$.

Des notations plus spécifiques seront indiquées au fur et à mesure.

2 Objets de base

Notion intuitive d'automate cellulaire

Un automate cellulaire est une machine parallèle à fonctionnement uniforme, synchrone et local. Nous explicitons ces notions.

Machine : les automates cellulaires permettent d'effectuer des *calculs*. Les fonctions calculées par les automates cellulaires sont celles calculées par les machines de Turing, ce qui en fait un modèle universel [GMM93, LN90, Mar94, Smi71b]. Les automates cellulaires travaillent au moins aussi vite que les machines de Turing [AČ87, LN90, Mar94, Smi71b, Wol84b] et le plus souvent plus vite [Smi72, Fis65], ce qui contribue à l'intérêt du modèle.

Parallèle : un automate cellulaire regroupe plusieurs unités de calcul (élémentaires), capables éventuellement de travailler indépendamment les unes des autres, c'est-à-dire sans s'attendre entre elles sauf pour communiquer.

Uniforme : ces machines élémentaires, les cellules, sont toutes identiques. À un instant du calcul elles ne diffèrent que par les données qu'elles contiennent. C'est peut être la propriété la plus caractéristique du modèle qui se différencie ainsi d'autres machines parallèles (les *parallel random access machines* par exemple).

Synchrone : toutes les cellules travaillent par « bonds » identiques et synchrones. En un bond, chaque cellule prend connaissance de l'état de certaines de ses voisines et utilise cette information pour calculer son nouvel état. Cette caractéristique permet d'exprimer le comportement de l'ensemble de la machine par une fonction et de le représenter par un diagramme espace-temps dont on peut étudier les propriétés géométriques.

Local : chaque cellule a accès à une quantité d'information limitée. Elle connaît l'état de quelques voisines seulement et dispose d'une mémoire finie. En particulier, une cellule ne peut pas mémoriser toute seule le nombre total de cellules ni accéder directement aux valeurs de cellules lointaines (mais en revanche, plusieurs cellules peuvent coopérer pour y parvenir).

Définition 2.1 (automate cellulaire)
Un automate cellulaire A est un quadruplet $(\mathcal{Q}, d, \mathcal{V}, \delta)$ où :

- $\mathcal{Q} = \{q_1, \ldots, q_s\}$ *est un ensemble fini d'états ;*
- $d \in \mathbb{N}, (d > 0)$ *est la dimension ;*
- $\mathcal{V} \subset \mathbb{Z}^d, \mathcal{V} = \{v_1, \ldots, v_{Card(\mathcal{V})}\}$ *est le voisinage (fini);*
- $\delta : dom(\delta) \subset \mathcal{Q}^{\mathcal{V}} \to \mathcal{Q}$ *est la fonction de transition locale.*

Remarque
Dans la définition précédente, il est essentiel que l'ensemble des états et le voisinage soient finis. Si l'une de ces conditions n'est pas vérifiée, la machine obtenue peut calculer une fonction non récursive.

Notions subséquentes

- Une *configuration* de l'automate cellulaire A est une application $c : \mathbb{Z}^d \to \mathcal{Q}$. En d'autres termes, c'est une description complète de l'état global de l'automate cellulaire. De façon équivalente, une configuration peut être donnée sous forme de liste infinie plutôt que sous forme de fonction ;

– la *fonction de transition globale* induite par δ est la fonction Δ définie par :

$$\begin{array}{llll} \Delta : & \mathcal{Q}^{\mathbb{Z}^d} & \longrightarrow & \mathcal{Q}^{\mathbb{Z}^d} \\ & c & \longmapsto & \Delta(c), \forall i\, \Delta(c)(i) = \delta(c(i+v_1), \ldots, c(i+v_{Card(\mathcal{V})})) \end{array}$$

c'est la fonction qui calcule le passage d'une configuration à la suivante ;

– la fonction diagramme espace-temps de $\mathbb{Z}^d \times \mathbb{N}$ dans $\mathcal{Q}$ est la liste des configurations successives de l'espace de cellules. Elle est notée $\langle\ \rangle$. Afin de clarifier certains exemples (notamment dans les chapitres 4 et 5) nous pourrons utiliser les notations abrégées suivantes :

$$\begin{array}{c} \langle c_1, c_2, c_3, \ldots | t\rangle \text{ pour } (\langle c_1, t\rangle, \langle c_2, t\rangle, \langle c_3, t\rangle, \ldots) \text{ et} \\ \langle a..b|t\rangle \text{ pour } (\langle a, t\rangle, \langle a+1, t\rangle, \ldots, \langle b-1, t\rangle, \langle b, t\rangle). \end{array}$$

ces notations seront commodes lors des simulations dans lesquelles certaines cellules simulantes contiennent des produits d'états de cellules simulées.

Définition 2.2 (automate cellulaire simple)
Un automate cellulaire simple est un automate cellulaire $A = (\mathcal{Q}, d, \mathcal{V}, \delta)$:

– *de dimension un ($d = 1$) ;*

– *dont le voisinage est restreint aux plus proches voisins ($\mathcal{V} = \{-1, 0, 1\}$) ;*

– *et dans lequel on distingue un état quiescent particulier $\sharp$ (tel que $\delta(\sharp, \sharp, \sharp) = \sharp$).*

Frontière

– Un automate cellulaire simple a la propriété de *frontière à gauche* si :
$\forall x \in \mathcal{Q}, \delta(\sharp, \sharp, x) = \sharp$. L'état quiescent $\sharp$ est alors appelé état frontière gauche ;

– un automate cellulaire simple a la propriété de *frontière à droite* si :
$\forall x \in \mathcal{Q}, \delta(x, \sharp, \sharp) = \sharp$. L'état quiescent $\sharp$ est alors appelé état frontière droite ;

– un automate cellulaire simple a la propriété de *frontière* s'il possède les propriétés de frontière à gauche et à droite. L'état $\sharp$ est alors appelé état frontière.

Remarques

– par la suite, nous utilisons essentiellement des machines à base d'automates cellulaires simples ;

– pour de tels automates cellulaires les configurations sont données par des mots bi-infinis plutôt que par des fonctions de $\mathbb{Z}$ dans $\mathcal{Q}$. Dans le cadre qui nous intéresse (reconnaissance de mots, calcul de fonction), nous utilisons des configurations *finies*, c'est-à-dire presque partout quiescentes. Plus précisément, nous pouvons construire une configuration $c(u)$ à partir d'un mot fini $u = u_1 \cdots u_n$ de la façon suivante :

$$\begin{array}{c} \forall i < 0, c(i) = \sharp \\ \forall 0 \leq i < n, c(i) = u_{i+1} \\ \forall i \geq n, c(i) = \sharp \end{array}$$

Le mot décrivant $c(u)$ est alors ${}^{\omega}\sharp \cdot u_1 \cdots u_n \cdot \sharp^{\omega}$;

- quand la fonction de transition δ n'est pas totale, on se restreint aux *configurations non bloquantes*, c'est-à-dire pour lesquelles les états de trois cellules consécutives forment bien un triplet de $dom(\delta)$;
- le diagramme espace-temps est alors un demi plan, que nous pourrons limiter en fonction des propriétés de frontière de l'automate cellulaire simple ;
- à partir de maintenant, et sauf mention contraire, nous écrirons automate cellulaire au lieu d'automate cellulaire simple.

3 Différentes sortes de machines

Des machines séquentielles classiques, comme les machines de Turing [Min67, Pap94] ou les machines RAM (*Random Access Machines*), peuvent être utilisées de différentes manières : comme reconnaisseurs elles décident si un mot appartient ou non à un certain langage, comme calculateurs elles transforment un mot en un autre. Les reconnaisseurs réalisent des prédicats sur les mots, les calculateurs réalisent des fonctions partielles sur les mots.

Nous donnons ici des notions similaires pour les machines à base d'automates cellulaires simples.

Notion intuitive de reconnaisseur

Un grand nombre de problèmes attendent une réponse de type oui/non ; ce sont les problèmes de décision. Typiquement ils s'énoncent ainsi : « le mot u appartient-t-il au langage $\mathcal{L}$? ». Pour traiter ces problèmes, les automates cellulaires doivent être munis d'un ensemble d'entrée (l'alphabet du langage) et d'au moins deux symboles de sortie signifiant respectivement **oui** et **non**. Un tel dispositif est appelé reconnaisseur. Dans [DM94], M. Delorme et J. Mazoyer proposent un survol des différents types de reconnaisseurs et des problèmes associés.

Définition 2.3 (reconnaisseur)

Un reconnaisseur R est un quadruplet $(A, \mathcal{E}, \mathcal{S}_a, \mathcal{S}_r)$ où :

- $A = (\mathcal{Q}, \delta)$ *est un automate cellulaire simple ;*
- $\mathcal{E} \subset \mathcal{Q}$ *est un alphabet fini de symboles d'entrée ;*
- $\mathcal{S}_a \subset \mathcal{Q}$ *est un ensemble fini de symboles d'acceptation, non nécessairement disjoint de* $\mathcal{E}$ *;*
- $\mathcal{S}_r \subset \mathcal{Q}$ *est un ensemble fini de symboles de rejet, non nécessairement disjoint de* $\mathcal{E}$ *mais disjoint de* $\mathcal{S}_a$ *;*
- *dans un but de simplification et sans perte de généralité, l'état quiescent distingué* $\sharp$ *n'appartient ni à* $\mathcal{E}$ *ni à* $\mathcal{S}_a$ *ni à* $\mathcal{S}_r$*.*

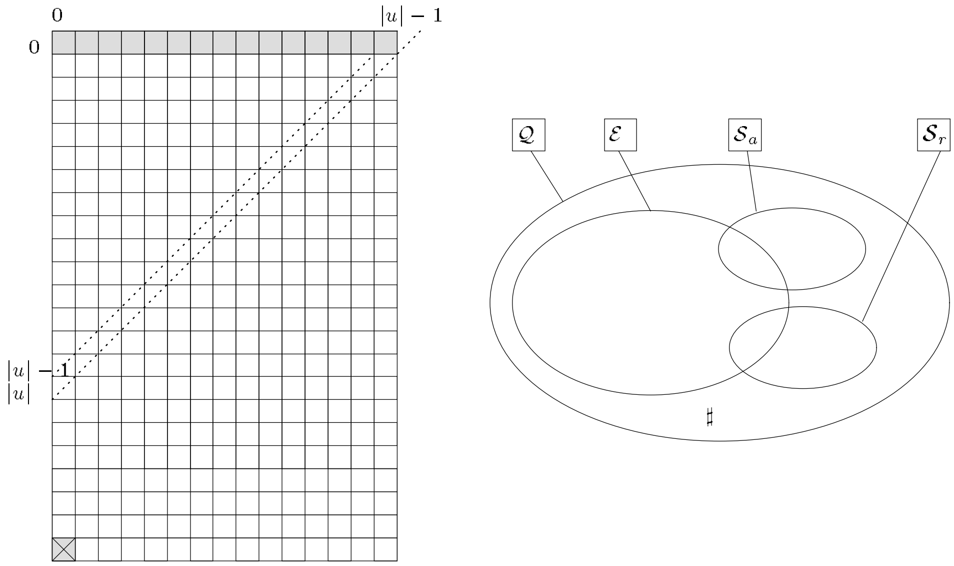


FIG. 2.1 – *Un reconnaisseur.*

Frontière

Dans la suite nous ne considérons que des reconnaisseurs calculant à partir d'une configuration finie telle que décrite en section 2 de ce chapitre. Un reconnaisseur est dit fini (resp. semi-infini) si sa partie automate cellulaire possède la propriété de frontière (resp. de frontière à gauche). Il est dit bi-infini si sa partie automate cellulaire ne possède aucune propriété de frontière.

Notions subséquentes

- Nous désignons par la même lettre R le nom d'un reconnaisseur et le prédicat sur les mots qu'il réalise, c'est-à-dire la fonction $R : \mathcal{E}^* \to \mathcal{S}_a \cup \mathcal{S}_r$, qui à un mot u associe la première occurrence d'un état de $\mathcal{S}_a$ ou de $\mathcal{S}_r$ sur la cellule de numéro zéro quand R est lancé sur la configuration initiale $c(u)$;
- le temps de calcul est la fonction $T : \mathcal{E}^* \to \mathbb{N}$, qui à un mot u associe le nombre minimum d'étapes de Δ pour aboutir à la configuration résultat. En termes de diagramme espace-temps : $T(u) = \min\{t > 1, \langle 0, t\rangle \in \mathcal{S}_a \cup \mathcal{S}_r\}$;
- remarquons que le calcul de R peut ne jamais aboutir (même pour les reconnaisseurs finis), et donc les fonctions ci-dessus ne sont pas nécessairement totales.

Modes d'entrée et de sortie

- Le mode d'entrée retenu pour les reconnaisseurs (et ci-après pour les calculateurs) est dit parallèle : au temps zéro, toutes les cellules connaissent leur information de départ. Ceci suppose que chaque cellule dispose d'un moyen de lire une donnée provenant de « l'extérieur ». Dans le cadre de la composition d'automates cellulaires, cet extérieur est un autre automate cellulaire. Dans un cadre plus physique il peut s'agir d'un périphérique d'acquisition (rétine, capteur de température ou information provenant d'un banc de disques) ...

- de plus, la cellule zéro dispose d'un moyen de communication vers l'extérieur puisque c'est elle qui « affiche » le résultat ;
- généralement, on exige que les ensembles d'entrée et de sortie soient disjoints. Nous remplaçons cette condition par la suivante : le résultat (éventuel) d'un calcul de reconnaisseur n'est pris en compte qu'après un pas de calcul (au moins). Ainsi, un mot d'entrée commençant par un symbole de $\mathcal{S}_a$ ou de $\mathcal{S}_r$ ne donne pas lieu à une sortie immédiate (en temps zéro). Cette remarque prend plus de sens dans le cadre des calculateurs.

« Insuffisance » des reconnaisseurs

Tous les problèmes (calculatoires) ne s'énoncent pas de façon naturelle sous forme de problèmes de décision. Par exemple, la conversion d'une base vers une autre, la multiplication d'entiers, le tri de valeurs s'accommodent mal d'un formalisme du type reconnaisseur. Certes, il est possible de soumettre à un reconnaisseur une question du type « la partie droite de l'entrée correspond elle à la valeur d'une fonction sur la partie gauche de l'entrée ? », en énumérant différentes entrées jusqu'à obtenir une réponse positive. La perte d'efficacité, cependant, rend cette méthode inutilisable en pratique.

Notion intuitive de calculateur

Il faut donc élaborer un modèle capable de traiter efficacement les problèmes de transformation de mots. En munissant chaque cellule (et non plus seulement la cellule zéro) d'un moyen d'envoyer des données vers l'extérieur, on obtient un modèle qui *généralise* les reconnaisseurs et permet de traiter naturellement les problèmes évoqués ci-dessus.

Définition 2.4 (calculateur)

Un calculateur C est un triplet $(A, \mathcal{E}, \mathcal{S})$ où :

- $A = (\mathcal{Q}, \delta)$ *est un automate cellulaire simple ;*
- $\mathcal{E} \subset \mathcal{Q}$ *est un alphabet fini de symboles d'entrée ;*
- $\mathcal{S} \subset \mathcal{Q}$ *est un alphabet fini de symboles de sortie, non nécessairement disjoint de* $\mathcal{E}$ *;*
- *l'état quiescent* $\sharp$ *n'appartient ni à* $\mathcal{E}$ *ni à* $\mathcal{S}$*.*

Frontière

Comme pour les reconnaisseurs, on distingue les calculateurs finis, semi-infinis et bi-infinis.

Notions subséquentes

- Nous désignons par la même lettre C la fonction de $\mathcal{E}^*$ dans $\mathcal{S}^*$ et le calculateur qui la réalise. Cette fonction est définie sur u si et seulement si, le calculateur étant lancée sur la configuration initiale $c(u)$, la cellule zéro entre dans un état de $\mathcal{S}$, et au premier temps où elle y rentre la configuration est de la forme $c(v)$ avec $v \in \mathcal{S}^*$.

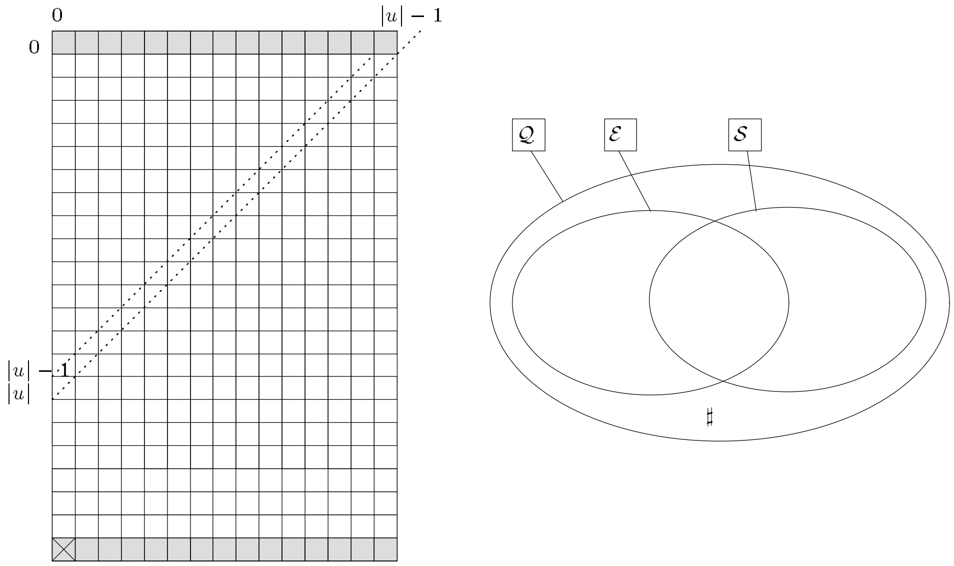


FIG. 2.2 – *Un calculateur.*

- la notion de temps de calcul sur une entrée est définie de la même manière que pour les reconnaisseurs ;
- là encore, les fonction C et T ne sont pas nécessairement totales.

Modes d'entrée et de sortie

- Comme pour les reconnaisseurs, le mode d'entrée est parallèle ;
- le mode de sortie utilisé est le suivant : après un premier pas de calcul (obligatoire), la sortie est la configuration atteinte par la machine dès que la cellule zéro entre dans un état de $\mathcal{S}$. Remarquons qu'à cet instant, toutes les cellules doivent être dans un état de $\mathcal{S}$. Il y a donc une notion de sortie valide ou non (cohérente avec le fait que la fonction P réalisée n'est pas toujours totale) ;
- dans [Maz92], J. Mazoyer décrit d'autres modes d'entrée et de sortie, ainsi que des façons de les simuler mutuellement. Selon la terminologie suggérée, nos calculateurs utilisent un mode proche de l'entrée parallèle / sortie parallèle *asynchrone* ;
- indépendamment, une sortie peut être *stable* (c'est-à-dire le résultat reste « affiché ») ou non (c'est-à-dire le calcul se poursuit, par exemple pour une composition de calculateurs).

Notion intuitive de synchroniseur

Il existe une variété particulière de calculateurs, suffisamment remarquable pour que nous en donnions une définition spécifique : les synchroniseurs. Il s'agit de calculateurs finis qui, partant d'une configuration entièrement quiescente savent aboutir à une configuration où toutes les cellules entrent *simultanément* et pour la *première fois* dans un état particulier.

Définition 2.5 (synchroniseur)

Un synchroniseur est un calculateur $S = (A, \{q_0\}, \{q_1\})$ *(*q_0*,* $\sharp$ *et* q_1 *distincts) qui respecte les conditions suivantes (pout tout* $n > 0$*) :*

1. *S est fini (c'est-à-dire* $\delta(\sharp, \sharp, x) = \delta(y, \sharp, \sharp) = \sharp$*) ;*
2. *S réalise le programme (total)* $P(\sharp.q_0^n\sharp) = \sharp.q_1^n.\sharp$ *;*
3. $\delta(q_0, q_0, q_0) = q_0$ *(*q_0 *est quiescent) ;*
4. $\forall\, 0 < i \leq T(\sharp.{q_0}^n.\sharp), \forall\, 0 \leq j < n, \langle i, j\rangle \neq q_1$ *(c'est-à-dire* q_1 *n'apparaît pas avant la fin du calcul).*

Remarques

– cette définition diffère légèrement des définitions usuelles de synchroniseurs (*cf.* [Bal67a, Maz87, Yun94] par exemple) car il n'existe pas d'état particulier pour démarrer la synchronisation. Ce sont les conditions aux bords (c'est-à-dire les valeurs de $\delta(\sharp, a, a)$, de $\delta(a, a, \sharp)$ et éventuellement de $\delta(\sharp, a, \sharp)$) qui conditionnent le début du calcul ;

– dans ces conditions, les temps usuels de synchronisation (typiquement $n - 1$ et $2n - 2$, *cf.* section 5 de ce chapitre) sont augmentés d'une unité, ce qui est cohérent avec notre requête que tout calcul effectue au moins un pas ;

– le mode de sortie utilisé par les synchroniseurs est un cas particulier de sortie parallèle, dite *synchrone* (selon J. Mazoyer dans [Maz92]) ce qui correspond à la clause 3 de la définition 2.5 ;

– dans le « folklore » des automates cellulaires, le problème de la recherche de synchroniseurs est appelé problème de la synchronisation d'une ligne de fusiliers (FSSP pour *Firing Squad Synchronization Problem*). L'état « soldat » est désigné par a, l'état « **feu** » par b. Un état désigné de démarrage du processus serait appelé le « général » ;

– on trouvera un bref état de l'art de la synchronisation à la section 5 de ce chapitre.

Le lemme que nous énonçons maintenant s'avère utile tout au long des chapitres concernant l'accélération des automates cellulaires:

Lemme 2.1 (synchronisation à partir des deux extrémités de la ligne)

Il existe un synchroniseur $S = (A, \{q_0\}, \{q_1\})$ *qui, pour tout entier n strictement positif, partant de la configuration partout quiescente de longueur n (*$\sharp.q_0^n.\sharp$*) aboutit à la configuration* $\sharp.q_1^n.\sharp$ *en exactement n étapes.*

Une preuve est décrite par Mazoyer et Reimen [MR92]. Elle est basée sur l'utilisation d'une solution en temps minimal à partir d'une seule extrémité de la ligne (voir [Bal67a, Maz87]).

4 Quelques propriétés

Remarque

À propos du temps de réponse (T) d'un calculateur sur une entrée (u), il est classique de ne considérer que la longueur (n) de l'entrée dans le calcul de $T(u)$. Ceci correspond à une évaluation dans le pire des cas, c'est-à-dire $\hat{T}(n) = \max\{T(u), |u| = n\}$. Par abus de notation, on confond souvent $\hat{T}$ et T.

Il se trouve que dans ce travail nous pouvons souvent déterminer exactement $T(u)$. Pour les machines simulantes en particulier on exprimera leur temps de calcul en fonction du temps de calcul $T(u)$ de la machine simulée.

Par ailleurs, étant donnée une entrée u, on montre que la dernière lettre de u ne peut influer sur le résultat qu'à partir du temps $|u| - 1$ (voir figure 2.2). Si l'on exige, de plus, que la cellule zéro (seule apte à décréter la fin d'un calcul) prenne note que *tout* le mot d'entrée à été lu, il faut attendre l'étape $|u|$.

De fait, nous nous intéressons essentiellement aux calculs en temps au moins $|u|$ et utilisons dorénavant la notation $t(u)$ où $t(u) = T(u) - |u|$.

Une grande partie de la thèse est consacrée au développement de méthodes de simulation efficaces de machines entre elles. Nous définissons l'équivalence de deux machines puis une notion d'équivalence modifiée : la simulation.

Notion intuitive d'équivalence

Deux machines sont équivalentes si elles réalisent les mêmes prédicats (ou programmes). En particulier elles doivent avoir même alphabet d'entrée et de sortie.

Définition 2.6 (équivalence d'automates cellulaires)

- *deux reconnaisseurs* $R = ((\mathcal{Q}, \delta), \mathcal{E}, \mathcal{S}_a, \mathcal{S}_r)$ *et* $R' = ((states', \delta'), \mathcal{E}, \mathcal{S}_a, \mathcal{S}_r)$ *sont équivalents si* $\forall u \in \mathcal{E}^*, R(u) = R'(u)$ *;*

- *deux calculateurs* $C = ((\mathcal{Q}, \delta), \mathcal{E}, \mathcal{S})$ *et* $C' = ((\mathcal{Q}', \delta'), \mathcal{E}, \mathcal{S}')$ *sont équivalents si* $\mathcal{S} = \rho(\mathcal{S}')$ *et* $\forall u \in \mathcal{E}^*, C(u) = \rho(C'(u))$ *où* $\rho : \mathcal{S}' \to \mathcal{S}$ *est une fonction de renommage ;*

- *deux calculateurs* $C = ((\mathcal{Q}, \delta), \mathcal{E}, \mathcal{S})$ *et* $C' = ((\mathcal{Q}', \delta'), \mathcal{E}, \mathcal{S})$ *sont fortement équivalents si* $\forall u \in \mathcal{E}^*, C(u) = C'(u)$.

Remarques

- au sens de la définition précédente, tous les synchroniseurs sont équivalents, à un renommage près de leur symbole de départ ;

- dans le cas des reconnaisseurs, il est facile d'exiger l'égalité des alphabets de sortie (l'important n'étant pas le symbole de sortie en lui-même mais son appartenance à $\mathcal{S}_a$ ou $\mathcal{S}_r$). Cette dernière condition est souvent plus difficile à mettre au point pour les calculateurs, aussi autorise t'on souvent un renommage (éventuellement partiel) de l'alphabet de sortie.

Notion intuitive de simulation

La simulation d'une machine, reconnaisseur ou calculateur, par une autre du même type est une équivalence particulière où la fonction de transition simulante δ' est construite à partir de la fonction initiale δ (par exemple par composition avec les fonctions de projection, de permutation des entrées et de construction de k-uplets).

On trouve diverses notions de simulation dans la littérature (en particulier celles de S. Cole [Col69], C. Choffrut et K¿ Čulik [CČ83], K. Morita [Mor95], B. Martin [Mar92] et, plus récemment J. O. Durand-Lose [Dur96b].

Dans cette thèse, nous utilisons une notion plus pragmatique de simulation, compatible avec la définition 2.6 d'équivalence :

Définition 2.7 (simulation pour reconnaisseurs)

Une simulation est une application qui au reconnaisseur $R = ((\mathcal{Q}, \delta), \mathcal{E}, \mathcal{S}_a, \mathcal{S}_r)$ *associe un reconnaisseur* $R' = ((\mathcal{Q}', \delta'), \mathcal{E}, \mathcal{S}_a, \mathcal{S}_r)$ *tel que :*

1. $\mathcal{Q} \subseteq \mathcal{Q}'$;
2. *R et R' ont le même état frontière* $\sharp$;
3. *δ' et δ coïncident sur* $\mathcal{Q} \setminus \{\sharp\}$ *; c'est-à-dire* $\forall a, b, c \in \mathcal{Q} \setminus \{\sharp\}, \delta'(a, b, c) = \delta(a, b, c)$;
4. *R et R' sont équivalents.*

Remarque

La troisième condition impose la propriété suivante : c'est seulement à partir des bords gauche et droit de la configuration initiale (nécessairement dans $\mathcal{Q}$) que peuvent émerger des états de $\mathcal{Q}' \setminus \mathcal{Q}$ (grâce à des règles du type $\delta(\sharp, \sharp, a) = \ldots$ et $\delta(b, \sharp, \sharp) = \ldots$). En particulier le diagramme espace-temps de la machine simulante et celui de la machine simulée coïncident sur les sites de l'ensemble $\{(i, t) : t < \lfloor n/2 \rfloor, t < i < n - t\}$. Cette condition prend tout son sens dans les simulations par regroupement (*cf.* chapitre 3).

Les zones identiques dans la machine simulante et dans la machine simulée sont mises en évidence dans la figure 2.3.

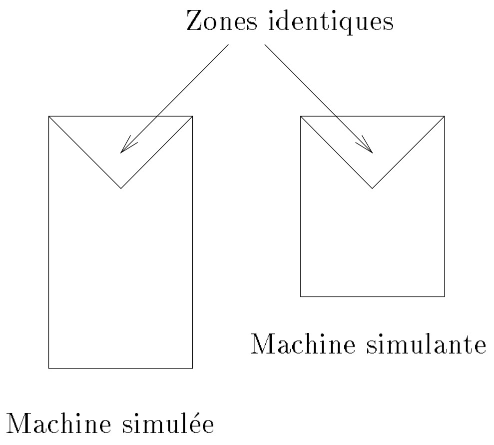


FIG. 2.3 – *Diagrammes espace-temps des machines simulante et simulée.*

Définition 2.8 (simulation pour calculateurs)

La notion de simulation pour calculateurs est définie de la même manière que pour les *reconnaisseurs. C'est une simulation faible si l'on autorise une fonction de renommage partiel de l'ensemble d'arrivée. C'est une simulation forte si l'on exige l'identité des ensembles d'arrivée.*

Remarques

- par la suite nous utiliserons essentiellement des simulations faibles pour les calculateurs, certains résultats de base (*cf.* lemme 2.3) semblant difficiles à obtenir en version forte ;
- puisque les reconnaisseurs sont des calculateurs particuliers, une simulation forte de calculateurs est *a fortiori* une simulation de reconnaisseurs (il suffit de prendre les symboles terminaux du calculateur comme symboles d'acceptation du reconnaisseur). Une simulation faible de calculateurs se transforme aisément en simulation (forte) de reconnaisseurs, il suffit de modifier toutes les règles du bord gauche qui donnent un état renommé acceptant ou rejetant ;
- il est possible de comparer deux méthodes de simulation S_1 et S_2 pour une machine M selon le nombre d'états et les temps de calculs respectifs des machines $S_1(M)$ et $S_2(M)$ (rapportés au temps de calcul de M), ainsi que les taux d'utilisation des nouveaux états (c'est-à-dire la densité, définie ci-après) ;
- on trouve également de nombreuses simulations dans les travaux de O. H. Ibarra *et al.*, [IPK85, IKM85], J. T. Butler [But74] et de A. R. Smith [Smi71a, Smi72].

Définition 2.9 (densité d'une simulation)
C'est le nombre d'occurrences d'états de $\mathcal{Q}' \setminus \mathcal{Q}$ lors d'un calcul. On l'exprime comme une fonction de la longueur n de la configuration initiale.

La fonction de densité de $S(M)$, c'est-à-dire la fonction qui, à l'entrée finie u associe le nombre d'occurrences d'états n'appartenant pas à $\mathcal{Q}$ dans le diagramme espace-temps de la machine $S(M)$ lancée sur l'entrée $c(u)$. La fonction de densité est linéaire (resp. quadratique) si elle est de la forme $\mathcal{O}(|u|)$ (resp. $\mathcal{O}(|u|^2)$) où n est la longueur de u. Remarquons que la densité ne peut pas excéder la valeur de $2|u|.t(u) - |u|^2/4$ pour les automates cellulaires bi-infinis ni la valeur $|u|.t(u) - |u|^2/4$ pour les automates cellulaires finis (voir figure 2.4).

Un grand nombre de simulations utilisent des techniques de regroupement qui permettent à une seule cellule de simuler le travail de plusieurs. Nous détaillerons ces techniques au fur et à mesure des besoins mais nous définissons dès maintenant la notion importante de configuration k-groupée (voir [Col69] pour l'utilisation classique d'une configuration k-groupée).

Définition 2.10 (configurations k-groupées)
Étant donnée une configuration $c = a_1 \dots a_n$ de longueur n sur un ensemble d'états $\mathcal{Q}$, ses k-groupées sont les configurations sur l'alphabet $\mathcal{Q}^k$ qui ont la forme suivante :

$$\sharp^k \dots \sharp^k . \underbrace{(\sharp \dots \sharp . a_1 \dots a_p)}_{\alpha} . (a_{p+1} \dots a_{p+k}) \dots \underbrace{(a_{n-l} \dots a_n, \sharp \dots \sharp)}_{\beta} . \sharp^k \dots \sharp^k$$

où α (resp. β) est le rang de la première (resp. dernière) cellule non totalement quiescente (c'est-à-dire différente de $\sharp^k$) de la ligne. On dit que la configuration k-groupée s'étend entre α et β.

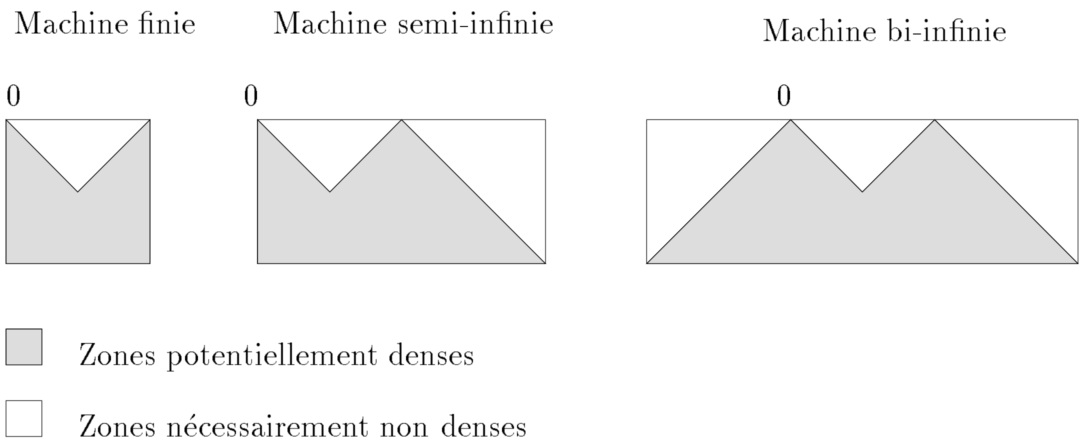


FIG. 2.4 – *Zones potentiellement denses d'un diagramme espace-temps.*

Remarquons que $\beta - \alpha$ appartient à $\{\lfloor n/k \rfloor - 1, \lfloor n/k \rfloor, \lfloor n/k \rfloor - 1\}$ et qu'il existe plusieurs configurations k-groupées pour α et β donnés. Souvent nous parlerons de *la* configuration k-groupée d'une configuration donnée, il s'agit alors de celle pour laquelle le premier k-uplet non entièrement quiescent commence par un état non quiescent. Cette configuration à pour forme :

$$\sharp^k \dots \sharp^k . \underbrace{(a_1 \dots a_k)}_{\alpha} . (a_{k+1} \dots a_{2k}) \dots \underbrace{(a_{n-l} \dots a_n, \sharp \dots \sharp)}_{\beta} . \sharp^k \dots \sharp^k$$

on a alors $\beta - \alpha = \lfloor n/k \rfloor - 1$

4.1 Simulation des reconnaisseurs bi-infinis

Dans les chapitres suivant nous allons exposer des résultats d'accélération valables pour divers modèles d'automates cellulaires. Malheureusement, nous ne connaissons pas de méthodes d'accélération directe pour les reconnaisseurs bi-infinis (voir définition 2.3). Nous exposons un lemme technique qui autorise la simulation d'un reconnaisseur bi-infini par un reconnaisseur semi-infini, sans perte de temps (mais au prix d'un accroissement quadratique du nombre d'états). Ainsi, des résultats comme l'accélération d'une constante et l'accélération linéaire pour les reconnaisseurs semi-infinis peuvent être transposés dans le monde des reconnaisseurs bi-infinis.

Lemme 2.2 (simulation Bi/Semi)
Tout reconnaisseur bi-infini à s états peut être simulé en temps réel par un reconnaisseur semi-infini à s^2 états.

Preuve
L'idée est la même que dans le cas des machines de Turing pour passer d'un ruban bi-infini à un ruban semi-infini. La figure 2.5 donne le principe de la simulation.

Nous illustrons l'idée générale sur un fragment de diagramme espace-temps obtenu pour $|u| = 4$ et pour t variant entre 0 et 3, avant de donner les règles de transition. La figure 2.6

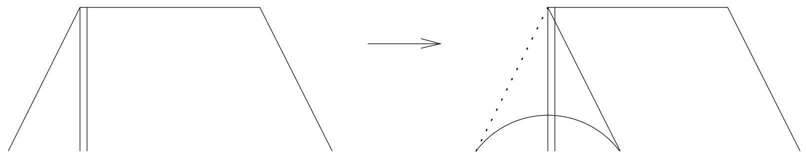


FIG. 2.5 – *Simulation d'un reconnaisseur bi-infini par un semi-infini (principe).*

montre un fragment du diagramme espace-temps d'un reconnaisseur bi-infini et sa version simulée sur un reconnaisseur semi-infini.

$\sharp$	$\sharp$	$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\sharp$	$\sharp$	$\sharp$
$\sharp$	$\sharp$	$\langle -1,1\rangle$	$\langle 0,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\sharp$	$\sharp$
$\sharp$	$\langle -2,2\rangle$	$\langle -1,2\rangle$	$\langle 0,2\rangle$	$\langle 1,2\rangle$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$	$\langle 5,2\rangle$	$\sharp$
$\langle -3,3\rangle$	$\langle -2,3\rangle$	$\langle -1,3\rangle$	$\langle 0,3\rangle$	$\langle 1,3\rangle$	$\langle 2,3\rangle$	$\langle 3,3\rangle$	$\langle 4,3\rangle$	$\langle 5,3\rangle$	$\langle 6,3\rangle$

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\sharp$	$\sharp$	$\sharp$
$\sharp$	$\langle 0,1\rangle\langle -1,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\sharp$	$\sharp$
$\sharp$	$\langle 0,2\rangle\langle -1,2\rangle$	$\langle 1,2\rangle\langle -2,2\rangle$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$	$\langle 5,2\rangle$	$\sharp$
$\sharp$	$\langle 0,3\rangle\langle -1,3\rangle$	$\langle 1,3\rangle\langle -2,3\rangle$	$\langle 2,3\rangle\langle -3,3\rangle$	$\langle 3,3\rangle$	$\langle 4,3\rangle$	$\langle 5,3\rangle$	$\langle 6,3\rangle$

FIG. 2.6 – *Fonctionnement d'un reconnaisseur bi-infini et simulation sur un reconnaisseur semi-infini.*

Nous indiquons maintenant la liste des règles de transition de l'automate cellulaire simulant R' (fonction δ') en fonction de celles de l'automate cellulaire simulé R (fonction δ).

Notation

Nous rappelons les conventions adoptées pour la description de règles de simulation :

- les lettres $a, b, c, \ldots$ représentent des états non quiescents ;
- les lettres $\ldots, x, y, z$ représentent des états éventuellement quiescents ou frontières;
- le plus souvent nous ignorerons le cas particulier de la ligne de longueur 1 où les triplets d'états sont toujours du type $(\sharp, a, \sharp)$ et où le calcul de l'automate cellulaire est celui d'un automate fini très simple (c'est-à-dire ne contenant que des ε-transitions) ;
- afin de les désigner plus facilement nous nommons les règles de transition (`Normal`, `Pliage`) ...

Ensemble des états

$\mathcal{Q}' = \mathcal{Q}^2 \cup \mathcal{Q}$.

Premiers pas

`Normal` : $\delta'(a, b, x) = \delta(a, b, x)$,

`Initialise`: $\delta'(\sharp, a, b) = (\delta(\sharp, a, b), \delta(\sharp, \sharp, a))$,
`Gauche1`: $\delta'(\sharp, (a, b), c) = (\delta(b, a, c), \delta(\sharp, b, a))$.

Cas général
`Gauche2`: $\delta'(\sharp, (a, b), (c, d)) = (\delta(b, a, c), \delta(d, b, a))$,
`Pliage1`: $\delta'((a, b), c, d) = (\delta(a, c, d), \delta(\sharp, \sharp, b))$,
`Pliage2`: $\delta'((a, b), (c, d), e) = (\delta(a, c, e), \delta(\sharp, d, b))$,
`Pliage3`: $\delta'((a, b), (c, d), (e, f)) = (\delta(a, c, e), \delta(f, d, b))$.

Dernier pas
La règle `Gauche2` peut être modifiée pour que $\delta'(\sharp, (a, b), (c, d)) = \delta(b, a, c)$ si $\delta(b, a, c)$ est un état terminal.

□

Remarque
Le dernier pas éventuel doit être spécial pour produire un résultat dans $\mathcal{S}_a \cup \mathcal{S}_r$, et non dans $(\mathcal{S}_a \cup \mathcal{S}_r) \times \mathcal{Q}$;

4.2 Ralentissement des machines

Nous rappelons ici quelques résultats simples concernant le ralentissement des machines ; ils sont utiles, par exemple, pour modifier et combiner des temps de synchronisation (voir chapitre 7).

Lemme 2.3 (ralentissement d'une constante)
Soit k un entier strictement positif.

Tout calculateur à s états, travaillant en temps $t(u)$ sur l'entrée u peut être simulé faiblement par un calculateur de même nature à $(k+1).s$ états, travaillant en temps $k + t(u)$.

Tout reconnaisseur à s états, travaillant en temps $t(u)$ sur l'entrée u peut être simulé par un reconnaisseur de même nature à $2k + s$ états, travaillant en temps $k + t(u)$.

Preuve
Nous utilisons ici des techniques classiques de simulation d'automates cellulaires. Ce ne sera pas le cas pour la version forte du lemme où nous utiliserons une instance très simple de la méthode de regroupement (décrite formellement au chapitre 3).

Pour les calculateurs, nous rappelons simplement la preuve de [CČ83] consistant à conserver la configuration initiale pendant k étapes avant de commencer le calcul.

Ensemble des états
$\mathcal{Q}' = \mathcal{Q} \cup (\{1, \ldots, k\} \times \mathcal{Q})$.

Règles de transition
`Compte1`: $\delta'(a, b, c) = (1, b)$,
`CompteI`: $\delta'((i, a), (i, b), (i, c)) = (i + 1, b)$ (pour $1 \leq i < k$) ,
`Simule`: $\delta'((k, a), (k, b), (k, c)) = (k, \delta(a, b, c))$.
Plus les règles analogues associées aux bords droit et gauche.

Pour les reconnaisseurs possédant un seul état d'acceptation et un seul état de rejet (on peut se ramener à ce cas en considérant que les états d'acceptation sont équivalents entre eux et de même pour les états de rejet), il existe une preuve plus économe en nombre d'états:

Ensemble des états

$\mathcal{Q}' = \mathcal{Q} \cup \{0, \ldots, k\} \cup \{0', \ldots, k'\}$.

Règles de simulation

`Normal`: $\delta'(a, b, c) = \delta(a, b, c)$,

$$\delta'(\sharp, a, b) = \begin{cases} 0 \text{ si } \delta(\sharp, a, b) \in \mathcal{S}_a \\ 0' \text{ si } \delta(\sharp, a, b) \in \mathcal{S}_r \\ \delta(\sharp, a, b) \text{ sinon} \end{cases}$$

`OuiI`: $\delta'(\sharp, i, a) = i + 1$ pour $(1 \leq i < k)$,

`NonI`: $\delta'(\sharp, i', a) = (i + 1)'$ pour $(1 \leq i < k)$,

`DernierOui`: $\delta'(\sharp, (k - 1), a) = oui$ (où *oui* est l'état d'acceptation) ,

`DernierNon`: $\delta'(\sharp, (k - 1)', a) = non$ (où *non* est l'état de rejet) .

□

Lemme 2.4 (ralentissement d'une constante, version forte)

Tout calculateur à s états travaillant en temps $|u| - 1 + t(u)$ peut être simulé fortement par un calculateur à $\mathcal{O}(s^2)$ états travaillant en temps $|u| - 1 + t(u) + k$ (avec le même ensemble d'arrivée donc).

Remarques

- ce lemme fonctionne *a fortiori* pour les reconnaisseurs. Dans ce cas particulier, la simulation fonctionne même si le temps de calcul de la machine initiale sur le mot u dure moins de $|u|$ étapes et l'on peut alors énoncer le lemme en utilisant la notation $T(u)$: Tout reconnaisseur travaillant en temps $T(u)$ peut être simulé par un reconnaisseur travaillant en temps $T(u) + k$;
- quant à la méthode de preuve de ce lemme : nous n'avons pas encore introduit formellement la notion d'automate cellulaire regroupeur (c'est l'objet du chapitre 3), mais la simulation utilisée pour prouver le lemme précédent est un regroupement extrêmement simple, qui peut être appréhendé sans faire appel au formalisme général. Un autre exemple simple de regroupement sera donné au chapitre 3, cette fois avec la terminologie adaptée.

Preuve

Nous donnons simplement l'idée de la preuve pour $k = 1$, les autres cas s'en déduisant directement.

Dès le temps 1, la cellule de numéro 0 est ralentie. Puis le ralentissement se propage de gauche à droite, d'une cellule par unité de temps. Les cellules reprennent un comportement normal (c'est-à-dire conforme à celui de l'automate cellulaire initial) après avoir subi un ralentissement, sauf la cellule 0 qui mémorise une information supplémentaire (symbolisée par « $*$ » dans les règles ci-après). Cette information permet de ne pas initier un nouveau processus de ralentissement à partir de la gauche dès la troisième étape. Pour tout calcul non trivial, c'est-à-dire prenant au moins $|u|$ étapes sur l'entrée $c(u)$, la configuration juste avant le dernier pas est donc la même que celle de la machine initiale un temps plus tôt pour toutes les cellules d'indice plus grand que 0. Le symbole « $*$ » est effacé lors du dernier pas de calcul.

Ensemble d'états

$\mathcal{Q}' = \mathcal{Q} \cup \mathcal{Q} \times \{*\} \cup \mathcal{Q}^2$.

Règles de simulation générales

`Simule1`: $\delta'(a, x, y) = \delta(a, x, y)$,
`Simule2`: $\delta'(a*, x, y) = \delta(a, x, y)$.

Règles concernant le bord gauche

`Initialise1`: $\delta'(\sharp, a, b) = (a, b)$,
`Initialise2`: $\delta'(\sharp, (a, b), c) = \delta(\sharp, a, b)*$,
`Gauche`: $\delta'(\sharp, a*, b) = \begin{cases} \delta(\sharp, a, b) \text{ si } \delta(\sharp, a, b) \in \mathcal{S}_a \cup \mathcal{S}_r \\ \delta(\sharp, a, b)* \text{ sinon} \end{cases}$

Propagation du ralentissement

`Ralentis1`: $\delta'((a, b), c, d) = (c, d)$,
`Ralentis2`: $\delta'(a, (b, c), d) = \delta'(a*, (b, c), d) = \delta(a, b, c)$,
`Ralentis3`: $\delta'(a, b, (c, d)) = \delta'(a*, b, (c, d)) = \delta(a, b, c)$,
`Droite`: $\delta'((a, b), c, \sharp) = \delta'((a, b)*, c, \sharp) = c$.

La figure 2.7 montre un fragment de diagramme espace-temps illustrant la méthode utilisée (pour l'exemple, on suppose $|u| = 5$ et $|u| + t(u) = 9$).

Remarque

On constate que la version forte du lemme utilise plus de ressources que la version faible proposée par Choffrut et Čulik. Pour la version forte, notre opinion est qu'il n'existe pas de méthode exigeant moins d'un accroissement polynomial du nombre d'états.

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$
$\sharp$	$\langle 0,0\rangle\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$
$\sharp$	$\langle 0,1\rangle*$	$\langle 1,1\rangle\langle 2,1\rangle$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$
$\sharp$	$\langle 0,2\rangle*$	$\langle 1,2\rangle$	$\langle 2,2\rangle\langle 3,2\rangle$	$\langle 3,3\rangle$	$\langle 4,3\rangle$
$\sharp$	$\langle 0,3\rangle*$	$\langle 1,3\rangle$	$\langle 2,3\rangle$	$\langle 3,3\rangle\langle 4,3\rangle$	$\langle 4,4\rangle$
$\sharp$	$\langle 0,4\rangle*$	$\langle 1,4\rangle$	$\langle 2,4\rangle$	$\langle 3,4\rangle$	$\langle 4,4\rangle$
$\sharp$	$\langle 0,5\rangle*$	$\langle 1,5\rangle$	$\langle 2,5\rangle$	$\langle 3,5\rangle$	$\langle 4,5\rangle$
$\sharp$	$\langle 0,6\rangle*$	$\langle 1,6\rangle$	$\langle 2,6\rangle$	$\langle 3,6\rangle$	$\langle 4,6\rangle$
$\sharp$	$\langle 0,7\rangle*$	$\langle 1,7\rangle$	$\langle 2,7\rangle$	$\langle 3,7\rangle$	$\langle 4,7\rangle$
$\sharp$	$\langle 0,8\rangle$	$\langle 1,8\rangle$	$\langle 2,8\rangle$	$\langle 3,8\rangle$	$\langle 4,8\rangle$

FIG. 2.7 – *Ralentissement d'un pas d'un calculateur.*

□

Lemme 2.5 (ralentissement d'un facteur, version faible)

Soit k un entier strictement positif.

Tout reconnaisseur à s états, travaillant en temps $t(u)$ sur l'entrée u peut être simulé par un reconnaisseur de même nature à $k.s$ états, travaillant en temps $k.t(u)$.

Le même résultat est valable pour les calculateurs.

Preuve

l'idée est très simple : il s'agit de simuler un pas de calcul réel toutes les k étapes. La preuve est valable pour les reconnaisseurs et pour les calculateurs puisque la configuration finale est exactement la même que dans la machine simulée.

Ensemble d'états

$\mathcal{Q}' = \mathcal{Q} \times \{1, \ldots, k-1\} \cup \mathcal{Q}$.

Principales règles de simulation

`Compte1` : $\delta'(a, b, c) = (b, 1)$,
`CompteI` : $\delta'((a, i), (b, i), (c, i)) = (b, i+1)$ (pour $1 < i < k-1$) ,
`Simule` : $\delta'((a, k-1), (b, k-1), (c, k-1)) = \delta(a, b, c)$.

□

Remarque

Dans l'énumération des règles ci-dessus, nous omettons les cas particuliers aux bords gauches et droits.

5 Synchronisation d'une ligne de fusiliers

Nous présentons ici un bref état de l'art du problème de la synchronisation d'une ligne de fusiliers (où encore FSSP pour *Firing Squad Synchronization Problem*) en dimension 1.

Certains résultats énoncés ici sous forme de lemmes sont directement utilisés dans les sections suivantes.

Temps de synchronisation

Dans le cadre des automates cellulaires simples, on dispose d'une large variété de temps de synchronisation. Plus la panoplie est étendue, plus il est possible d'enchaîner des calculs, puisque certaines méthodes de composition des automates cellulaires reposent sur une synchronisation. Nous donnons une liste des principaux temps de synchronisation connus, avec la liste des références associées (remarquons qu'en l'absence de généraux explicites, les temps de synchronisation minimaux doivent être augmentés d'une unité).

Temps minimal

La synchronisation est réalisée avec un général explicite placé à une extrémité de la ligne en $2n-2$ étapes de calcul. E. Goto (non publié), A. Waksman [Wak66], R. Balzer [Bal67a] et J. Mazoyer [Maz87] ont proposé de telles solutions. À partir de deux généraux explicites placés aux deux extrémités de la ligne, le temps minimal est de $n-1$ étapes. J. Mazoyer et N. Reimen [MR92] ont proposé une telle solution utilisant comme « sous-programme » une solution minimale à un seul général. Nous l'utilisons à plusieurs reprises dans les chapitres suivants. Enfin, Moore et Langton [ML68] et Čulik [Čul89] ont proposé des solutions minimales pour des variantes du problème initial.

Temps non minimal

Ici la panoplie est très large mais on distingue deux techniques de production de solutions en temps non minimal : par construction directe (voir par exemple M. Minsky [Min67], E.F. Moore [Moo64]), J.B. Yunès [Yun93]), ou alors par modifications et opérations sur des solutions antérieures (voir J. Mazoyer [Maz90]). Notre contribution à l'étude des temps de synchronisation apparaît au chapitre 7 où nous prouvons que la classe des temps de synchronisation est fermée par composition avec tout polynôme à coefficients rationnels (pour peu qu'il excède un certain minimum). Au chapitre 8 nous fournissons également une solution synchronisante partielle à 6 états fonctionnant en temps $3n$.

Nombre d'états

Une préoccupation importante est la recherche de solutions synchronisantes à peu d'états. D'un point de vue pratique, minimiser le nombre d'états permet d'obtenir des automates cellulaires plus petits (plus simples à fabriquer). D'un point de vue théorique, la recherche de petites solutions permet de mieux comprendre la structure du FSSP.

La figure 2.8 résume l'état de la recherche concernant les solutions en temps minimal. Les nombres d'états sont valables pour des solutions synchronisantes traditionnelles, c'est-à-dire avec généraux explicites. Si les généraux sont implicites il faut augmenter de 1 toutes les valeurs du tableau (*cf.* remarques suivant la définition 2.5).

Nombre d'états	Pour tout n	Pour une infinité de n
2	Non	Non
3	Non (Balzer [Bal67b], Yunès [Yun94])	?
4	Non (Balzer [Bal67b], Yunès [Yun94])	?
5	?	Oui (Mazoyer)
6 (ou plus)	Oui (Mazoyer [Maz87])	Oui

FIG. 2.8 – *Solutions synchronisantes en temps minimal.*

La figure 2.9 concerne les solution en temps non minimal (notamment celles de Minsky en temps proche de $3n$).

Nombre d'états	Pour tout n	Pour une infinité de n
2	Non	Non
3	Non (Yunès [Yun94])	?
4	?	?
5	?	?
6	?	Oui (même avec signaux larges)
7 (ou plus)	Oui (Yunès [Yun94])	Oui

FIG. 2.9 – *solutions synchronisantes en temps quelconque.*

Des tableaux similaires peuvent exister pour les solutions à deux généraux. Néanmoins, celles-ci sont souvent exprimées comme une juxtaposition de deux (demi) synchronisations à un seul général.

Remarque

La recherche d'une solution au FSSP en temps 3n avec 6 états est la motivation initiale du chapitre 8.

Nombre de communications

Si l'on compte comme une communication entre deux cellules un échange d'états significatifs (disons non quiescents), on sait mettre en évidence une différence structurelle entre deux grandes familles de solutions synchronisantes en temps linéaire (*cf.* [Hee92]) :

- les solutions planaires utilisent $\mathcal{O}(n^2)$ communications. Ce sont par exemple les synchroniseurs en temps minimal de Balzer [Bal67a] ou de Mazoyer [Maz87] ;
- les solutions filiformes en temps $3n$ utilisent $\mathcal{O}(n \log n)$ communications. Ce sont par exemple les synchroniseurs de type Minsky [Min67, Yun94].

Chapitre 3

Le regroupement

Résumé

La plupart des simulations données dans les chapitres 4, 5 et 6 peuvent s'exprimer comme la simple composition d'un automate cellulaire initial et d'un automate cellulaire « déformant » les diagrammes espace-temps. De tels automates cellulaires sont appelés regroupeurs et doivent vérifier certaines conditions que nous énumérons dans ce chapitre.

Afin d'illustrer la méthode, nous proposons un exemple complet et caractéristique : la transformation d'une configuration quelconque en configuration groupée. C'est cet exemple qui donne son nom à la méthode.

Introduction

Un grand nombre de problèmes sur lignes d'automates se résolvent par simulation ; outre les problèmes d'accélération abordés dans cette thèse, on trouve des questions liées à la réversibilité [Mor95, Dur96b, Dub93], à l'universalité [LN90, Mar94] sans oublier les simulations entre automates cellulaires de voisinages ou topologies différents [CČ84, CČ83].

La démarche est la suivante : à partir d'un automate cellulaire quelconque, on cherche un nouvel automate cellulaire, effectuant le même calcul mais présentant une propriété supplémentaire (il « va plus vite », il est réversible) .... Il est de bon ton que le dispositif transformant la machine initiale en la machine simulante ait une faible complexité (et si possible très inférieure à celle de la machine initiale).

La difficulté dans ce type de travail réside premièrement dans le *design* du dispositif de transformation, deuxièmement dans la preuve qu'il est correct. Nous pensons que le premier point restera toujours difficile, puisqu'il contient la part de compréhension du problème et d'imagination. En revanche, le deuxième point se prête mieux à la formalisation (et même à la mécanisation). C'est un des buts de notre méthode de regroupement.

Par ailleurs, les simulation usuelles utilisent toujours la fonction de transition de la machine initiale (pour effectuer des calculs sur les états) et une autre fonction, réalisable par automate cellulaire, qui à pour but de « mélanger » l'information contenue dans différents sites du diagramme espace-temps. Hélas, dans le formalisme usuel, on ne présente que la composée de ces deux fonctions (sans nécessairement expliciter la seconde).

Le second but de notre méthode de regroupement est de fournir un cadre formel pour l'expression des fonctions simulantes *indépendamment* de la fonction simulée. Chacune de ces fonctions est, bien entendu, réalisable par un automate cellulaire et la machine simulante complète s'exprime le produit des deux.

1 Définitions

1.1 Notions de base

Nous introduisons ici les notions permettant la définition formelle d'un *automate regroupeur.*

Tout d'abord, pour un ensemble de sites donné, ses *conséquences* sont les sites dont on peut connaître l'état. En d'autres termes, la fonction conséquences répond à la question « que peut-on déduire (avec certitude) d'un ensemble de sites? »

La figure 3.1 illustre cette définition.

Définition 3.1 (conséquences)
Soit X un ensemble de sites. L'ensemble des conséquences immédiates strictes de X, noté $\mathrm{csq}(X)$ *est défini par :* $\mathrm{csq}(X) = \{(i,t) : (i,t) \notin X, (i-1,t-1) \in X, (i,t-1) \in X, (i+1,t-1) \in X\}$*. L'ensemble des conséquences globales (larges) de X, noté* $\mathrm{CSQ}(X)$*, est la clôture de la fonction* csq*, définie par :*

$$\mathrm{CSQ}(X) = \bigcup_{s\in\mathbb{N}} X_s \text{ où } X_0 = X \text{ et } X_{s+1} = X_s \cup \mathrm{csq}(X_s)$$

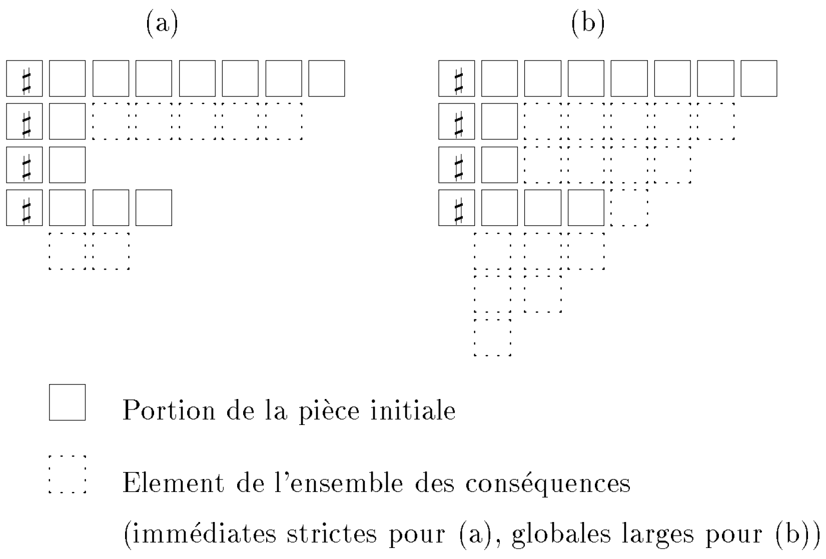


FIG. 3.1 – *Notion de conséquences.*

Nous introduisons maintenant la notion de *pièce*. C'est un ensemble (fini ou non) de sites contenant au moins le site $(0,0)$ appelé l'origine de la pièce. Ce site doit être « le plus à gauche des sites les plus hauts » dans la pièce.

Définition 3.2 (pièce)

Une pièce finie est une partie finie X de $\mathbb{Z} \times \mathbb{N}$ telle que $(0,0) \in X$ et $\inf\{i : (i,0) \in X\} = 0$. La pièce finie $\{(0,0)\}$ est représentée graphiquement par le symbole $\square$. La pièce infinie « bord gauche » représentée graphiquement par le symbole $\boxed{\text{G}}$ est définie par $G = \{(i,j) : i \leq 0, j \geq 0\}$. La pièce infinie « bord droit » représentée graphiquement par le symbole $\boxed{\text{D}}$ est définie par $D = \{(i,j) : i \geq 0, j \geq 0\}$.

Une pièce est soit une pièce finie, soit $\boxed{\text{G}}$, soit $\boxed{\text{D}}$, soit $\boxed{\text{G}} \cup (X + (\alpha, 0))$ soit $\boxed{\text{D}} \cup (X + (\alpha, 0))$ (où X est une pièce finie et α un entier positif ou nul).

La figure 3.2 indique les conventions pour la représentation des pièces. Dans les cas où aucune ambigüité n'est possible, on pourra simplifier la représentation des bords (symboles $\boxed{\text{G}}$et $\boxed{\text{D}}$) en les désignant tous deux par le symbole $\boxed{\$}$.

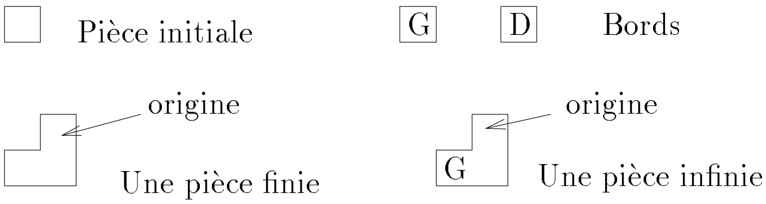


FIG. 3.2 – *Représentation des pièces.*

Une notion centrale pour la définition du regroupement est celle d'agencement de pièces ; informellement, il s'agit d'une liste de pièces *positionnées* les unes par rapport aux autres à l'aide de vecteurs.

Définition 3.3 (k-agencement, agencement F-correct, famille rigide)

- *un k-agencement est défini par un k-uplet de pièces et un $(k-1)$-uplet de vecteurs de $\mathbb{Z}^2$;*
- *soit F une famille de k-agencements et l un entier supérieur à k, le l-agencement défini par $(P_0, \ldots, P_{l-1}, v_1, \ldots, v_{l-1})$ est F-correct si :*

$$\forall j \in \{0, \ldots, l-k\}, (P_j, \ldots, P_{j+k-1}, v_{j+1}, \ldots, v_{j+k-1}) \in F$$

- *une famille F de k-agencements est dite* ***rigide*** *si elle possède la propriété suivante :*

$$\text{si } (P_0, \ldots, P_{k-1}, v_1, \ldots, v_{k-1}) \in F \text{ et } (P_0, \ldots, P_{k-1}, w_1, \ldots, w_{k-1}) \in F \text{ alors } v_i = w_i \text{ (pour tout } i).$$

Remarques

- les $k-1$ vecteurs d'un k-agencement permettent de positionner les k pièces les unes par rapport aux autres : v_{i+1} est le vecteur qui envoie l'origine de P_i sur l'origine de P_{i+1} ;
- un l-agencement est F-correct si tous les k-agencements qui le constituent sont bien dans F ;

- une famille rigide est une famille où les pièces de tout k-uplet sont placées sans ambigüité les unes par rapport aux autres. Elle peut être décrite par un ensemble Φ de k-uplets de pièces et par une application $\omega = (\omega_1, \ldots, \omega_k - 1) : \Phi \to (\mathbb{Z}^2)^{k-1}$ qui calcule de proche en proche les positions des pièces : $\omega(P_0, \ldots, P_{k-1}) = (v_1, \ldots, v_{k-1})$;
- les familles que nous utilisons pour définir le regroupement (voir ci-après) sont rigides ;
- si F est une famille rigide de 3-agencements, décrite par Φ et ω alors le l-agencement $(P_0, \ldots, P_{l-1}, v_1, \ldots, v_{l-1})$ est F-correct si et seulement si :
  1. $(P_i, P_{i+1}, P_{i+2}) \in \Phi$ pour tout $i \in \{0, \ldots, l-3\}$,
  2. $\omega_2(P_i, P_{i+1}, P_{i+2}) = \omega_1(P_{i+1}, P_{i+2}, P_{i+3})$ pour tout $i \in \{0, \ldots, l-4\}$.

Lemme 3.1 (familles rigides)
Si F est une famille rigide de k-agencements et si $l \geq k$, alors la famille des l-agencements F-corrects est rigide.

Preuve
Supposons qu'il existe deux l-agencements différents :

$$(P_0, \ldots, P_{l-1}, v_1, \ldots, v_i, \ldots, v_{l-1}) \text{ et } (P_0, \ldots, P_{l-1}, w_1, \ldots, w_i, \ldots, w_{l-1}), \text{ avec } v_i \neq w_i.$$

Puisque la famille est F-correcte, les deux k-agencements :

$$(P_{i-j}, \ldots, P_{k+i-j-1}, v_{i-j+1}, \ldots, v_i, \ldots, v_{l-i-j-1})$$
$$\text{et}$$
$$(P_{i-j}, \ldots, P_{k+i-j-1}, w_{i-j+1}, \ldots, w_i, \ldots, v_{l-i-j-1})$$
$$\text{(où } j \text{ est choisi de sorte que } i - j \geq 0 \text{ et } k + i - j - 1 \leq l - 1)$$

sont dans F d'une part et $v_i \neq w_i$ est toujours vrai d'autre part ; ceci contredit la rigidité de la famille F.

□

Remarque
si l'ensemble de pièces $\mathcal{Q}$ est fini alors toute famille rigide de k-agencements sur $\mathcal{Q}$ est également finie (de taille au plus $k^{Card(\mathcal{Q})}$).

1.2 Automate cellulaire regroupeur

En nous appuyant sur les notions précédentes nous pouvons donner les définitions d'automate cellulaire regroupeur et les opérations qui s'y rapportent.

Intuitivement, un regroupeur est un automate cellulaire dont les états peuvent être interprétés comme des pièces et dont le comportement est compatible avec la rigidité d'une famille de 3-agencements sur ces pièces.

La définition formelle énoncée ci-après peut paraître technique. Elle est suivie de remarques qui mettent en avant l'intuition et le caractère essentiellement géométrique de cette notion. Rappelons que la translatée d'une portion P du plan selon un vecteur v est simplement notée $P + v$.

Définition 3.4 (automate cellulaire regroupeur)
Soient les trois éléments suivants :

1. *un automate cellulaire* $(\mathcal{Q}, \delta)$ *dont l'ensemble des états* $\mathcal{Q}$ *est une famille finie de pièces et dont la fonction de transition* δ *est définie sur* $\Phi \subset \mathcal{Q}^3$ *;*

2. *une famille rigide* F *de 3-agencements des pièces de* $\mathcal{Q}$ *constituée des triplets de* Φ *et de leurs vecteurs de placement (fonctions* ω_1 *et* ω_2*) ;*

3. *une fonction* $\theta : \Phi \to \mathbb{Z}^2$*, dite de placement.*

Le triplet $M = ((\mathcal{Q}, \delta), F, \theta)$ *est un* ***regroupeur*** *si les trois conditions suivantes sont vérifiées :*

Départ

- *Les états* $\boxed{\text{G}}$, $\square$ *et* $\boxed{\text{D}}$ *sont dans* $\mathcal{Q}$*, et sont quiescents ;*

- *les 3-agencements suivants sont dans* F *:*
  $(\boxed{\text{G}}, \boxed{\text{G}}, \boxed{\text{G}}, \vec{e}, \vec{e}), (\boxed{\text{G}}, \boxed{\text{G}}, \square, \vec{e}, \vec{e}),$
  $(\boxed{\text{G}}, \square, \square, \vec{e}, \vec{e}), (\square, \square, \square, \vec{e}, \vec{e}),$
  $(\square, \square, \boxed{\text{D}}, \vec{e}, \vec{e}), (\square, \boxed{\text{D}}, \boxed{\text{D}}, \vec{e}, \vec{e}),$
  $(\boxed{\text{D}}, \boxed{\text{D}}, \boxed{\text{D}}, \vec{e}, \vec{e}), (\boxed{\text{G}}, \square, \boxed{\text{D}}, \vec{e}, \vec{e})$ *;*

Faisabilité
Pour tout 3-agencement $(P_0, P_1, P_2, v_1, v_2)$ *de* F *:*

$$\delta(P_0, P_1, P_2) + v_1 + \theta(P_0, P_1, P_2) \subseteq \mathrm{CSQ}(P_0 \cup (P_1 + v_1) \cup (P_2 + v_2))$$

Cohérence
Si $(P_0, \ldots, P_4, v_1, \ldots, v_4)$ *est un 5-agencement* F*-correct alors le 3-agencement ci-dessous est dans* F *:*

$$(\delta(P_0, P_1, P_2), \delta(P_1, P_2, P_3), \delta(P_2, P_3, P_4), v_2 + \theta(P_1, P_2, P_3) - \theta(P_0, P_1, P_2),$$
$$v_3 + \theta(P_2, P_3, P_4) - \theta(P_1, P_2, P_3))$$

Les remarques suivantes ont pour but de clarifier cette définition.

Remarques

- La condition de départ assure simplement qu'un regroupeur peut simuler le comportement d'un automate cellulaire quelconque. En particulier, pour tout $n \geq 0$, la configuration $\boxed{\text{G}}.\square^n.\boxed{\text{D}}$ est F-correcte, et donc non bloquante. En fait, il existe un regroupeur trivial (dit identité) qui simule exactement l'automate initial ; ce regroupeur ne contient que les pièces $\boxed{\text{G}}$, $\square$ et $\boxed{\text{D}}$.

- Clarifions maintenant la notion de faisabilité : un regroupeur associe à trois pièces correctement positionnées une nouvelle pièce, dont la forme est calculée par la fonction de transition (δ) et dont la position relativement aux trois pièces de départ est indiquée par la fonction de placement (θ). La condition de faisabilité dit simplement que le contenu de la pièce calculée par δ se déduit bien par un calcul d'automate cellulaire du contenu des trois pièces sur lesquelles opère δ. La figure 3.3 illustre cette condition de faisabilité.

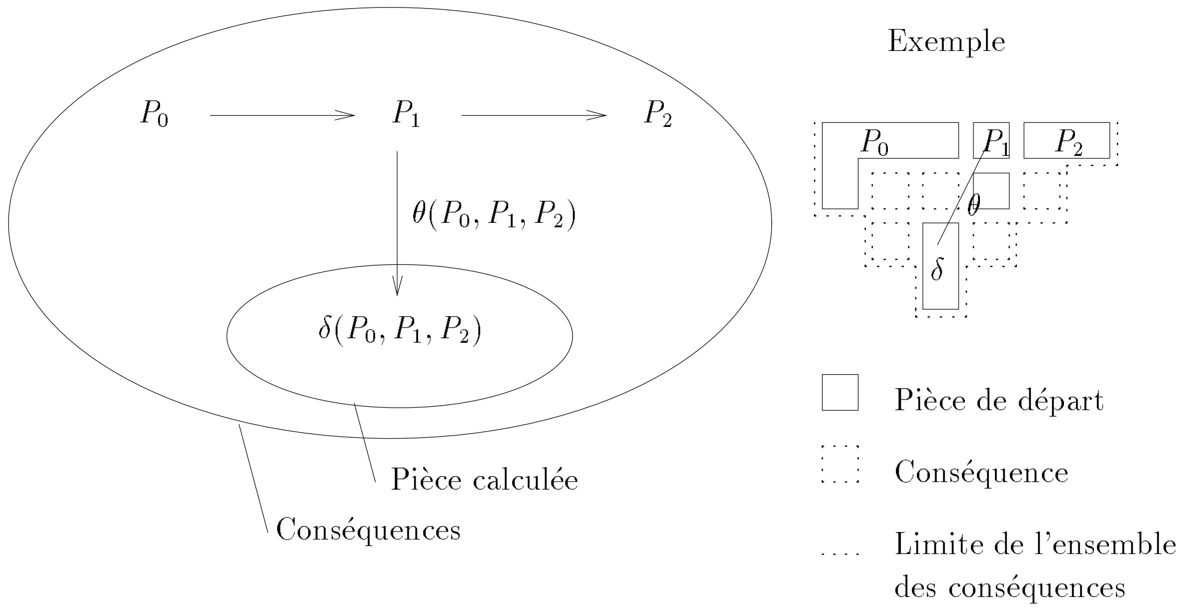


FIG. 3.3 – *La condition de faisabilité pour les regroupeurs.*

– La condition de cohérence assure qu'une configuration localement F-correcte (c'est-à-dire une suite de pièces positionnées de proche en proche par les fonctions ω_1 et ω_2) est transformée par le regroupeur en une configuration qui est encore F-correcte. Ainsi, le calcul du regroupeur peut se poursuivre de configurations en configurations. La figure 3.4 illustre cette notion de cohérence. Sur la figure 3.4 le vecteur $\omega_1(\delta(P_0, P_1, P_2), \delta(P_1, P_2, P_3), \delta(P_2, P_3, P_4)))$ vaut $v_2 + \theta(P_1, P_2, P_3) - \theta(P_0, P_1, P_2)$ (mais attention, en général $\omega_1(\delta(P_0, P_1, P_2), \delta(P_1, P_2, P_3), \delta(P_2, P_3, P_4))) \neq v_2$). Pareillement, le vecteur $\omega_2(\delta(P_0, P_1, P_2), \delta(P_1, P_2, P_3), \delta(P_2, P_3, P_4))$ vaut $v_3 + \theta(P_2, P_3, P_4) - \theta(P_1, P_2, P_3)$ (mais là aussi, il est généralement différent de v_3).

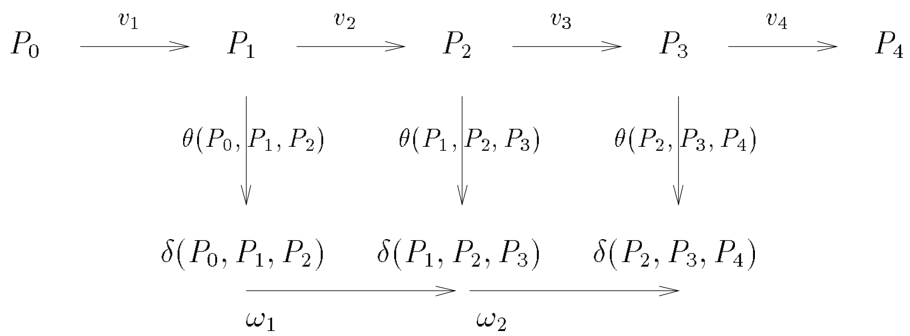


FIG. 3.4 – *Notion de cohérence.*

Cohérence faible

Pour les cas étudiés dans cette thèse, la configuration initiale du regroupeur est toujours la même : $\boxed{\text{G}}.\square^n.\boxed{\text{D}}$ (c'est indispensable dans le cadre de simulations, puisque les configurations de départ des machines simulée et simulante doivent être strictement identiques). On peut alors restreindre la vérification de la cohérence aux *seules* configurations atteignables à partir de la configuration de départ imposée (les autres ne nous intéressent pas puisqu'elles n'apparaissent jamais).

Configuration de regroupeur

Nous revenons maintenant sur la notion de *configuration*, commune à tous les automates cellulaires, dans le cas particulier des regroupeurs. Une configuration de regroupeur est une suite localement F-correcte de pièces. En pratique, nous nous restreignons aux configurations finies, du type :

$$\ldots\boxed{\text{G}}\ldots\boxed{\text{G}}.P_0\ldots P_{n-1}.\boxed{\text{D}}\ldots\boxed{\text{D}}\ldots$$

où les P_i ne sont ni $\boxed{\text{G}}$ ni $\boxed{\text{D}}$ quand i varie entre 0 et $n-1$, où les bords gauches (resp. droits) sont numérotés avec des indices négatifs (resp. supérieurs à $n-1$) et où les vecteurs plaçant les pièces les unes par rapport aux autres sont implicites (puisque la famille F est supposée rigide).

Remarques

- la configuration initiale $\ldots\boxed{\text{G}}\ldots\boxed{\text{G}}.\square\ldots\square.\boxed{\text{D}}\ldots\boxed{\text{D}}\ldots$ imposée dans le cadre d'une simulation, est F correcte puisque tous les triplets qu'elle contient sont dans F grâce à la condition de **départ** des regroupeurs.
- un automate cellulaire regroupeur étant, en particulier, un automate cellulaire « normal », toutes les autres notions définies au chapitre 2 sont encore valides, notamment celle de fonction de transition globale (Δ).

Proposition 3.2

Si c est une configuration pour le regroupeur $M = ((\mathcal{Q}, \delta), F, \theta)$, alors $\Delta(c)$ est encore une configuration.

Preuve

C'est une conséquence directe de la condition de cohérence qui précise justement que tout 5-agencement F-correct est transformé par application de Δ en un 3-agencement de F et donc toute la configuration obtenue est également F-correcte, grâce au lemme des familles rigides (lemme 3.1).

$\square$

Jusqu'à maintenant, un automate cellulaire regroupeur M est un automate cellulaire simple muni de quelques conditions particulières (faisabilité, cohérence). En fait ces conditions autorisent une *interprétation géométrique* du calcul de M : par plongement dans le demi-plan discret on fait correspondre à un état de M la translatée d'une pièce du demi-plan. Nous définissons formellement cette notion, qui est le fondement de l'intuition de la définition de regroupeur donnée plus haut.

Définition 3.5 (plongement d'un diagramme dans $\mathbb{Z} \times \mathbb{N}$)
Soient $M = ((\mathcal{Q}, \delta), F, \theta)$ un regroupeur et $\langle\, \rangle_M$ la fonction diagramme espace-temps associée à M (la pièce $\langle i, t\rangle_M$ est l'état de la cellule i au temps t), on définit le plongement $\pi : \mathbb{Z} \times \mathbb{N} \rightarrow \mathcal{P}(\mathbb{Z} \times \mathbb{N})$ où $\mathcal{P}(\mathbb{Z} \times \mathbb{N})$ désigne l'ensemble des parties de $\mathbb{Z} \times \mathbb{N}$ par :

$$\pi(i,t) = < i, t > + v_{i,t} \text{ où } \left\{ \begin{array}{lcl} v_{i,0} & = & (0,0) \text{ si } i \leq 0 \\ v_{i+1,0} & = & v_{i,0} + \omega_1(\langle i, 0\rangle_M, \langle i+1, 0\rangle_M, \langle i+2, 0\rangle_M \\ v_{i,t+1} & = & v_{i,t} + \theta(\langle i-1, t\rangle_M, \langle i, t\rangle_M, \langle i+1, t\rangle_M) \end{array} \right.$$

La figure 3.5 montre comment le plongement π envoie un site du regroupeur dans le demi-plan.

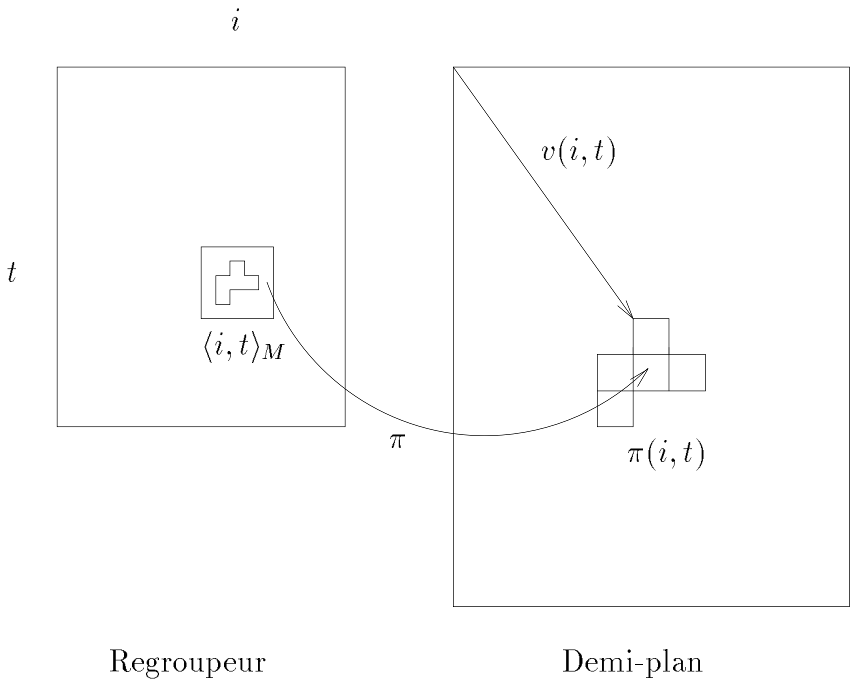


FIG. 3.5 – *Plongement du diagramme espace-temps d'un regroupeur dans le demi-plan discret.*

Fait 3.3 *La rigidité de la famille F de 3-agencements jointe à la condition de cohérence permet de connaître les valeurs de π à partir des valeurs $\langle i, t\rangle_M$ et de la donnée pour tout temps t du* seul *vecteur $v_{0,t}$.*

Ce fait s'avère très utile en pratique pour l'écriture d'équations de récurrences sur les propriétés d'un regroupeur : on peut se contenter de donner la liste des pièces au temps t et la valeur de $v_{t,0}$ sans préciser les valeurs des autres vecteurs de placement $v_{i,t}$ qui s'en déduisent implicitement. L'écriture est donc allégée.

Produit d'un automate cellulaire par un regroupeur

Nous définissons maintenant la notion essentielle de produit d'un automate cellulaire quelconque par un regroupeur. C'est par ce produit que nous pourrons faire opérer un regroupeur sur un automate cellulaire donné afin d'obtenir une simulation. Le lemme suivant est un préliminaire à la définition :

Lemme 3.4

Soit A un automate cellulaire et M un regroupeur. Si (P_1, P_2, P_3) est un triplet de Φ, si $f_1 : P_1 \to \mathcal{Q}_A$, $f_2 : P_2 \to \mathcal{Q}_A$ et $f_3 : P_3 \to \mathcal{Q}_A$ sont trois fonctions étiquetant les sites des pièces P_1, P_2 et P_3 par des états de $\mathcal{Q}_A$, alors il existe au plus une fonction f ayant les propriétés suivantes :

- $f : (P_1 \cup (P_2 + \omega_1(P_1, P_2, P_3)) \cup (P_3 + \omega_2(P_1, P_2, P_3)) \cup (\delta(P_1, P_2, P_3) + \omega_1(P_1, P_2, P_3) + \theta(P_1, P_2, P_3))) \to \mathcal{Q}_A$;
- *la restriction de f à la pièce P_1 coïncide avec f_1 et de même pour f_2 décalée de $\omega_1(P_1, P_2, P_3)$ et f_3 décalée de $\omega_2(P_1, P_2, P_3)$;*
- *la fonction f est compatible avec δ_A, c'est-à-dire :*
  $f(i,t) = \delta_A(f(i-1, t-1), f(i, t-1), f(i+1, t-1))$ *si les 4 sites (i,t), $(i-1, t-1)$, $(i, t-1)$ et $(i+1, t)$ sont dans le domaine de f.*

C'est une conséquence immédiate de la condition de faisabilité.

Définition 3.6 (produit par un regroupeur)

Soient un automate cellulaire $A = (\mathcal{Q}_A, \delta_A)$ et un regroupeur $M = ((\mathcal{Q}_M, \delta_M), F, \theta)$; on définit l'automate cellulaire produit $A \otimes M = (\mathcal{Q}_{A\otimes M}, \delta_{A\otimes M})$ de la façon suivante :

- *les états de l'automate cellulaire produit sont les fonctions des pièces de M vers les états de A, i.e. $\mathcal{Q}_{A\otimes M} = \{g : P \to \mathcal{Q}_A \text{ t.q. } P \in \mathcal{Q}_M\}$.*
- *la fonction de transition est la fonction partielle $\delta_{A\otimes M} : (f_1, f_2, f_3) \mapsto f$ donnée par le lemme 3.4.*

Fait 3.5

Soit u la configuration de départ de l'automate cellulaire A.

Si la fonction $f : \langle i, t\rangle_M \to \mathcal{Q}_A$ est l'état du site (i,t) dans le diagramme espace-temps de la machine $A \otimes M$ lancée sur l'entrée $\boxed{\text{G}}.\square^{|u|}.\boxed{\text{D}}$ (étiquetée par les lettres de u), alors $f(a,b)$ est l'état du site $(a,b) + v_{i,t}$ dans le diagramme espace-temps du calcul de la machine A lancée sur l'entrée $\$.\sharp^{|u|}.\$$.

Ce fait provient de l'existence d'un plongement et de l'unicité, quand il existe, de l'état d'un site de la machine produit (conséquence directe de la condition de faisabilité).

Remarques

- à propos des états de la machine produit $A \otimes M$, on ne retiendra en pratique que les fonctions qui sont compatibles avec la fonction de transition δ_A de l'automate à simuler (les autres sont incohérentes) ;
- c'est ce produit qui garantit qu'un regroupeur peut simuler un automate quelconque. La démarche pour obtenir une simulation est alors la suivante :
  1. imaginer une transformation,

2. exhiber un automate cellulaire qui la réalise,
3. vérifier que c'est un regroupeur (étape automatisable),
4. régler les conditions de terminaison en fonction du type de simulation désiré (faible ou forte, pour reconnaisseurs ou calculateurs).

2 Contre-exemple et exemple de regroupement

Contre-exemples

La plupart des automates cellulaires usuels ne sont pas des regroupeurs, soit parce que leurs états ne peuvent pas être interprétés comme des pièces (typiquement : un état pourrait avoir deux interprétations possibles), soit parce que la faisabilité ou la cohérence ne sont pas satisfaites.

Certains automates cellulaires « ressemblent » à des regroupeurs mais ne peuvent pas satisfaire toutes les conditions. C'est le cas pour la méthode de simulation des reconnaisseurs bi-infinis par des reconnaisseurs semi-infinis exposée au chapitre précédent (lemme 2.2). En effet, certaines cellules simulantes sont amenées à stocker des configuration du type : $(\langle t-1, t\rangle, \langle -t, t\rangle)$ pour des valeurs arbitrairement nombreuses de t (*cf.* section 4.1 du chapitre 2). L'ensemble des pièces est donc arbitrairement grand et une fonction de regroupement opérant sur cet ensemble ne pourrait pas être réalisée par un automate cellulaire.

Exemple

Nous proposons maintenant un exemple de regroupement très simple et typique puisque c'est lui qui donne le nom à la méthode. Si n est la longueur de la ligne, il transforme en n étapes une configuration non groupée en une configuration 2-groupée (au sens de la définition 7.1), tout en simulant le calcul initial à vitesse normale.

Avant de donner une description précise du regroupeur nous donnons deux schémas essentiels à la compréhension du mécanisme mis en œuvre (ici il est simple, pour d'autres regroupements il peut être plus complexe et de tels schémas deviennent indispensables).

La figure 3.6 montre le diagramme espace-temps du regroupeur vu comme un simple automate cellulaire (rappelons que les états de cette machine sont des pièces non étiquetés). Pour cette figure également, la configuration de départ vaut $\boxed{\text{G}}.\square^5.\boxed{\text{D}}$.

La figure 3.7 montre le plongement du regroupeur dans le demi-plan discret lorsqu'il est lancé sur la configuration initiale $\boxed{\text{G}}.\square^5.\boxed{\text{D}}$ placée par le vecteur $v_{0,0} = (0,0)$.

Nous passons maintenant à la description du regroupeur. Afin d'alléger l'écriture de la fonction de transition, on pose : $\vec{e} = (1,0)$ et $\vec{s} = (0,1)$ (E comme Est, S comme Sud).

Notons également que la valeur la plus fréquente pour θ est $\vec{s}$. *Nous n'indiquerons la valeur de θ que lorsqu'elle diffère de cette valeur par défaut.*

Regroupeur

$M = ((\mathcal{Q}, \delta), F, \theta)$.

Ensemble des états

$\mathcal{Q} = \{\boxed{\text{G}}, \square, \boxed{}, \boxed{\text{D}}\}$.

Famille de 3-agencements

$F = \{(\boxed{\text{G}}, \boxed{\text{G}}, \boxed{\text{G}}, \vec{e}, \vec{e}), (\boxed{\text{G}}, \boxed{\text{G}}, \square, \vec{e}, \vec{e}), (\boxed{\text{G}}, \boxed{\text{G}}, \boxed{}, \vec{e}, \vec{e}),$

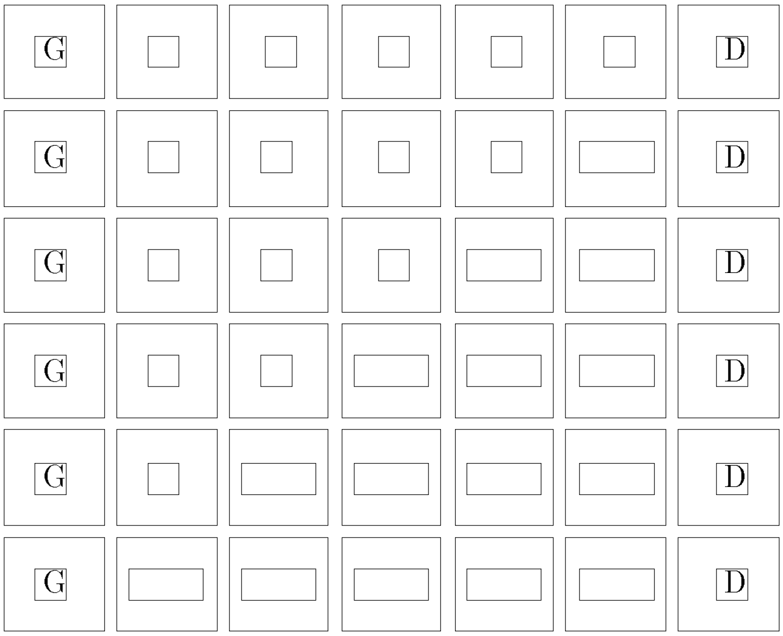


FIG. 3.6 – *Exemple de regroupement : diagramme espace-temps du regroupeur.*

$(\boxed{\mathrm{G}}, \square, \square, \vec{e}, \vec{e}), (\square, \square, \square, \vec{e}, \vec{e}), (\square, \square, \boxed{\mathrm{D}}, \vec{e}, \vec{e}),$
$(\square, \boxed{\mathrm{D}}, \boxed{\mathrm{D}}, \vec{e}, \vec{e}), (\boxed{\mathrm{D}}, \boxed{\mathrm{D}}, \boxed{\mathrm{D}}, \vec{e}, \vec{e}), (\boxed{\mathrm{G}}, \square, \boxed{}, \vec{e}, \vec{e}),$
$(\boxed{\mathrm{G}}, \boxed{}, \boxed{}, \vec{e}, 2.\vec{e}), (\square, \square, \boxed{}, \vec{e}, \vec{e}), (\square, \boxed{}, \boxed{}, \vec{e}, 2.\vec{e}),$
$(\boxed{}, \boxed{}, \boxed{}, 2.\vec{e}, 2.\vec{e}), (\square, \boxed{}, \boxed{\mathrm{D}}, 2.\vec{e}, \vec{e}), (\boxed{}, \boxed{}, \boxed{\mathrm{D}}, 2.\vec{e}, 2.\vec{e})\}$
(*N.B.* : nous avons cité ici tous les 3-agencements utiles, y compris ceux imposés par la condition de départ que nous ne rappelrons plus désormais.

Règles de base

`Normal` : $\delta(\square, \square, \square) = \square$,
`FrontièreG` : $\delta(\boxed{\mathrm{G}}, \boxed{\mathrm{G}}, x) = \boxed{\mathrm{G}}, \forall x \in \mathcal{Q}$,
`FrontièreD` : $\delta(x, \boxed{\mathrm{D}}, \boxed{\mathrm{D}}) = \boxed{\mathrm{D}}, \forall x \in \mathcal{Q}$ avec $\theta = \vec{s} + \vec{e}$,

Première étape

`Gauche1` : $\delta(\boxed{\mathrm{G}}, \square, \square) = \square$,
`Droite1` : $\delta(\square, \square, \boxed{\mathrm{D}}) = \boxed{}$.

Ligne de longueur 1

`Dégénéré1` : $\delta(\boxed{\mathrm{G}}, \square, \boxed{\mathrm{D}}) = \boxed{}$,
`Dégénéré2` : $\delta(\boxed{\mathrm{G}}, \boxed{}, \boxed{\mathrm{D}}) = \boxed{}$.

Cas général

`Droite2` : $\delta(\square, \boxed{}, \boxed{\mathrm{D}}) = \boxed{}$ avec $\theta = \vec{s} + \vec{e}$,
`Droite3` : $\delta(\boxed{}, \boxed{}, \boxed{\mathrm{D}}) = \boxed{}$ avec $\theta = \vec{s} + \vec{e}$,
`Couple1` : $\delta(\square, \square, \boxed{}) = \boxed{}$
`Couple2` : $\delta(\square, \boxed{}, \boxed{}) = \boxed{}$ avec $\theta = \vec{s} + \vec{e}$,

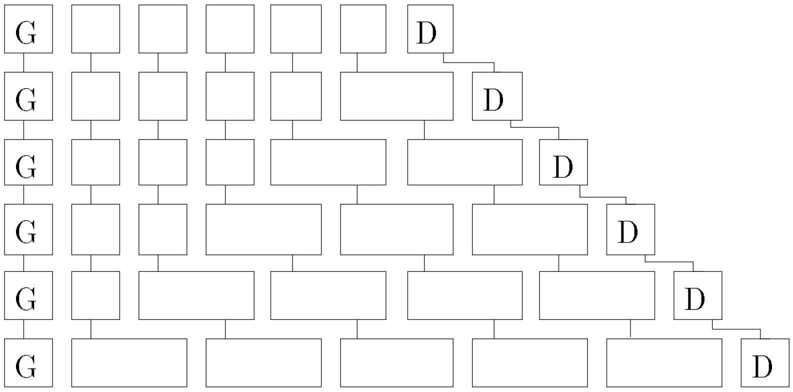


FIG. 3.7 – *Exemple de regroupement : plongement dans le demi-plan.*

`Couple3` : $\delta(\boxed{}, \boxed{}, \boxed{}) = \boxed{}$ avec $\theta = \vec{s} + \vec{e}$.

Règle au bord gauche

`Gauche2` : $\delta(\boxed{\text{G}}, \boxed{}, \boxed{}) = \boxed{}$.

Remarques

– ici nous avons précisé le cas d'une ligne réduite à une seule cellule non quiescente. Ce cas dégénéré se réduit en fait à un problème sur un automate fini très simple, ne contenant que des transitions non étiquetées. Dans toutes les simulation présentées au cours de la thèse ce cas est trivial ; Désormais nous l'omettrons.

– ce regroupeur n'est pas conçu pour fonctionner au delà du temps n, il doit être stoppé uniformément dès la n ème étape par une synchronisation en temps minimal à partir des deux extrémités de la ligne, après laquelle l'automate peut utiliser d'autres règles de transition (typiquement des règles de calcul rapide à partir de la configuration groupée) ;

– les règles `FrontièreG` et `FrontièreD` sont en fait des familles de règles.

Pour prouver que cette machine est bien un regroupeur, il faut :

1. prouver la propriété de faisabilité ;

2. prouver la propriété de cohérence.

Faisabilité

Il suffit de montrer que le membre droit de chaque règle, correctement placé par la fonction θ est bien un sous ensemble des conséquences de la réunion des trois pièces du membre gauche associé (chaque pièce étant correctement placée puisque la famille F des 3-agencements est rigide). Cette tâche est facilement automatisable (*cf.* Annexe A) mais nous montrons une vérification « à la main » pour quelques règles (figure 3.8).

Les règles `Couple1,2,3` concernent des 3-agencements de pièces finies. On calcule donc directement l'ensemble des conséquences (en pointillés sur la figure) et on vérifie que la nouvelle pièce est bien à l'intérieur.

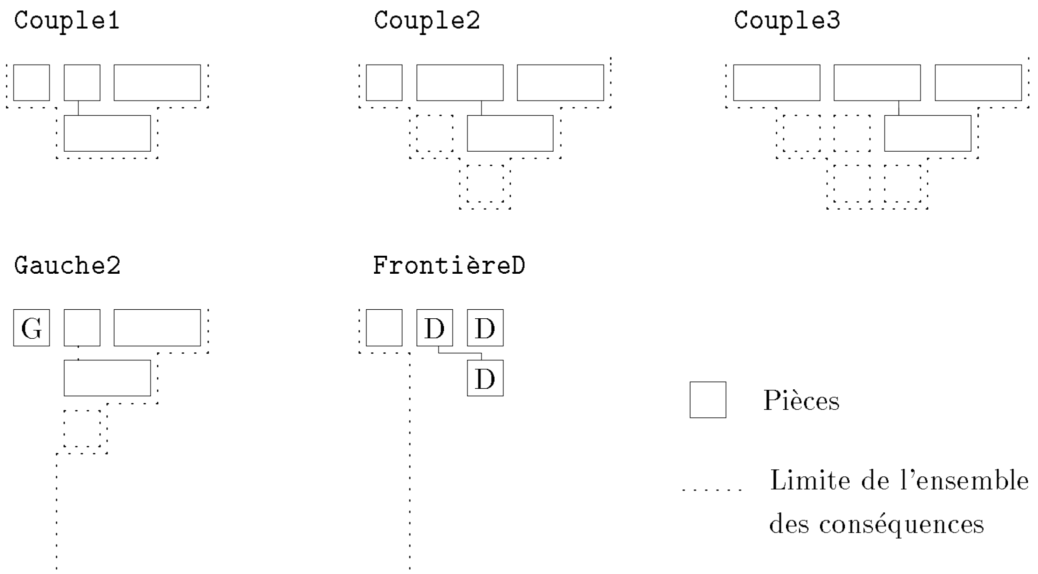


FIG. 3.8 – *Vérification de la faisabilité pour quelques règles de simulation.*

Les règles `Gauche1` et `FrontièreD` contiennent des pièces infinies, l'ensemble des conséquences est donc infini (il contient des bords) et l'on vérifie également que la pièce calculée est bien contenue dans les conséquences.

Cohérence

Il s'agit de vérifier que tous les 5-agencements F-corrects (au sens de la définition 3.3) sont transformés par les fonctions δ et θ en des agencement de F. Là encore cette tâche est automatisable mais nous effectuons le travail « à la main » sur trois exemples significatifs (figure 3.9).

Dans chacun de ces cas, on vérifie que le résultat de l'application du regroupeur à un 5-agencement est un 3-agencement de F et que le placement par les vecteurs ω_i est compatible avec le placement calculé par la fonction θ.

Remarque

La preuve qu'un automate est un regroupeur s'apparente donc plus à une vérification qui, nous l'avons dit, peut être automatisée (*cf.* Annexe A).

Il ne reste qu'à exhiber la propriété qui fait l'intérêt de notre regroupeur :

Proposition 3.6

Pour t variant entre 0 et $n-1$, la configuration au temps t plongée dans le demi-plan discret $\mathbb{Z}\times\mathbb{N}$ est de la forme $c_t = \boxed{G}.\boxed{\ }^{n-t}.\boxed{\quad\ }^{t-1}.\boxed{D}$ avec comme vecteur de positionnement $v_{0,t} = t.\vec{s}$.

En particulier au temps $n-1$ la configuration est entièrement groupée. Bien que la preuve soit extrêmement simple, nous en donnons une version détaillée à titre d'exemple d'utilisation des techniques du regroupement. Les preuves des lemmes 4.4, et 4.6 sont plus complexes mais utilisent les mêmes techniques.

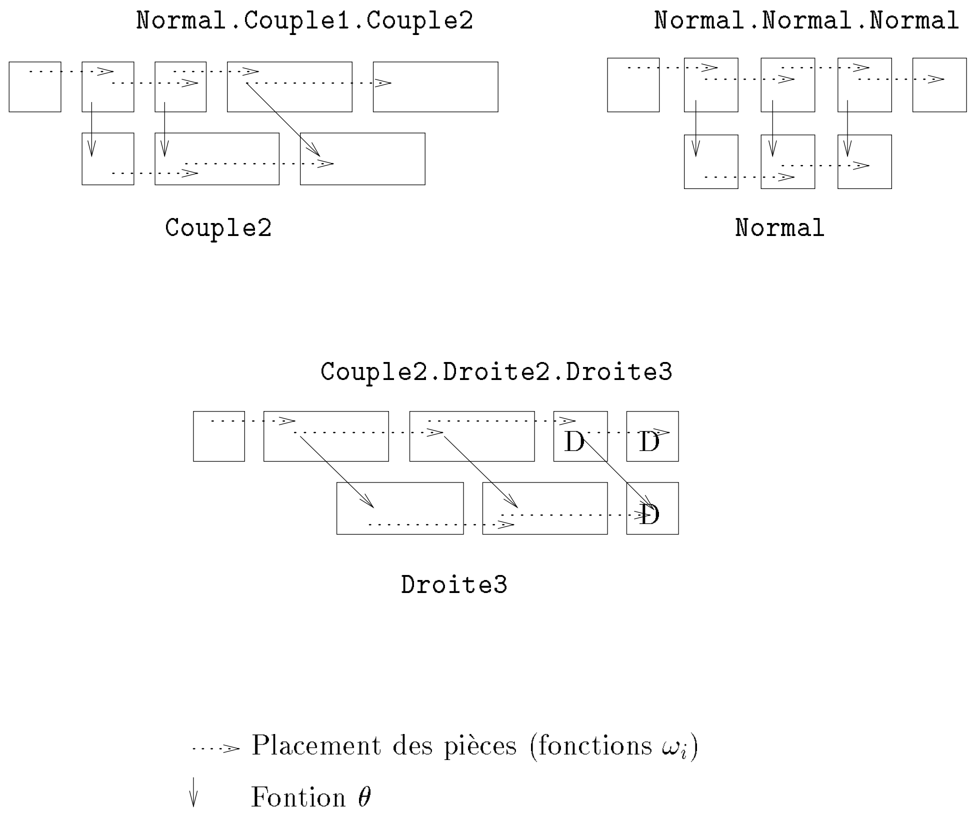


FIG. 3.9 – *Vérification de la cohérence pour quelques règles de simulation.*

Preuve

La configuration de départ, plongée dans le demi-plan discret est décrite par la liste des pièces $\boxed{\text{G}}.\square^n.\boxed{\text{D}}$ et par le vecteur de positionnement $v_{0,0} = (0,0)$ de la cellule d'indice 0. La preuve se fait par récurrence sur le temps t.

Cas de base

Au temps $t = 0$, on constate que les seules règles applicables sont, de gauche à droite, `FrontièreG`, `Gauche1`, `Normal`, `Droite1`, `FrontièreD`. On a :

$$\Delta(\boxed{\text{G}}.\square^n.\boxed{\text{D}}) = \boxed{\text{G}}.\square^{n-1}.\boxed{}.\boxed{\text{D}}$$

ce qui est bien un cas particulier de la formule générale donnée dans la proposition. Par ailleurs, on vérifie facilement que le vecteur de positionnement $v_{0,1} = v_{0,0} + \theta(\boxed{\text{G}}, \square, \square)$ vaut $\vec{s}$.

Au temps $t = 1$, les mêmes règles s'appliquent, à l'exception de `Droite1` qui est remplacée par `Droite2`. On a :

$$\Delta^2(\boxed{\text{G}}.\square^n.\boxed{\text{D}}) = \boxed{\text{G}}.\square^{n-2}.\boxed{}.\boxed{}.\boxed{\text{D}}$$

qui est encore un cas particulier de la formule donnée dans la proposition. Le vecteur de positionnement $v_{0,2} = v_{0,1} + \theta(\boxed{\text{G}}, \square, \square)$ vaut bien $2.\vec{s}$.

Pas de récurrence

Pour $2 < t \leq n-2$, observons la configuration $c_t = \boxed{\scriptstyle G}.\square^{n-t}.\boxed{}^{t-1}.\boxed{\scriptstyle D}$ placée par le vecteur $v_{0,t} = t.\vec{s}$ et recensons les règles qui peuvent s'appliquer ; il y a : `FrontièreG`, `Gauche1`, `Normal`, `Couple1,2,3`, `Droite3` et `FrontièreD`. Il est donc simple de calculer l'image de c_t par la fonction de transition globale Δ, on a :

$$\Delta(c_t) = \boxed{\scriptstyle G}.\square^{n-t-1}.\boxed{}^{t}.\boxed{\scriptstyle D}$$

qui vaut bien c_{t+1}. Par ailleurs, le vecteur de placement $v_{0,t+1} = v_{0,t} + \theta(\boxed{\scriptstyle G}, \square, \square)$ vaut bien $(t+1).\vec{s}$.

Cas particuliers

Dans le cas $t = n-1$ la règle spéciale `Gauche2` doit être appliquée. Nous écartons le cas dégénéré $n = 1$ pour lequel le regroupement n'a pas de sens.

Récapitulatif

Nous avons caractérisé la forme générale de la configuration pour tout $n > 0$ et pour tout $t > 0$, sans rencontrer de 3-agencement inconnnu. Ceci prouve la cohérence faible, qui nous suffit dans le cadre d'une simulation. De plus, au temps n la c_n est bien une configuration k-regroupée de la configuration initiale.

$\square$

Conclusions, perspectives

Dans la suite de cette thèse, nous appliquons avec succès la méthode de regroupement dans les deux cas suivants :

- accélération d'une constante des reconnaisseurs (finis ou semi-infinis) ;
- accélération linéaire des reconnaisseurs (finis ou semi-infinis).

Dans les deux cas, l'utilisation du regroupement est plus claire, diminue le nombre d'états nécessaires pour réaliser la simulation et permet de supprimer une synchronisation auxiliaire. En particulier, dans le deuxième cas, nous pensons que le nombre d'états de la machine simulante ($2.s^k$ pour une machine initiale à s états) est minimal.

Dans d'autres cas, le regroupement permet de retrouver des résultats classiques simples (voir l'exemple « canonique » en section 2 de ce chapitre) et dans certains cas il échoue. En particulier le regroupement ne permet pas d'exprimer simplement le produit avec interaction de deux automates (typiquement, le produit d'un automate cellulaire quelconque et d'un synchroniseur, ou une machine détectant la parité de la longueur du mot d'entrée). C'est le cas pour les simulations sur les calculateurs (*cf.* chapitre 5). Les simulations exposées sont néanmoins très proches d'un regroupement « pur » et nous pensons que le formalisme peut être étendu (travail en cours). Nous donnons une interprétation partielle en terme de regroupements lorsque c'est possible.

Enfin, nous avons fait la remarque dans les section précédentes qu'il existe toujours un regroupeur trivial simulant exactement une machine donnée. Ce regroupeur joue le rôle d'identité pour le produit $\otimes$. Propriété qui peut s'avérer intéressante dans le cadre d'une étude des propriétés algébriques des opérateurs sur automates cellulaires (*cf.* discussion au chapitre 9).

Chapitre 4

Accélération d'une constante sur reconnaisseurs

Résumé

Nous introduisons de nouvelles méthodes d'accélération d'une constante pour les reconnaisseurs. Le but est de réduire la quantité de ressources supplémentaires et la fréquence de leur utilisation dans le diagramme espace-temps. Ce chapitre peut être vu comme une mise à l'épreuve de la méthode de regroupement sur un cas simple. Par la suite, elle est appliquée aux calculateurs (chapitre 5), pour lesquels le problème d'accélération est bien plus difficile.

Introduction

Nous rappelons tout d'abord les préoccupations qui ont conduit à la mise au point du résultat. Dès que l'on utilise les automates cellulaires comme moyen de calcul, il est naturel de vérifier que des propriétés de base en complexité sont valides pour ce modèle. Les résultats les plus élémentaires - l'accélération d'une constante et l'accélération linéaire - ont été largement étudiés [CČ83, Col69, IKM85, MR92].

Pourtant, d'un point de vue structurel, il subsiste un paradoxe : alors qu'il semble plus facile d'accélérer d'une constante que d'un facteur, les méthodes développées jusqu'alors nécessitent *plus* de nouveaux états pour le faire.

Plus précisément, la quantité de ressources (états) nécessaires pour gagner k pas de calcul sur une machine à s états est exponentielle, alors que calculer k fois plus vite ne nécessite qu'un accroissement polynomial du nombre d'états. Notons que la version forte du théorème d'accélération linéaire [MR92] qui peut être utilisée pour diminuer le nombre d'états est postérieure à la preuve initiale du théorème d'accélération d'une constante [CČ83].

Ceci conduit à quelques questions auxquelles nous tentons de répondre dans ce chapitre :

1. Est-il possible de réaliser l'accélération d'une constante avec aussi peu d'états que pour l'accélération linéaire ?

2. Une synchronisation est-elle toujours nécessaire pour l'accélération d'une constante ?

3. Est-il possible de réaliser l'accélération d'une constante avec *moins* d'états que pour l'accélération linéaire, comme le suggère l'intuition ?

4. À quelle fréquence les nouveaux états sont-ils utilisés dans le diagramme espace-temps de la machine accélérée ?

En section 1 nous posons formellement le problème et évoquons les différentes méthodes de preuve envisagées.

En section 2 nous adaptons l'accélération linéaire de Mazoyer et Reimen [MR92] pour obtenir une réponse positive à la première question.

En section 3 nous proposons une nouvelle preuve qui ne nécessite pas de synchronisation, répondant ainsi par la négative à la deuxième question. De plus, la densité d'utilisation des nouveaux états est fortement diminuée.

En section 4 nous obtenons une accélération de k pas pour des machines à s états « suffisamment lentes ». La méthode utilisée conduit à un automate accéléré à $s^2+\mathcal{O}(k)$ états seulement. Nous évoquons également quelques aspects spécifiques de l'accélération d'une constante (tels que la densité) ainsi qu'une généralisation des précédents résultats aux machines infinies. Enfin nous donnons quelques indications à propos de la troisième question qui reste un problème ouvert pour le théorème en version forte.

1 Problématique

Nous discutons maintenant de différents moyens de prouver le théorème 4.1, certains issus (ou adaptés) de la littérature, d'autres nouveaux selon l'auteur.

Théorème 4.1 (accélération d'une constante pour reconnaisseur)

Soient k un entier strictement positif et u un mot d'entrée.

Première forme : *tout reconnaisseur (fini, semi-infini ou bi-infini) travaillant en temps $|u|+k+t(u)$ peut être simulé par un reconnaisseur de même nature, travaillant en temps $|u|+t(u)$.*

Deuxième forme : *tout reconnaisseur (fini, semi-infini ou bi-infini) travaillant en temps $|u|+t(u)$ peut être simulé par un reconnaisseur de même nature, travaillant en temps $|u|+\sup\{0,t(u)-k\}$.*

Remarques

– toute preuve de la première forme peut être transformée en une preuve de la deuxième au prix de modifications minimes (voir page 112 le lemme 7.11 de modification finie pour les synchroniseurs pour une idée des techniques utilisables) ;

– tel qu'il est énoncé ci-dessus, le théorème 4.1 n'indique rien sur l'accroissement du nombre d'états. En effet, ce dernier dépend de la technique de preuve utilisée et nous cherchons à le minimiser. Cet aspect est pris en compte dans les propositions 4.3, 4.5 et 4.7.

La figure 4.1 synthétise les différentes techniques de preuve envisagées ainsi que leur spécificités. Les parties en gris foncé sont accélérées (où certaines cellules ont déjà pu calculer leur état futur). Les parties en gris clair sont les zones denses dans les diagrammes (c'est-à-dire utilisant beaucoup de produits d'états, *cf.* définition 2.9 chapitre 2).

Considérons la ligne *densité* : l'ensemble d'états de la machine simulante contient les anciens et les nouveaux (souvent des produits), ces derniers sont utilisés sur de plus ou moins grandes zones des diagrammes espace-temps. La taille de ces zones est quadratique en n

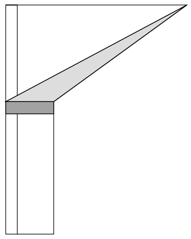
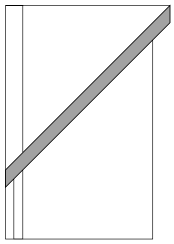
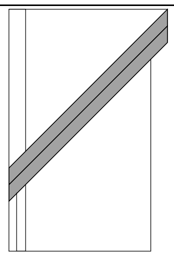
Schéma			
Méthode	Regroupement	Diagonale simple	Diagonale double
Nombre d'états	s^{k+1}	s^{2k+1}	s^{k+1}
Densité	Type surface	Linéaire	Linéaire

FIG. 4.1 – *Accélération des reconnaisseurs : techniques de preuve.*

(de type surface) pour la solution avec regroupement alors qu'elle est linéaire en n pour les solutions basées sur des diagonales.

Avant de nous consacrer au détail des preuves nous introduisons une notation, redondante avec la terminologie des regroupements, mais qui permet d'exprimer plus clairement certaines règles de transition.

Notation

Certaines simulations utilisent deux types particuliers de couples d'états : les couples « horizontaux » usuels notés (x, y) où $x = \langle i, t\rangle$ et $y = \langle i+1, t\rangle$, et les couples « verticaux » notés $\overline{(x, y)}$ où $x = \langle i, t\rangle$ et $y = \langle i, t+1\rangle$.

2 Accélérer d'un facteur pour gagner une constante

Cette première méthode repose sur le fait que l'accélération d'une constante peut être vue comme un cas particulier de l'accélération linéaire (théorème 4.2 énoncé ci-après).

Pour gagner k pas de calcul, il suffit en effet de préparer une accélération linéaire d'un facteur $k+1$ pendant les premiers pas du calcul, d'effectuer *un seul* pas de calcul à la vitesse $k+1$ puis de revenir à la vitesse normale (c'est-à-dire celle de la machine simulée : un pas de calcul par unité de temps).

Nous reprendrons ce raisonnement à la section 4.1 de ce chapitre à propos de la version faible du théorème, bien plus économe en ressources.

Tout d'abord nous rappelons la version forte du théorème d'accélération linéaire :

Théorème 4.2 (accélération linéaire pour reconnaisseurs) [IKM85, MR92]

Soient k un entier strictement positif et u un mot d'entrée.

tout reconnaisseur (semi-infini, fini) à s états travaillant en temps $|u| + t(u)$ peut être simulé par un reconnaisseur de même nature à $\mathcal{O}(s^k)$ états travaillant en temps $|u| + \lceil t(u)/k\rceil$.

Remarques

- une nouvelle preuve du théorème 4.2 est proposée au chapitre 6, mais la méthode qui convient ici est celle de Mazoyer et Reimen [MR92] ;

- nous ne connaissons pas de tel théorème pour les reconnaisseurs bi-infinis ni pour les calculateurs, même finis. En revanche il en existe une version faible pour les calculateurs travaillant en temps linéaire ;

- à cause de l'indispensable processus de synchronisation en temps minimal (à partir des deux extrémités de la ligne), la constante multiplicative dans l'expression $\mathcal{O}(s^k)$ est de l'ordre de 5 ;

- le cas fini est obtenu directement lorsque la ligne a la propriété de finitude.

Nous présentons ici une adaptation (directe) du théorème 4.2 au cas de l'accélération d'une constante. La figure 4.2 montre la répartition des sites du diagramme espace-temps de l'automate cellulaire simulé dans le diagramme espace-temps de l'automate cellulaire simulant.

Remarque pour la lecture du diagramme : la case la plus à droite ne contient pas un état mais une information concernant le déroulement de la synchronisation en temps minimal lancée à partir des deux extrémités de la ligne dès le premier pas de calcul. Cette information est accessible à toutes les cellules d'indice compris entre 0 et $|u|-1$ (dans le diagramme de la figure 4.2, on a $|u|=4$).

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\sharp$	$\sharp$	avant synchr.
$\sharp$	$\langle 0,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle\langle 4,1\rangle$	$\sharp$	$\sharp$	avant synchr.
$\sharp$	$\langle 0,2\rangle$	$\langle 1,2\rangle$	$\langle 2,2\rangle\langle 3,2\rangle$	$\langle 4,2\rangle\langle 5,2\rangle$	$\sharp$	$\sharp$	avant synchr.
$\sharp$	$\langle 0,3\rangle$	$\langle 1,3\rangle\langle 2,3\rangle$	$\langle 3,3\rangle\langle 4,3\rangle$	$\langle 5,3\rangle\langle 6,3\rangle$	$\sharp$	$\sharp$	avant synchr.
$\sharp$	$\langle 0,4\rangle\langle 1,4\rangle$	$\langle 2,4\rangle\langle 3,4\rangle$	$\langle 4,4\rangle\langle 5,4\rangle$	$\langle 6,4\rangle\langle 7,4\rangle$	$\sharp$	$\sharp$	synchronisation
$\sharp$	$\langle 0,6\rangle\langle 1,6\rangle$	$\langle 2,6\rangle\langle 3,6\rangle$	$\langle 4,6\rangle\langle 5,6\rangle$	$\langle 6,6\rangle\langle 7,6\rangle$	$\langle 8,6\rangle\ \sharp$	$\sharp$	après synchr.
$\sharp$	$\langle 0,7\rangle\langle 1,7\rangle$	$\langle 2,7\rangle\langle 3,7\rangle$	$\langle 4,7\rangle\langle 5,7\rangle$	$\langle 6,7\rangle\langle 7,7\rangle$	$\langle 8,7\rangle\ \sharp$	$\langle 9,7\rangle\ \sharp$	après synchr.

FIG. 4.2 – *Utilisation de l'accélération linéaire pour réaliser l'accélération d'une constante.*

Le diagramme présenté en figure 4.2 appelle quelques remarques.

Remarques

- nous voulons que la simulation fonctionne pour les reconnaisseurs semi-infinis (pour lesquels $\delta(a,b,\sharp)\neq\sharp$) ; dans le diagramme espace temps de l'automate initial, la longueur de la ligne augmente donc d'une unité à chaque pas de calcul ; dans la machine simulante, cette accroissement vers la droite n'intervient pas avant le temps $|u|+1$ car il est exactement compensé par la compression de la configuration due au regroupement des états ;

- la machine simulante est le produit d'un synchroniseur en temps minimal à partir des deux extrémités de la ligne, tel qu'il est décrit dans [MR92] et de l'automate obtenu par composition du regroupeur donné en exemple au chapitre précédent. Au temps $|u|$, le synchroniseur et le regroupeur *interagissent*. L'automate résultant n'est donc plus un « pur » regroupeur. Certes, il est possible d'étendre les définitions du chapitre 3 de façon à prendre en compte cette interaction mais nous ne le ferons pas ici : nous présentons un énoncé classique des règles de simulation et donnons seulement un aperçu de l'interprétation en terme de regroupement (voir figure 4.3 pour le fonctionnement du regroupeur et figure 4.4 à la fin de cette section pour une interprétation géométrique des règles de transition). En revanche, des exemples de purs regroupements apparaissent dans la suite du document où ils sont traités comme tels.

Avant de passer à l'énoncé « classique » des règles de simulation, rappelons que les lettres $a, b, \ldots$ représentent des états non quiescents tandis que les lettres $\ldots y, z$ représentent des états potentiellement quiescents. Par ailleurs, la fonction de transition δ' exposée ci-dessous est conditionnée par l'état de la solution synchronisante utilisée en parallèle : certaines règles valides avant la synchronisation ne le sont plus après, d'autres sont valides seulement pendant l'étape de synchronisation.

Ensemble des états

$\mathcal{Q}' = \mathcal{Q} \cup \mathcal{Q}^2$.

Premier pas

`Normal`: $\delta'(x, y, a) = \delta(x, y, a)$,

`Initialise1`: $\delta'(a, b, \sharp) = (\delta(a, b, \sharp), \delta(a, \sharp, \sharp))$.

Regroupement (avant synchronisation)

`Contracte1`: $\delta'(a, b, (c, d)) = (\delta(a, b, c), \delta(b, c, d))$,

`Contracte2`: $\delta'(a, (b, c), (d, e)) = (\delta(b, c, d), \delta(c, d, e))$,

`Contracte3`: $\delta'((a, b), (c, d), (e, f)) = (\delta(c, d, e), \delta(d, e, f))$,

`Contraction4`: $\delta'((a, b), (c, d), \sharp) = (\delta(c, d, \sharp), \delta(d, \sharp, \sharp))$,

`Droite1`: $\delta'((a, b), \sharp, \sharp) = \sharp$,

`Droite2`: $\delta'(a, (b, c), \sharp) = (\delta(b, c, \sharp), \delta(c, \sharp, \sharp))$,

`Gauche`: $\delta'(\sharp, a, (b, x)) = \begin{cases} \delta(\sharp, a, b) \text{ si } \delta(\sharp, a, b) \in \mathcal{S}_a \cup \mathcal{S}_r \\ \delta(\sharp, \delta(\sharp, a, b), \delta(a, b, x)) \text{ si } \delta(\sharp, \delta(\sharp, a, b), \delta(a, b, x)) \in \mathcal{S}_a \cup \mathcal{S}_r \\ (\delta(\sharp, a, b), \delta(a, b, x)) \text{ sinon} \end{cases}$

(cette règle fait l'objet d'une remarque plus bas dans le texte).

À la synchronisation

`Rapide1`: $\delta'((a, b), (c, d), (e, f)) =$

$$(\delta(\delta(a, b, c), \delta(b, c, d), \delta(c, d, e)), \delta(\delta(b, c, d), \delta(c, d, e), \delta(d, e, f)))\ ,$$

`Rapide2`: $\delta'((a, b), (c, d), \sharp) =$

$$(\delta(\delta(a, b, c), \delta(b, c, d), \delta(c, d, \sharp)), \delta(\delta(b, c, d), \delta(c, d, \sharp), \delta(d, \sharp, \sharp)))\ ,$$

`Rapide3`: $\delta'(\sharp, (a, b), (x, y)) =$

$$\begin{cases} \delta(\sharp, \delta(\sharp, a, b), \delta(a, b, x)) \text{ si } \delta(\sharp, \delta(\sharp, a, b), \delta(a, b, x)) \in \mathcal{S}_a \cup \mathcal{S}_r \\ (\delta(\sharp, \delta(\sharp, a, b), \delta(a, b, x)), \delta(\delta(\sharp, a, b), \delta(a, b, x), \delta(b, x, y))) \text{ sinon} \end{cases}$$

`Quies2a`: $\delta'((a, b), \sharp, \sharp) = (\delta(b, \sharp, \sharp), \sharp)$.

Après la synchronisation

`Lent1`: $\delta'((a, b), (c, d), (e, f)) = (\delta(b, c, d), \delta(c, d, e))$,

`Lent2`: $\delta'(\sharp, (a, b), (c, d)) = \begin{cases} \delta(\sharp, a, b) \text{ si } \delta(\sharp, a, b) \in \mathcal{S}_a \cup \mathcal{S}_r \\ (\delta(\sharp, a, b), \delta(a, b, c)) \text{ sinon} \end{cases}$

`Quiescent1`: $\delta'(\sharp, \sharp, (a, b)) = \sharp$,

`Droite3`: $\delta'((a, b), (c, \sharp), \sharp) = (\delta(b, c, \sharp), \sharp)$,

`Droite4`: $\delta'((a, \sharp), \sharp, \sharp) = (\delta(a, \sharp, \sharp), \sharp)$.

Remarque

Le fonctionnement de la règle `Gauche` nécessite quelques explications : il diffère selon que le calcul termine immédiatement, au prochain pas ou plus tard.

Dans le premier cas, le résultat est envoyé et le temps de simulation est $|u|$, le meilleur possible en fait car tout calcul non trivial sur u nécessite au moins $|u|$ étapes.

Dans le deuxième cas, la cellule 0 dispose de toute l'information pour calculer son état un cran dans le futur. L'état terminal est donc renvoyé au temps $|u|$ (au lieu de $|u| - 1$) et l'accélération d'un pas est réalisée.

Dans le derniers cas, la cellule 0 reste dans un état regroupé et calcule à vitesse normale (plus tard, cette information sera utile à la terminaison anticipée du calcul).

À propos de la preuve

La correction du processus avant la synchronisation est prouvée dans [MR92] et la preuve de correction pour la partie à vitesse normale (après la synchronisation) prend quelques lignes dans l'article de Cole [Col69]). Enfin, la partie de cette machine qui se rapproche le plus d'un regroupeur (*i.e.* les règles permettant la contraction) est étudiée en détail dans le chapitre 3 section 2.

Généralisation

La méthode précédente peut être étendue directement pour économiser k pas, conduisant au résultat suivant :

Proposition 4.3 (acc. d'une constante pour reconnaisseurs, 2 $^{\text{ème}}$ forme)

Soient k un entier strictement positif et u un mot d'entrée.

Tout reconnaisseur fini à s états travaillant en temps $|u| + k + t(u)$ peut être simulé par un reconnaisseur fini à $s^{k+1} + s$ états travaillant en temps $|u| + t(u)$.

À titre indicatif seulement, nous proposons une interprétation partielle en termes de regroupement, en insistant sur le fait que la méthode utilisée n'est *pas* un pur regroupement (à cause de la synchronisation), contrairement à la méthode de la diagonale exposée plus loin.

Sur la figure 4.3, on observe en haut le résultat du plongement dans le demi-plan discret. La partie inférieure représente le fonctionnement de la machine simulante quand les états sont interprétés par des pièces. L'effet du synchroniseur est clairement visible, lors du « saut » pendant la transition du temps 4 au temps 5. Tels que nous les avons défini au chapitre 3, les regroupeurs ne savent pas réaliser ce saut (voir chapitre 9 pour une discussion sur un moyen d'intégrer ces possibilités).

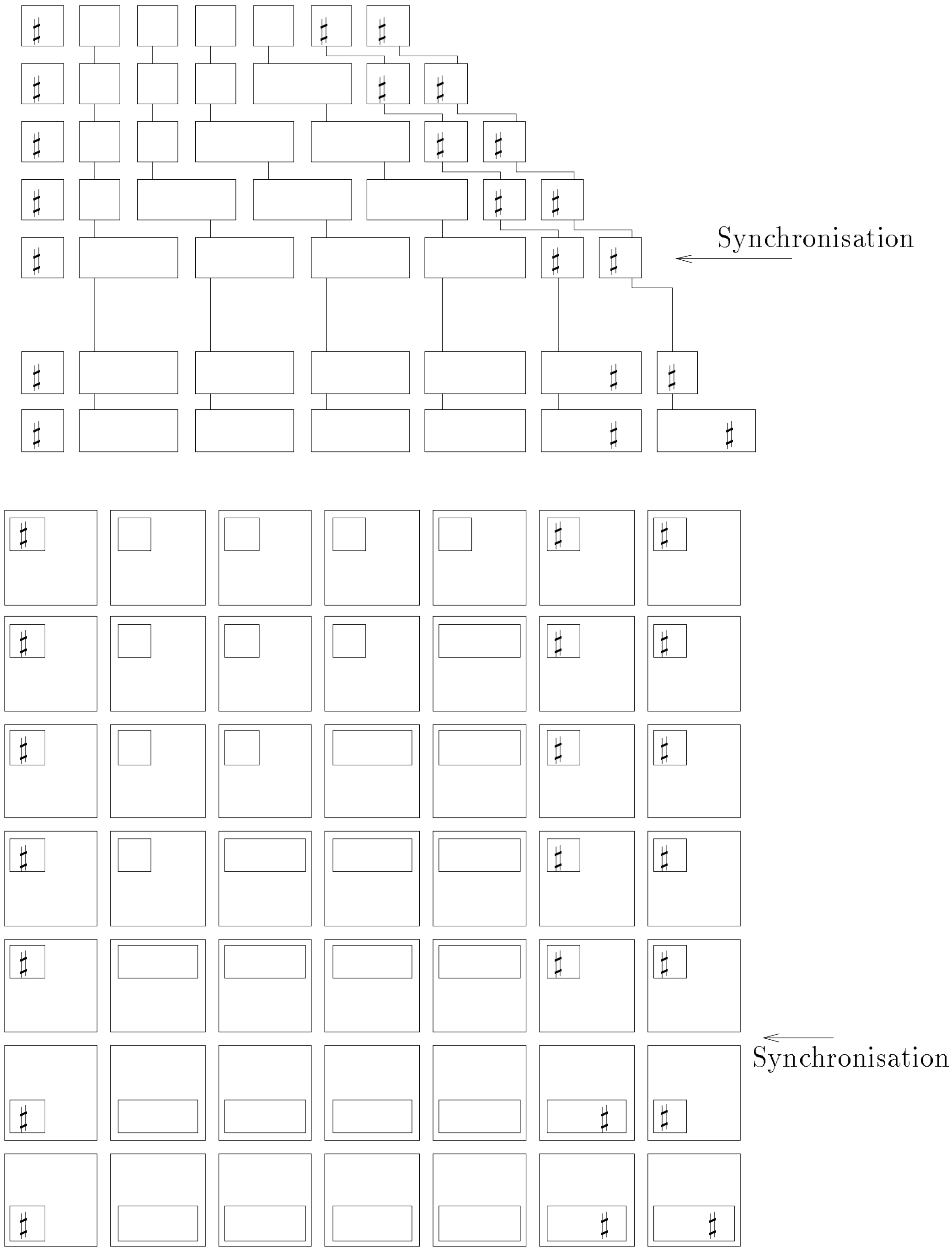


FIG. 4.3 – *Accélération avec synchronisation : plongement et fonctionnement.*

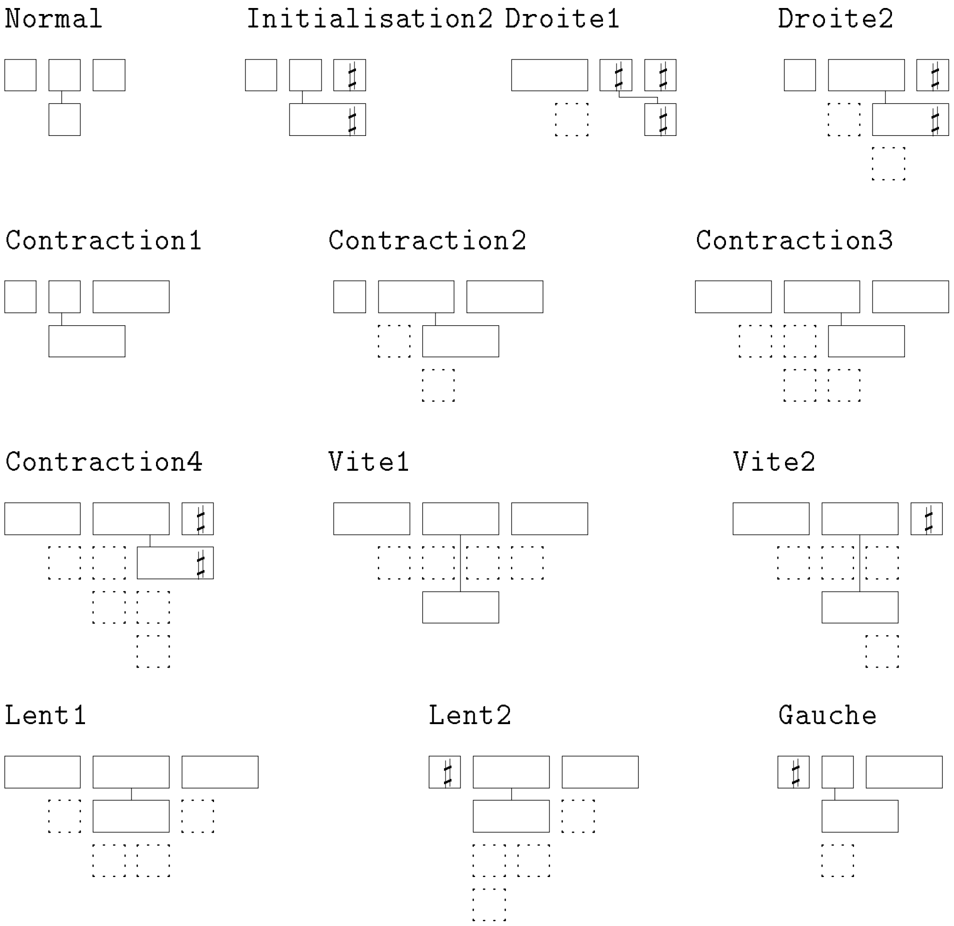


FIG. 4.4 – *Interprétation géométrique des principales règles de simulation de l'automate permettant l'accélération d'un pas des reconnaisseurs semi-infinis.*

3 Contraction/dilatation sans synchronisation

La technique précédente utilise une contraction complète de la ligne puisque k états sont gérés par chacune des cellules à l'issue de la synchronisation. Or on peut remarquer qu'une fois qu'un pas est économisé quelque part sur la ligne, la configuration regroupée n'est plus utile. En d'autres termes, un processus de dilatation pourrait avoir lieu juste après la contraction initiale.

Nous donnons tout d'abord une illustration simple et visuelle de cette idée (mais pas très efficace du point de vue des ressources), avec une preuve de correction complète, puis nous utilisons une technique d'optimisation, la méthode de la double diagonale, pour réduire la quantité de ressources supplémentaires. Nous étendons cette dernière méthode au cas général (c'est-à-dire à une accélération de k pas).

3.1 Gagner un pas grâce à la méthode de la diagonale simple

La figure 4.5 montre la répartition des sites du diagramme espace-temps de l'automate cellulaire simulé dans le diagramme espace-temps de l'automate cellulaire simulant et illustre la méthode de la diagonale simple pour $|u| = 6$ et t variant entre 0 et 8. On remarque qu'aucune synchronisation n'est requise.

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\langle 5,0\rangle$	$\sharp$
$\sharp$	$\langle 0,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\langle 5,1\rangle\sharp\sharp$	$\sharp$
$\sharp$	$\langle 0,2\rangle$	$\langle 1,2\rangle$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle\langle 5,2\rangle\langle 5,3\rangle$	$\$$	$\sharp$
$\sharp$	$\langle 0,3\rangle$	$\langle 1,3\rangle$	$\langle 2,3\rangle$	$\langle 3,3\rangle\langle 4,3\rangle\langle 4,4\rangle$	$\langle 5,4\rangle$	$\$$	$\sharp$
$\sharp$	$\langle 0,4\rangle$	$\langle 1,4\rangle$	$\langle 2,4\rangle\langle 3,4\rangle\langle 3,5\rangle$	$\langle 4,5\rangle$	$\langle 5,5\rangle$	$\$$	$\sharp$
$\sharp$	$\langle 0,5\rangle$	$\langle 1,5\rangle\langle 2,5\rangle\langle 2,6\rangle$	$\langle 3,6\rangle$	$\langle 4,6\rangle$	$\langle 5,6\rangle$	$\$$	$\sharp$
$\sharp$	$\sharp\langle 0,7\rangle\langle 1,7\rangle$	$\langle 2,7\rangle$	$\langle 3,7\rangle$	$\langle 4,7\rangle$	$\langle 5,7\rangle$	$\$$	$\sharp$
$\sharp$	$\sharp\langle 0,8\rangle\langle 1,8\rangle$	$\langle 2,8\rangle$	$\langle 3,8\rangle$	$\langle 4,8\rangle$	$\langle 5,8\rangle$	$\$$	$\sharp$
$\sharp$	$\sharp\langle 0,9\rangle\langle 1,9\rangle$	$\langle 2,9\rangle$	$\langle 3,9\rangle$	$\langle 4,9\rangle$	$\langle 5,9\rangle$	$\$$	$\sharp$

FIG. 4.5 – *Accélération d'un pas par la méthode de la diagonale simple.*

Contrairement à la précédente, cette méthode d'accélération est un pur regroupement : la machine simulante est obtenue par composition de l'automate cellulaire initial et du regroupeur $M = ((\mathcal{Q}_M, \delta_M), F, \theta)$ dont nous donnons les règles de transition ci-après. L'interprétation géométrique des principales d'entre elles apparaît en figure 4.6.

Ensemble des états

$\mathcal{Q}_M = \mathcal{Q}^3 \cup \mathcal{Q}^2 \cup \mathcal{Q} \cup \{\$\}$.

Famille de 3-agencements

$F = \{ \ (\square, \urcorner, \boxed{D}, \vec{e}, \vec{s}+2.\vec{e}), (\square, \square, \urcorner, \vec{e}, \vec{e}), (\square, \urcorner, \square, \vec{e}, \vec{s}+2.\vec{e}),$
$(\urcorner, \square, \square, \vec{s}+2.\vec{e}, \vec{e}), (\boxed{G}, \square, \urcorner, \vec{e}, \vec{e}), (\boxed{G}, \urcorner, \square, \vec{e}, \vec{s}+2.\vec{e}),$
$(\boxed{G}, \boxed{G\quad\quad}, \square, \vec{e}, 2.\vec{e}), (\boxed{G\quad\quad}, \square, x, 2.\vec{e}, \vec{e}) \ \}$.

N.B. : nous ne rappelons pas les 3-agencements imposés par la condition de départ de la définition 3.4 des regroupeurs.

Premiers pas

`Normal` : $\delta_M(x, \square, \square) = \square$,

`Initialise` : $\delta_M(\square, \square, \boxed{D}) = \urcorner$,

`Vide` : $\delta_M(\square, \urcorner, \boxed{D}) = \boxed{\$}$.

Diagonale

`Accélère1` : $\delta_M(\square, \square, \urcorner)) = \urcorner$,

`Accélère2` : $\delta_M(\square, \urcorner, \square) = \square$ avec $\theta = 2.\vec{s}+\vec{e}$,

`Accélère3` : $\delta_M(\urcorner, \square, \square) = \square$.

Côté droit

`Quiescent`: $\delta_M(\urcorner, \boxed{\text{D}}, \boxed{\text{D}}) = \boxed{\text{D}}$, $\delta_M(\square, \square, \boxed{\$}) = \square$, $\delta_M(x, \boxed{\$}, \boxed{\text{G}}) = \boxed{\$}$, $\delta_M(\square, \urcorner, \boxed{\$}) = \square$, $\delta_M(\urcorner, \square, \boxed{\$}) = \square$, $\delta_M(\boxed{\$}, \boxed{\text{D}}, \boxed{\text{D}}) = \boxed{\text{D}}$.

Côté gauche et dernier pas

`Gauche1`: $\delta_M(\boxed{\text{G}}, \square, \urcorner) = \begin{cases} \square \text{ avec } \theta = \vec{s} \text{ si } A \text{ termine à cette étape,} \\ \square \text{ avec } \theta = 2.\vec{s} \text{ si } A \text{ termine à la prochaine étape,} \\ \boxed{\text{G}\qquad} \text{ avec } \theta = 2.\vec{s} - \vec{e} \text{ sinon} \end{cases}$

`Gauche2`: $\delta_M(\boxed{\text{G}}, \boxed{\text{G}\qquad}, \square) = \begin{cases} \square \text{ si le calcul termine,} \\ \boxed{\text{G}\qquad} \text{ sinon} \end{cases}$

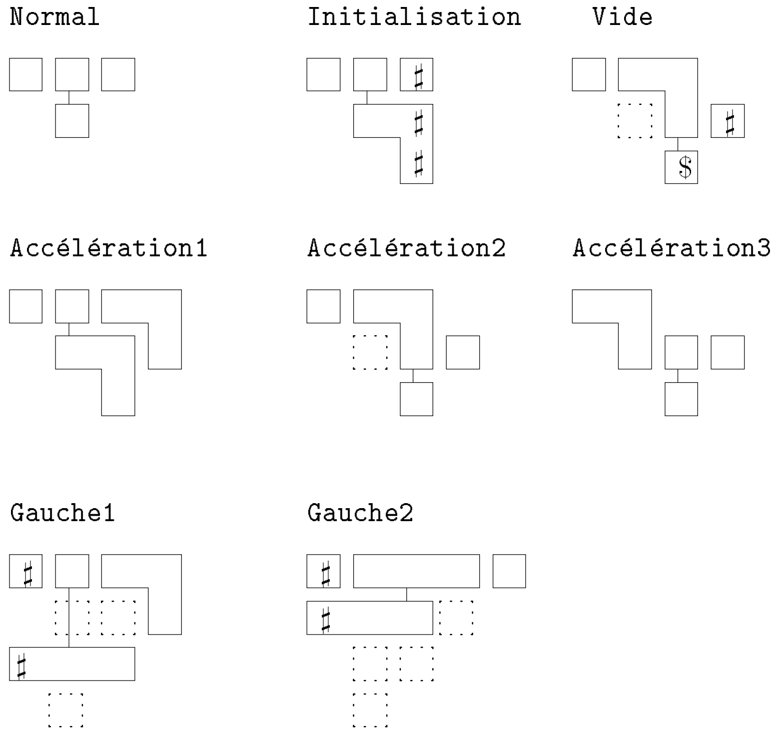


FIG. 4.6 – *Interprétation géométriques des principales règles pour l'accélération des reconnaisseurs par la méthode de la diagonale simple.*

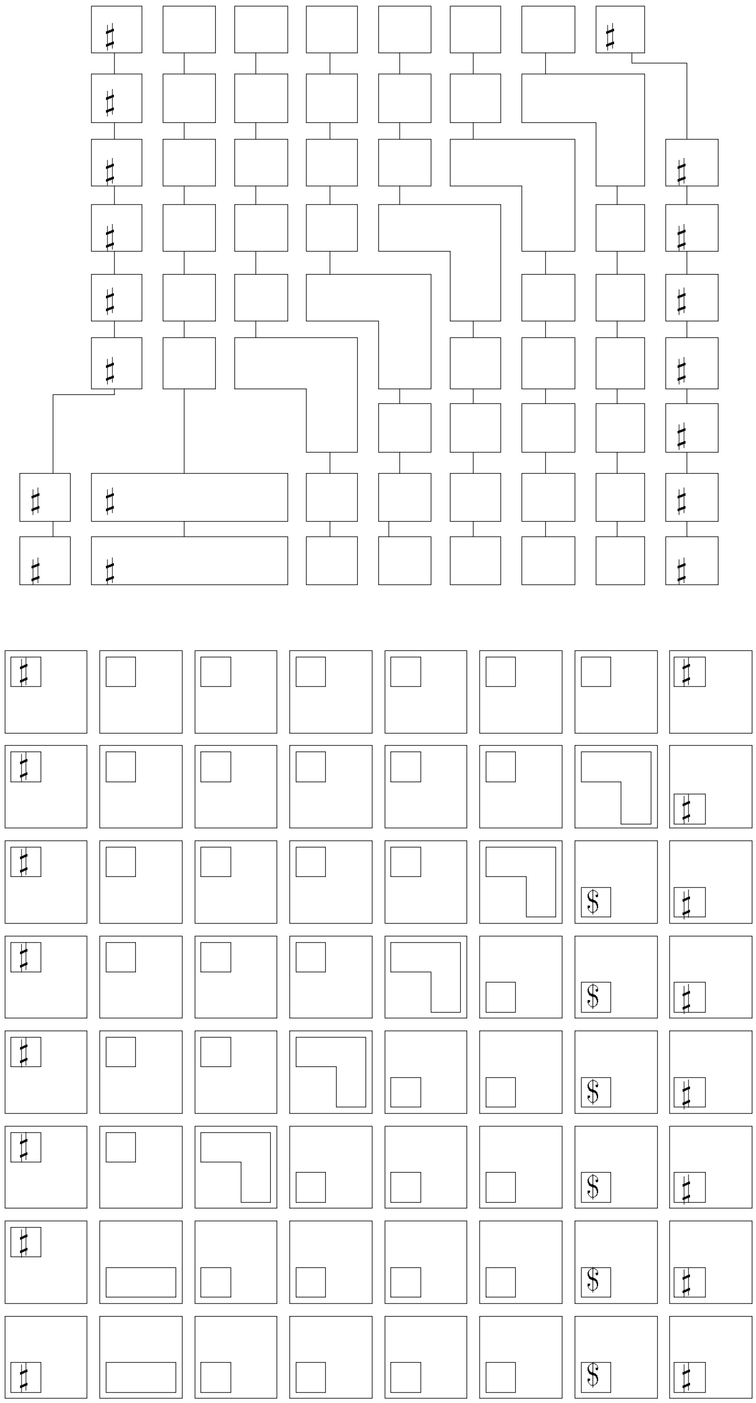


FIG. 4.7 – *Méthode de la diagonale simple : plongement et fonctionnement.*

Sur la figure 4.7 on observe en haut le plongement du regroupeur dans le demi-plan discret $\mathbb{Z} \times \mathbb{N}$. On y voit notamment la mise en place de la diagonale et le « saut » réalisé sur la cellule zéro entre le temps $|u|-1$ et le temps $|u|$ (ici $|u|=6$). La partie inférieure représente le diagramme espace-temps réel du regroupeur (non plongé dans le demi-plan discret).

Les phases successives de la simulation sont répertoriées dans le lemme 4.4 ci-après. Prouver ce lemme revient à prouver la cohérence faible (c'est-à-dire la pérénité des lignes de temps) en même temps que la propriété qui fait l'intérêt du regroupeur : à partir du temps $|u|$ les valeurs de la cellule zéro sont « en avance » d'une étape de calcul.

Lemme 4.4

Notons $c_0 = \boxed{\text{G}}.\square^n.\boxed{\text{D}}$ la configuration initiale du regroupeur. Au temps $t > 0$, l'image - plongée dans le demi-plan discret - de c_0 vaut :

$$Si\ t=1, c_1 = \boxed{\text{G}}.\square^{n-1}.\,\urcorner\!\!\lrcorner\,.\boxed{\text{D}} \quad avec\ v_{0,1} = \vec{s}$$

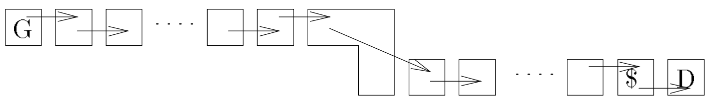
$$Si\ 2 \le t \le n-1, c_t = \boxed{\text{G}}.\square^{n-t}.\,\urcorner\!\!\lrcorner\,.\square^{t-1}.\boxed{\$}.\boxed{\text{D}} \quad avec\ v_{0,t} = t.\vec{s}$$

$$D'où\ v_{i,t} = \begin{cases} si\ 0 \le i \le n-t & (i,t) \\ si\ i \ge n-t+1 & (i+1,t+1) \end{cases}$$

$$Si\ t \ge n, c_t = \boxed{\text{G}}.\boxed{\qquad\quad}.\square^{n-1}.\boxed{\$}.\boxed{\text{D}} \quad avec\ v_{0,t} = (t+1).\vec{s}$$

Remarque

Compte tenu du lemme 3.1 des familles rigides, pour trois pièces données dans un certain ordre, il existe un seul 3-agencement possible. Par exemple, dans le cas $2 \le t \le n-1$ la configuration c_t s'agence obligatoirement de la façon suivante :



N.B. : les vecteurs apparaissant sur le diagramme sont les valeurs des fonction ω_1 et ω_2.

Preuve

Par récurrence sur t pour chaque cas :

Si $t=1$

Il suffit d'observer que les seules règles applicables à partir de la configuration de départ sont `Normal` et `Initialise`. L'image de la configuration initiale est alors :

$$\Delta(c_0) = \boxed{\text{G}}.\square^{n-1}.\,\urcorner\!\!\lrcorner\,.\boxed{\text{D}} = c_1 \text{ avec } v_{0,1} = (0,0) + \theta(\boxed{\text{G}}, \square, \square) = \vec{s}$$

Si $2 \le t \le n-1$

Cas de base

Du temps 1 au temps 2, peu de règles sont applicables ; il s'agit de `normal`, `Accélération1` et `Vide` qui crée l'état $\boxed{\$}$ en position n. On vérifie aisément que la configuration ainsi obtenue au temps 2 est bien un cas particulier de celle dessinée dans le lemme.

Étape de récurrence

En recensant les règles qui s'appliquent à la configuration au temps t on trouve, de gauche à droite `Normal`, `Accélération1`, `Accélération2`, `Accélération3` et le groupe de règles `Quiescent`. La configuration obtenue est alors la suivante :

$$\Delta(c_t) = \boxed{G}.\square^{n-t-1}.\square.\square^{t}.\boxed{\$}.\boxed{G} \text{ avec } v_{0,t+1} = v_{0,t} + \theta(\boxed{G}, \square, \square) = (0, t+1)$$

On a bien $\Delta(c_t) = c_{t+1}$.

Si $t \geq n$

Cas de base

C'est maintenant qu'à lieu l'accélération d'un pas pour la cellule 0 (rappelons que dans le cadre d'une simulation forte, elle ne peut pas avoir lieu plus tôt). Entre le temps $n-1$ et le temps n, les seules règles actives (applicables) sont `Gauche1`, `Accélération2`, `Accélération3`, `Normal` et le groupe de règles `Quiescent`. Si le calcul termine à cet instant, la règle `Gauche1` renvoie l'état terminal (et seulement celui-ci) dans la cellule 0. Sinon, la configuration calculée est :

$$\Delta(c_{n-1}) = \boxed{G}.\square.\square^{n-2}.\boxed{\$}.\boxed{D} \text{ avec } v_{0,n} = v_{0,n-1} + \theta(\boxed{G}, \square, \square) = (n+1).\vec{s} \neq n.\vec{s}$$

L'accélération a bien eu lieu puisque le vecteur de placement $v_{0,n}$ de la cellule 0 au temps n vaut $(n+1).\vec{s}$ et non pas $n.\vec{s}$.

Étape de récurrence

On vérifie que la configuration calculée précédemment a bien la forme annoncée dans le lemme. Un rapide examen des règles applicables (`Gauche2`, `Normal`, `Quiescent`) suffit pour constater que la configuration calculée à la prochaine étape est encore de la même forme.

Récapitulatif

Nous avons caractérisé la forme de la configuration pour tout n (assez grand, nous passons sous silence le cas dégénéré $n = 1$) et pour tout $t > 0$, sans rencontrer de 3-agencement inconnu. Ceci prouve la cohérence faible, qui nous suffit dans le cadre d'une simulation. De plus, nous avons prouvé la propriété d'accélération d'un pas à partir du temps n par l'examen attentif des valeurs des vecteurs de placement.

$\square$

Remarque

À titre indicatif seulement, nous donnons l'expression classique (c'est-à-dire n'utilisant pas la terinologie des regroupeurs) des règles de transition de la machine simulante. Nous pensons que ce formalisme est plus lourd et masque la nature *géométrique* du processus. La preuve du lemme de correction peut se faire dans ce formalisme mais les deux aspects importants, cohérence et accélération, sont plus difficiles à dissocier, aussi nous n'écrivons pas cette preuve ici[1].

1. Elle apparaît dans une version plus ancienne de ce travail [Hee95a], réalisée alors que la méthode de regroupement n'était pas formalisée.

Énoncé « classique » des règles de transition

Ensemble des états

$\mathcal{Q}' = \mathcal{Q}^3 \cup \mathcal{Q}^2 \cup \mathcal{Q} \cup \{\$\}$.

Premiers pas

`Normal`: $\delta'(x, y, z) = \delta(x, y, z)$, sauf
`Initialise`: $\delta'(a, b, \sharp) = (\delta(a, b, \sharp), \sharp, \sharp)$,
`Vide`: $\delta'(a, (b, \sharp, \sharp), \sharp) = \$$.

Diagonale

`Accélère1`: $\delta'(a, b, (c, x, y)) = (\delta(a, b, c), \delta(b, c, x), \delta(\delta(a, b, c), \delta(b, c, x), y))$,
`Accélère2`: $\delta'(a, (b, c, d), e) = \delta(\delta(a, b, c), d, e)$,
`Accélère3`: $\delta'((a, b, c), d, e) = \delta(c, d, e)$.

Côté droit

`Quiescent`: $\delta'((a, \sharp, \sharp), \sharp, \sharp) = \sharp$,
$\delta'(a, b, \$) = \delta(a, b, \sharp)$,
$\delta'(a, \$, \sharp) = \delta'((a, b, c), \$, \sharp) = \$$,
$\delta'(a, (b, c, d), \$) = \delta'(a, (b, c, d), \sharp)$,
$\delta'((a, b, c), d, \$) = \delta'((a, b, c), d, \sharp)$,
$\delta'(\$, \sharp, \sharp) = \sharp$.

Côté gauche et dernier pas

`Gauche1`: $\delta'(\sharp, a, (b, x, y)) = \begin{cases} \delta(\sharp, \delta(\sharp, a, b), \delta(a, b, x)) \text{ si } \delta(\sharp, \delta(\sharp, a, b), \delta(a, b, x)) \in \mathcal{S}_a \cup \mathcal{S}_r \\ \delta(\sharp, a, b) \text{ si } \delta(\sharp, a, b) \in \mathcal{S}_a \cup \mathcal{S}_r \\ (\sharp, \delta(\sharp, \delta(\sharp, a, b), \delta(a, b, x)), \delta(\delta(\sharp, a, b), \delta(a, b, x), y)) \text{ sinon} \end{cases}$

`Gauche2`: $\delta'(\sharp, (\sharp, a, b), c) = \begin{cases} \delta(\sharp, a, b) \text{ si } \delta(\sharp, a, b) \in \mathcal{S}_a \cup \mathcal{S}_r \\ (\sharp, \delta(\sharp, a, b), \delta(a, b, c)) \text{ sinon} \end{cases}$

`Gauche3`: $\delta'(\sharp, (\sharp, a, b), \$) = \delta'(\sharp, (\sharp, a, b), \sharp)$.

Généralisation

La méthode utilisée se généralise facilement à l'accélération de k pas mais nous ne détaillons pas la construction puisqu'un meilleur résultat est obtenu au paragraphe suivant avec une méthode assez similaire. La généralisation conduirait au résultat suivant :

Proposition 4.5 (acc. d'une constante pour reconnaisseurs, 3 ème forme)

Soient k un entier strictement positif et u un mot d'entrée.

Tout reconnaisseur fini à s états travaillant en temps $|u|+k+t(u)$ peut être simulé par un reconnaisseur fini à $s^{2.k+1}+s^{k+1}+1$ états travaillant en temps $|u|+t(u)$, sans synchronisation et avec une densité linéaire.

Notons que du point de vue de la quantité d'états, cette forme est moins bonne que celle de la proposition 4.3, en revanche, la densité est plus faible ; ainsi, cette méthode est un premier pas vers celle utilisée ci-après qui améliore *effectivement* la proposition 4.3.

3.2 Gagner un pas grâce à la méthode de la diagonale double

Il est courant de distribuer l'information d'une seule cellule parmi plusieurs dans le but d'économiser des états (voir par exemple [Yun94] et le chapitre 8). Le contexte est ici légèrement différent puisqu'il s'agit de mener un calcul sur la zone distribuée, sans le perturber. Il est malgré tout possible d'utiliser deux cellules au lieu d'une (et donc des couples d'états au lieu de triplets) pour construire la diagonale. C'est ce que nous appelons la méthode de la diagonale double.

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\langle 5,0\rangle$	$\sharp$
$\sharp$	$\langle 0,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\langle 5,1\rangle$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,2\rangle$	$\langle 1,2\rangle$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$ $\langle 5,2\rangle$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,3\rangle$	$\langle 1,3\rangle$	$\langle 2,3\rangle$	$\langle 3,3\rangle$ $\langle 4,3\rangle$	$\langle 5,3\rangle$ $\langle 5,4\rangle$	$\$$	$\sharp$
$\sharp$	$\langle 0,4\rangle$	$\langle 1,4\rangle$	$\langle 2,4\rangle$ $\langle 3,4\rangle$	$\langle 4,4\rangle$ $\langle 4,5\rangle$	$\langle 5,5\rangle$	$\$$	$\sharp$
$\sharp$	$\langle 0,5\rangle$	$\langle 1,5\rangle$ $\langle 2,5\rangle$	$\langle 3,5\rangle$ $\langle 3,6\rangle$	$\langle 4,6\rangle$	$\langle 5,6\rangle$	$\$$	$\sharp$
$\sharp$	$\langle 0,6\rangle$ $\langle 1,6\rangle$	$\langle 2,6\rangle$ $\langle 2,7\rangle$	$\langle 3,7\rangle$	$\langle 4,7\rangle$	$\langle 5,7\rangle$	$\$$	$\sharp$
$\sharp$	$\langle 0,8\rangle$ $\langle 1,8\rangle$	$\langle 2,8\rangle$	$\langle 3,8\rangle$	$\langle 4,8\rangle$	$\langle 5,8\rangle$	$\$$	$\sharp$
$\sharp$	$\langle 0,9\rangle$ $\langle 1,9\rangle$	$\langle 2,9\rangle$	$\langle 3,9\rangle$	$\langle 4,9\rangle$	$\langle 5,9\rangle$	$\$$	$\sharp$

FIG. 4.8 – *Méthode d'accélération efficace pour reconnaisseurs.*

La figure 4.8 montre la répartition des sites du diagramme espace-temps de l'automate cellulaire simulé dans le diagramme espace-temps de l'automate cellulaire simulant et illustre la méthode pour $|u| = 6$ et t variant entre 0 et 9.

La figure 4.9 montre l'effet du plongement du regroupeur dans le demi plan discret et son fonctionnement en tant qu'automate cellulaire.

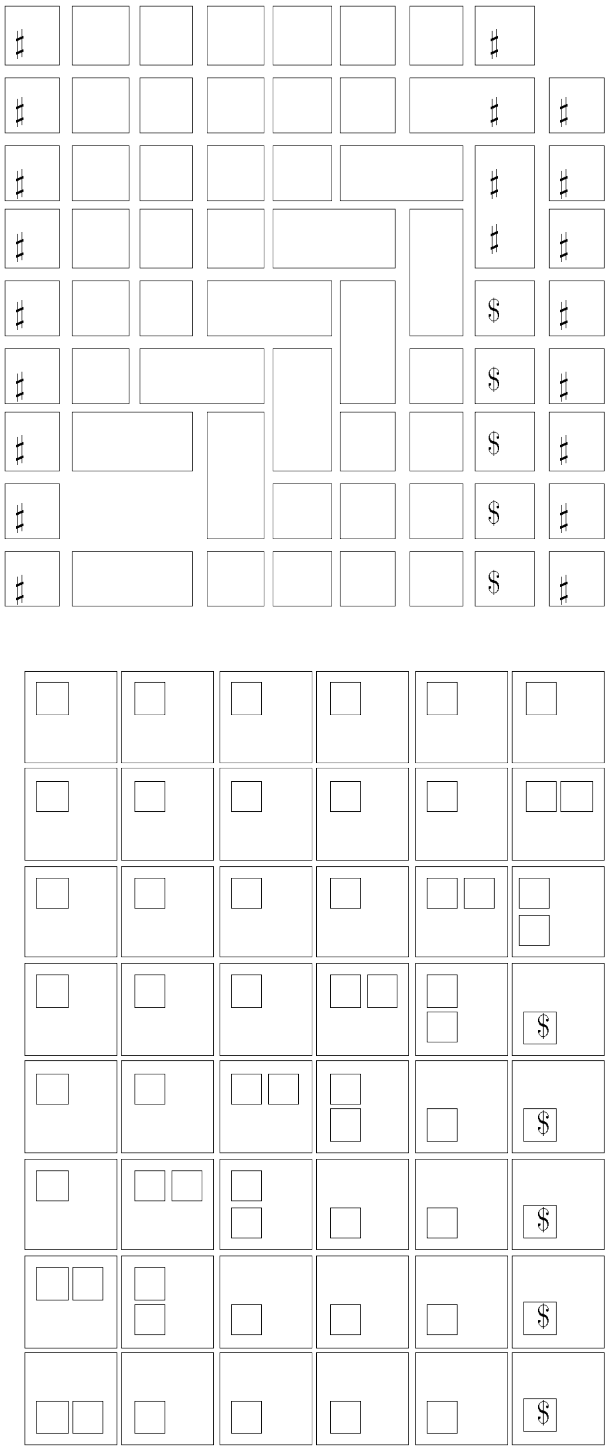


FIG. 4.9 – *Regroupeur réalisant l'accélération d'un pas par la méthode de la diagonale double (en haut le plongement, en bas le fonctionnement).*

Nous donnons maintenant la liste des règles de transition du regroupeur utilisé. Pour les principales d'entre elles, la preuve graphique de la faisabilité apparaît en figure 4.10.

Les règles marquées d'une étoile feront l'objet de remarques particulières juste après l'énumération. Par ailleurs, la valeur par défaut pour $\theta(x, y, z)$ est $\vec{s}$, nous n'indiquons que les valeurs différentes.

Ensemble des états

$$\mathcal{Q} = \left\{ \boxed{\scriptstyle G}, \square, \boxed{}, \boxed{\scriptstyle D}, \boxed{\vphantom{\Big|}}, \boxed{\substack{D\\D}}, \boxed{\scriptstyle \$}, \boxed{\scriptstyle D} \right\} .$$

Premiers pas

`Normal*`: $\delta_M(x, \square, \square) = \square$,
`Initialise`: $\delta_M(\square, \square, \boxed{\scriptstyle D}) = \boxed{\scriptstyle D}$,
`Accélère1`: $\delta_M(\square, \boxed{\scriptstyle D}, \boxed{\scriptstyle D}) = \boxed{\substack{D\\D}}$ avec $\theta = (1, 1)$,
`Vide`: $\delta_M(\boxed{}, \boxed{\substack{D\\D}}, \boxed{\scriptstyle D}) = \boxed{\scriptstyle \$}$ avec $\theta = (0, 2)$.

Diagonale

`Couple1`: $\delta_M(\square, \square, \boxed{}) = \boxed{}$,
`Accélère2`: $\delta_M(\square, \boxed{}, \boxed{\vphantom{\Big|}}) = \boxed{\vphantom{\Big|}}$ avec $\theta = (1, 1)$,
`Accélère3`: $\delta_M(\boxed{}, \boxed{\vphantom{\Big|}}, \square) = \square$ avec $\theta = (0, 2)$,
`Accélère4`: $\delta_M(\boxed{\vphantom{\Big|}}, \square, \square) = \square$.

Côté droit

L'état $\boxed{\scriptstyle \$}$ est équivalent à l'état $\boxed{\scriptstyle D}$ (ex. $\delta_M(\boxed{}, \boxed{}, \boxed{\scriptstyle \$}) = \square$, $\delta_M(\square, \square, \boxed{\scriptstyle \$}) = \delta(\square, \square, \boxed{\scriptstyle D}) \ldots$) sauf pour $\delta_M(\square, \boxed{\scriptstyle \$}, \boxed{\scriptstyle D}) = \boxed{\scriptstyle \$}$

Côté gauche et dernier pas

`AccélèreG`: $\delta_M(\boxed{\scriptstyle G}, \boxed{}, \boxed{\vphantom{\Big|}}) = \boxed{}$ avec $\theta = (0, 2)$,
`Gauche1*`: $\delta_M(\boxed{\scriptstyle G}, \square, \boxed{}) = \square$, avec $\theta = \vec{s}$ si le calcul de A termine à cette étape, $\theta = 2.\vec{s}$, s'il termine à la prochaine étape et $\delta_M(\boxed{\scriptstyle G}, \square, \boxed{}) = \boxed{}$ dans les autres cas.
`Gauche2*`: $\delta_M(\boxed{\scriptstyle G}, \boxed{}, x) = \begin{cases} \square \text{ avec } \theta = \vec{s} \text{ si } A \text{ termine,} \\ \boxed{} \text{ avec } \theta = \vec{s} \text{ sinon.} \end{cases}$

Remarques

- dans la règle `Normal`, la variable x désigne n'importe quelle pièce, en pratique $\boxed{\scriptstyle G}$ou $\square$;
- *stricto sensu* la règle `Gauche1` n'est pas une règle de regroupement, en effet, la valeur de la fonction de positionnement change selon que le calcul simulé termine au moment de l'utilisation de la règle, au moment suivant, ou plus tard ; cette entorse permet la restitution d'un résultat appartenant bien à $\mathcal{S}_a \cup \mathcal{S}_r$ (et non pas à $(\mathcal{S}_a \cup \mathcal{S}_r) \times \mathcal{Q}$) ; le regroupeur n'est pas dénaturé par cette exception puisqu'elle n'intervient qu'à la fin de la simulation (et n'a donc pas de conséquences) ;
- la même remarque s'applique à la règle `Gauche2`.

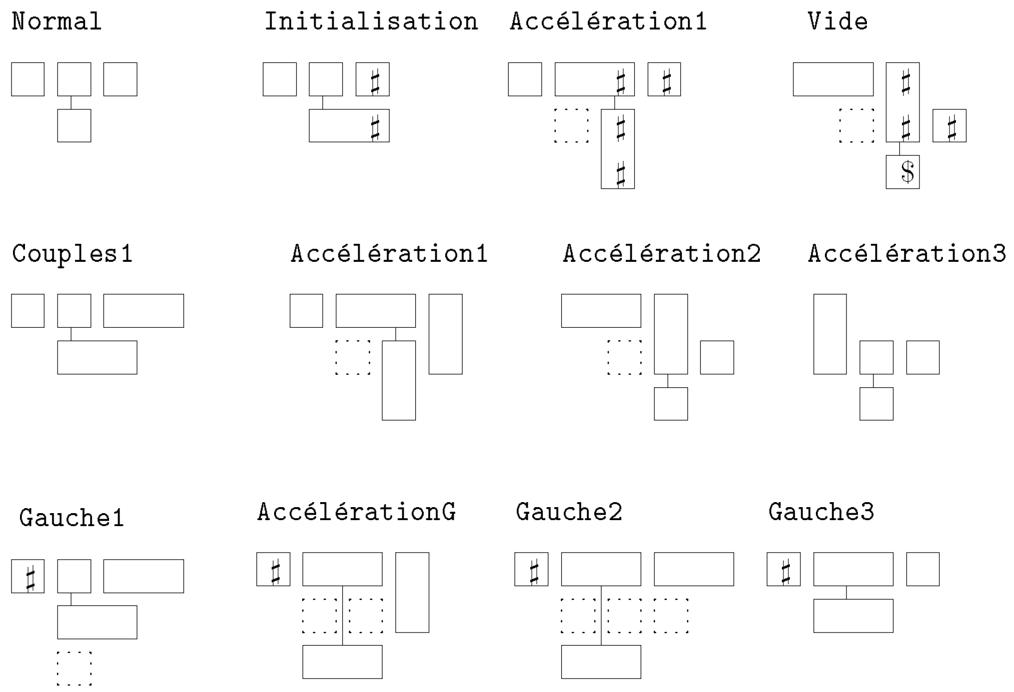


FIG. 4.10 – *Vérification de la faisabilité pour les principales règles de la méthode de la diagonale double.*

Voici la propriété qui garantit que le regroupeur vérifie bien la cohérence faible et qu'il réalise effectivement une accélération.

Lemme 4.6 (correction) *Notons* $c_0 = \boxed{\scriptstyle G}.\square^n.\boxed{\scriptstyle D}$ *la configuration initiale du regroupeur. Au temps* $t > 0$*, l'image - plongée dans le demi-plan discret - de* c_0 *vaut :*

$$Si\ t = 1 c_1 = \boxed{\scriptstyle G}.\square^{n-1}.\boxed{\scriptstyle D}.\boxed{\scriptstyle D} \text{ avec } v_{0,1} = \vec{s}$$

$$Si\ t = 2 c_2 = \boxed{\scriptstyle G}.\square^{n-2}.\boxed{}.\boxed{\begin{smallmatrix}D\\D\end{smallmatrix}}.\boxed{\scriptstyle D} \text{ avec } v_{0,2} = \vec{s}$$

$$Si\ 3 \leq t \leq n, c_t = \boxed{\scriptstyle G}.\square^{n-t}.\boxed{}.\boxed{\begin{smallmatrix}\\\end{smallmatrix}}.\square^{t-2}.\boxed{\scriptstyle \$}.\boxed{\scriptstyle D} \text{ avec } v_{0,t} = t.\vec{s}$$

$$Si\ t > n, c_t = \boxed{\scriptstyle G}.\boxed{}.\square^{n-2}.\boxed{\scriptstyle \$}.\boxed{\scriptstyle D} \text{ avec } v_{0,t} = (t+1).\vec{s}$$

Preuve

Si $t = 1$

Les seules règles applicables sont `Normal` et `Initialise`, la configuration au temps 1 est donc :

$$\Delta(c_0) = \boxed{\scriptstyle G}(\square)^{|u|-1}\boxed{}\boxed{\scriptstyle D} = c_1(u)$$

Puis, par application des règles `Normal` et `Accélère1` et on obtient :

$$\Delta^2(c_0) = \boxed{\text{G}}(\square)^{|u|-2}\boxed{}.\boxed{\substack{\text{D}\\ \text{D}}}.\boxed{\text{D}} = c_2(u)$$

Si $2 \leq t \leq n-1$

Cas de base

En recensant les règles applicables à partir de c_2 on trouve notamment le groupe de règles de bord droit, qui permettent l'apparition de la pièce $\boxed{\$}$. On vérifie facilement que $\Delta^3(c_0)$ est bien de la forme de c_t (pour $t = 3$).

Étape de récurrence

Appliquons Δ à c_t ; de gauche à droite de la configuration, seules les règles `Normal`, `Accélère2,3,4` et le groupe de règles de bord droit peuvent fonctionner ; on a :

$$\Delta(c_t) = \boxed{\text{G}}.\square^{n-t-1}.\boxed{}.\boxed{\vphantom{\Big|}\,}.\square^{t-1}.\boxed{\$}.\boxed{\text{D}}$$

on constate que cette expression vaut bien c_{t+1}.

Si $t \geq n$

Cas de base

Pour $t = n$, la configuration est le résultat de Δ appliquée à c_{t-1}, soit $\boxed{\text{G}}.\boxed{}.\boxed{\vphantom{\Big|}\,}.\square^{n-3}.\boxed{\$}.\boxed{\text{D}}$ avec $v_{0,n} = n.\vec{s}$. Dès lors, la règle spéciale `AccélèreG` s'applique conduisant, si le calcul de A ne termine pas à la configuration $\boxed{\text{G}}.\boxed{}.\square^{n-2}.\boxed{\$}.\boxed{\text{D}}$, qui vaut bien c_{n+1}. Par ailleurs, on sait que $\theta(\boxed{\text{G}}, \boxed{}, \boxed{\vphantom{\Big|}\,}) = 2.\vec{s}$ d'où : $v_{0,n+1} = (n+2).\vec{s}$ (et non pas $(n+1).\vec{s}$).

Étape de récurrence

Partant d'une configuration de la forme $\boxed{\text{G}}.\boxed{}.\square^{n-2}.\boxed{\$}.\boxed{\text{G}}$, il n'est pas possible d'appliquer d'autres règles que `Gauche2`, `Normal` et des règles de bord droit. L'application de ces règles ne modifie pas la forme de la configuration tant que le calcul de l'automate initial A ne termine pas.

Récapitulatif

En recensant systématiquement les règles applicables à partir de la configuration c_0 nous n'avons détecté aucun 3-agencement n'appartenant pas à F. Ceci prouve que le regroupeur possède la propriété de cohérence faible.

Par ailleurs, la valeur du vecteur de placement pour la cellule 0 est $(t+1).\vec{s}$ dès que $t \geq n$. En d'autres termes, le cellule 0 calcule avec une étape d'avance.

$\square$

3.3 Gagner k pas grâce à la méthode de la diagonale double

La méthode précédente s'étend directement au cas d'une accélération de k pas. Nous donnons les règles de transition en fonction du paramètre k dans le formalisme des regroupeurs tout d'abord puis, à titre de comparaison, dans le formalisme usuel. Pour de tels automates qui comportent un paramètre, la première méthode semble beaucoup plus claire.

Comme dans le paragraphe précédent, nous précisons la valeur de la fonction seulement θ lorsqu'elle diffère de la valeur par défaut (ici $\vec{s}$). Par ailleurs, la fonction diagramme espace-temps de l'automate cellulaire initial A est notée $\langle\rangle_A$.

Ensemble des états

$$\mathcal{Q}_M = \{\boxed{\mathrm{G}}, \square, \boxed{\quad k \quad}, \boxed{\begin{array}{c} k \\ \\ \end{array}}, \boxed{\mathrm{D}\ k\ \mathrm{D}}, \boxed{\$}, \boxed{\mathrm{D}}\} \ .$$

Premiers pas et côté droit

`Normal*`: $\delta_M(x, \square, \square) = \square$,

`Initialise1`: $\delta_M(\square, \square, \boxed{\mathrm{D}}) = \boxed{\mathrm{D}\ k\ \mathrm{D}}$,

`Initialise2`: $\delta_M(\square, \boxed{\mathrm{D}\ k\ \mathrm{D}}, \boxed{\mathrm{D}}) = \boxed{\begin{array}{c} \mathrm{D} \\ \mathrm{k} \\ \mathrm{D} \end{array}}$,

`Vide`: $\delta_M(\boxed{\quad k \quad}, \boxed{\begin{array}{c} \mathrm{D} \\ \mathrm{k} \\ \mathrm{D} \end{array}}, \boxed{\mathrm{D}}) = \boxed{\$}$.

L'état $\boxed{\$}$ est persistant (on a $\delta_M(x, \boxed{\$}, \boxed{\$}) = \delta_M(\boxed{\$}, \boxed{\$}, x) = \delta_M(\boxed{\$}, \boxed{\$}, \boxed{\$}) = \boxed{\$}$) et partout ailleurs il est équivalent à l'état $\boxed{\mathrm{D}}$.

Diagonale

`K-uplets`: $\delta_M(\square, \square, \boxed{\quad k \quad}) = \boxed{\quad k \quad}$,

`K-accélère1`: $\delta_M(\square, \boxed{\quad k \quad}, \boxed{\begin{array}{c} k \\ \\ \end{array}}) = \boxed{\begin{array}{c} k \\ \\ \end{array}}$,

`K-accélère2`: $\delta_M(\boxed{\quad k \quad}, \boxed{\begin{array}{c} k \\ \\ \end{array}}, \square) = \square$,

`K-accélère3`: $\delta_M(\boxed{\begin{array}{c} k \\ \\ \end{array}}, \square, \square) = \square$.

Côté gauche et dernier pas

`Gauche1`: $\delta_M(\boxed{\mathrm{G}}, \square, \boxed{\quad k \quad}) = \begin{cases} \square \text{ si } j = min\{0 \leq i < k, \langle 0, t+i\rangle_A \in \mathcal{S}_a \cup \mathcal{S}_r\} \text{ existe} \\ \boxed{\quad k \quad} \text{ sinon} \end{cases}$

`Gauche2`: $\delta_M(\boxed{\mathrm{G}}, \boxed{\quad k \quad}, \boxed{\begin{array}{c} k \\ \\ \end{array}}) = \begin{cases} \square \text{ si } \langle 0, t+k+1\rangle_A \in \mathcal{S}_a \cup \mathcal{S}_r \\ \boxed{\quad k \quad} \text{ sinon} \end{cases}$

`Gauche3`: $\delta_M(\boxed{\mathrm{G}}, \boxed{\quad k \quad}, \square) = \begin{cases} \square \text{ si } \langle 0, t+1\rangle_A \in \mathcal{S}_a \cup \mathcal{S}_r \\ \boxed{\quad k \quad} \text{ sinon} \end{cases}$

La figure 4.11 reprend graphiquement l'énumération ci-avant. Une telle figure contient toute l'information nécessaire à la constrution de la machine simulante, à l'exception des cas particulier lors de la terminaison du calcul qui, nous l'avons vu précédemment, ne relèvent pas entièrement du regroupement.

La preuve de correction se fait par récurrence sur le temps, et suit le même schéma que celle des lemmes 4.4 et 4.6. Ici, la généralisation ne complique pas le processus (d'où l'intérêt d'un exposé détaillé pour le cas $k = 2$).

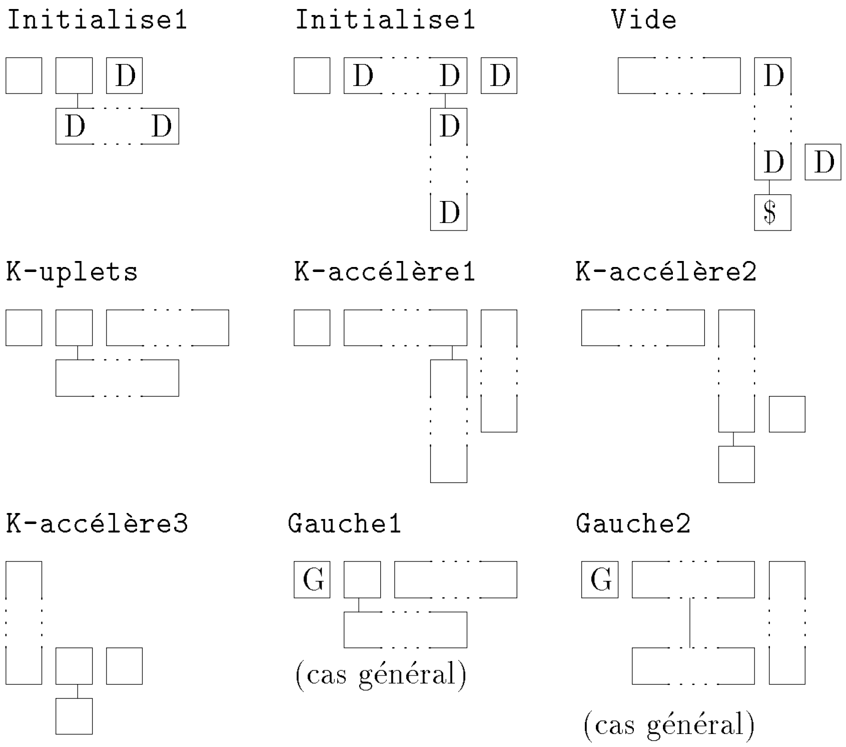


FIG. 4.11 – *Règles de regroupement pour l'accélération de k pas par la méthode de la diagonale double.*

Règles de simulation exprimées à l'aide du formalisme usuel

Ensemble des états

$\mathcal{Q}' = \mathcal{Q}^{k+1} \cup \overline{\mathcal{Q}^{k+1}} \cup \mathcal{Q} \cup \{\$\}$.

Premiers pas et côté droit

`Initialise`: $\delta'(\langle c,t\rangle\ , \langle c+1,t\rangle \ldots \langle c+k+1,t\rangle\ , \sharp) = \sharp^k$,

`Vide`: $\delta'(\langle c,t\rangle \ldots \langle c+k,t\rangle, \sharp^k, \sharp) = \$$.

L'état \$ est persistant (on a $\delta'(x,\$,\$) = \delta'(\$,\$,x) = \delta'(\$,\$,\$) = \$$) et partout ailleurs il est équivalent à l'état $\sharp$.

Diagonale

`K-uplets`: $\delta'(\langle c,t\rangle, \langle c+1,t\rangle, \langle c+2,t\rangle \ldots \langle c+k+2,t\rangle) =$

$$\langle c+1,t+1\rangle \ldots \langle c+k+1,t+1\rangle\ ,$$

`K-accélère1`:

$\delta'(\langle c,t\rangle, \langle c+1,t\rangle \ldots \langle c+k+1,t\rangle, \langle c+k+2,t\rangle \ldots \langle c+k+2,t+k\rangle) =$

$$\langle c+k+1,t+1\rangle \ldots \langle c+k+1,t+k+1\rangle\ ,$$

`K-accélère2`:

$\delta'(\langle c,t\rangle \ldots \langle c+k,t\rangle, \langle c+k+1,t\rangle \ldots \langle c+k+1,t+k\rangle, \langle c+k+2,t+k\rangle) =$

$$\langle c+k+1,t+k+1\rangle\ ,$$

`K-accélère3`: $\delta'(\langle c,t\rangle \ldots \langle c,t+k\rangle, \langle c+1,t+k\rangle, \langle c+2,t+k\rangle) = \langle c+1,t+k+1\rangle$.

Côté gauche et dernier pas

`Gauche1`: $\delta'(\sharp, \langle 0,t\rangle, \langle 1,t\rangle \ldots \langle k+1,t\rangle) =$

$$\begin{cases} \langle 0,j\rangle \text{ si } j = min\{i/\langle 0,t+i\rangle \in \mathcal{S}_a \cup \mathcal{S}_r\} \text{ existe} \\ \langle 0,t+1\rangle \ldots \langle k,t+1\rangle \text{ sinon} \end{cases}$$

`Gauche2`: $\delta'(\sharp, \langle 0,t\rangle \ldots \langle k,t\rangle, \langle k+1,t\rangle \ldots \langle k+1,t+k\rangle) =$

$$\begin{cases} \langle 0,t+k+1\rangle \text{ si } \langle 0,t+k+1\rangle \in \mathcal{S}_a \cup \mathcal{S}_r \\ \langle 0,t+k+1\rangle \ldots \langle k,t+k+1\rangle \text{ sinon} \end{cases}$$

`Gauche3`: $\delta'(\sharp, \langle 0,t\rangle \ldots \langle k,t\rangle, \langle k+1,t\rangle) = \begin{cases} \langle 0,t+1\rangle \text{ si } \langle 0,t+1\rangle \in \mathcal{S}_a \cup \mathcal{S}_r \\ \langle 0,t+1\rangle \ldots \langle k,t+1\rangle \text{ sinon} \end{cases}$

Récapitulatif

La méthode ci-avant nous donne une nouvelle forme autorisant le gain de k pas de calcul sans avoir recours à la composition de k accélérateurs successivement. La proposition est la forme la plus aboutie du théorème d'accélération d'une constante *en version forte*.

Proposition 4.7 (acc. d'une constante pour reconnaisseurs, 4 ème forme)

Soient k un entier strictement positif et u un mot d'entrée.

Tout reconnaisseur fini à s états travaillant en temps $|u| + k + t(u)$ peut être simulé par un reconnaisseur fini à $s^{k+1} + s + 1$ états travaillant en temps $|u| + t(u)$, sans synchronisation et avec une densité linéaire.

Remarqons que cette forme est équivalente à celle de la proposition 4.3 du point de vue du nombre d'états mais semble plus proche du problème de l'accélération d'une constante; celui-ci n'apparaît plus comme un simple cas particulier de l'accélération linéaire puisque la technique de preuve est entièrement différente, en particulier elle ne fait appel à aucune synchronisation et la densité est *linéaire*.

Une amélioration supplémentaire du nombre d'états est tentée au paragraphe 4.1 ci-après avec la méthode de la diagonale étendue mais n'aboutira qu'à une version *faible* du théorème (suffisante dans beaucoup de cas cependant).

4 Autres aspects de l'accélération d'une constante

4.1 Minimiser les ressources

Parmi les méthodes énoncées dans les sections précédentes, les plus économes exigent un accroissement du nombre des états de s à s^{k+1} pour gagner k pas de calcul.

Il est possible de faire mieux mais alors le théorème obtenu ne s'applique qu'aux machines dont le temps de calcul est « suffisamment long » (cette notion est précisée dans les hypothèses du théorème 4.8 donné ci-après). C'est ce que nous appelons la version faible du théorème d'accélération d'une constante, que nous donnons sous deux formes.

Théorème 4.8 (acc. d'une constante pour reconnaisseurs, version faible)

Soient k un entier strictement positif et u un mot d'entrée.

Première forme : *tout reconnaisseur à s états travaillant en temps $|u| + k/c + t(u) + k$*

peut être simulé par un reconnaisseur de même nature à $s^c + \mathcal{O}(k)$ *états travaillant en temps* $|u| + k/c + t(u)$*.*
Deuxième forme : *tout reconnaisseur à* s *états travaillant en temps* $|u| + k/c + t(u)$ *peut être simulé par un reconnaisseur de même nature à* $s^c + \mathcal{O}(k)$ *états travaillant en temps* $|u| + k/c + \sup\{0, t(u) - k\}$*.*

La figure 4.12 montre les deux techniques de preuve envisagées. La première méthode, probablement la plus simple, utilise le théorème d'accélération linéaire et une synchronisation pour calculer c fois plus vite pendant $k/(c-1)$ pas, avant de revenir à la vitesse normale (c'est l'affaiblissement naturel de la méthode exposée à la section 2 de ce chapitre). Ainsi, l'accroissement du nombre d'états n'excède pas $\mathcal{O}(s^c)$. Nous ne développerons pas plus avant cette méthode. Elle repose en effet sur le fait que l'accélération d'une constante est un cas particulier d'accélération linéaire et conduirait donc à une simulation où la densité serait quadratique.

La seconde méthode donne de meilleurs résultats du point de vue de la densité et permet d'éviter la synchronisation ; c'est la méthode dite de la diagonale étendue. La suite de la section lui est consacrée.

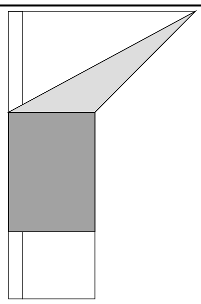
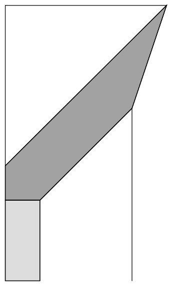
Schéma		
Méthode	Regroupement	Diagonale étendue
Nombre d'états	ks^2	$k + s^2$
Densité	Type surface	Linéaire

FIG. 4.12 – *Accélération d'une constante, version faible : techniques de preuve.*

4.2 Exemple simple de diagonale étendue

La figure 4.13 ci-après présente un cas simple de fonctionnement de la simulation pour $k = 2$, $n = 8$ et t variant entre 0 et 11. L'astérisque repère le temps de synchronisation $|u| + k$ utile pour revenir à la vitesse de calcul normale dans la partie gauche du diagramme. Nous ne spécifions pas les règles de transition puisque le cas général est détaillé au paragraphe suivant. Notons que l'ensemble des nouveaux états vaut : $\mathcal{Q}' = \mathcal{Q}^2 \cup \overline{\mathcal{Q}^2} \cup \mathcal{Q} \cup \{0, 1, 2, \$\}$.

4.3 Cas général pour la méthode de la diagonale étendue

Nous montrons comment gagner k pas en n'utilisant que des paires d'états et avec une densité linéaire. On notera que l'ensemble de cette solution n'est pas un pur regroupement (mais elle s'en rapproche). En effet, la composition avec un compteur sur la cellule de droite et un synchroniseur dans la partie gauche de la ligne ne s'expriment pas comme un regroupement.

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\langle 5,0\rangle$	$\langle 6,0\rangle$	$\langle 7,0\rangle$	$\sharp$
$\sharp$	$\langle 0,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\langle 5,1\rangle$	$\langle 6,1\rangle$	$\langle 7,1\rangle\sharp$	$\sharp$
$\sharp$	$\langle 0,2\rangle$	$\langle 1,2\rangle$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$	$\langle 5,2\rangle$	$\langle 6,2\rangle\langle 7,2\rangle$	0	$\sharp$
$\sharp$	$\langle 0,3\rangle$	$\langle 1,3\rangle$	$\langle 2,3\rangle$	$\langle 3,3\rangle$	$\langle 4,3\rangle$	$\langle 5,3\rangle\langle 6,3\rangle$	$\langle 7,3\rangle$ $\langle 7,4\rangle$	1	$\sharp$
$\sharp$	$\langle 0,4\rangle$	$\langle 1,4\rangle$	$\langle 2,4\rangle$	$\langle 3,4\rangle$	$\langle 4,4\rangle\langle 5,4\rangle$	$\langle 6,4\rangle$ $\langle 6,5\rangle$	$\langle 7,5\rangle\sharp$	2	$\sharp$
$\sharp$	$\langle 0,5\rangle$	$\langle 1,5\rangle$	$\langle 2,5\rangle$	$\langle 3,5\rangle\langle 4,5\rangle$	$\langle 5,5\rangle$ $\langle 5,6\rangle$	$\langle 6,6\rangle\langle 7,6\rangle$	$\sharp$ $\sharp$	$\$$	$\sharp$
$\sharp$	$\langle 0,6\rangle$	$\langle 1,6\rangle$	$\langle 2,6\rangle\langle 3,6\rangle$	$\langle 4,6\rangle$ $\langle 4,7\rangle$	$\langle 5,7\rangle\langle 6,7\rangle$	$\langle 7,7\rangle$ $\langle 7,8\rangle$	$\$$	$\$$	$\sharp$
$\sharp$	$\langle 0,7\rangle$	$\langle 1,7\rangle\langle 2,7\rangle$	$\langle 3,7\rangle$ $\langle 3,8\rangle$	$\langle 4,8\rangle\langle 5,8\rangle$	$\langle 6,8\rangle$ $\langle 6,9\rangle$	$\langle 7,9\rangle$	$\$$	$\$$	$\sharp$
$\sharp$	$\langle 0,8\rangle\langle 1,8\rangle$	$\langle 2,8\rangle$ $\langle 2,9\rangle$	$\langle 3,9\rangle\langle 4,9\rangle$	$\langle 5,9\rangle$ $\langle 5,10\rangle$	$\langle 6,10\rangle$	$\langle 7,10\rangle$	$\$$	$\$$	$\sharp$
$\sharp$	$\langle 0,10\rangle\langle 1,10\rangle$	$\langle 2,10\rangle\langle 3,10\rangle$	$\langle 4,10\rangle$ $\langle 4,11\rangle$	$\langle 5,11\rangle$	$\langle 6,11\rangle$	$\langle 7,11\rangle$	$\$$	$\$$	$\sharp$
$\sharp$	$\langle 0,12\rangle\langle 1,12\rangle$	$\langle 2,12\rangle\langle 3,12\rangle$	$\langle 4,12\rangle$	$\langle 5,12\rangle$	$\langle 6,12\rangle$	$\langle 7,12\rangle$	$\$$	$\$$	$\sharp*$
$\sharp$	$\langle 0,13\rangle\langle 1,13\rangle$	$\langle 2,13\rangle\langle 3,13\rangle$	$\langle 4,13\rangle$	$\langle 5,13\rangle$	$\langle 6,13\rangle$	$\langle 7,13\rangle$	$\$$	$\$$	$\sharp$

FIG. 4.13 – *Affaiblissement de la méthode de la diagonale.*

Nous travaillons à une première généralisation qui autoriserait les produits avec interaction avec d'autres automates cellulaires (*cf.* chapitre 9). Par ailleurs, la compasoante regroupeur de la machine décrite ci-après est presque identique au regroupeur décrit en section 3.3 de ce chapitre.

Nous proposons donc une description complète de l'automate cellulaire dans le formalisme usuel.

Ensemble des états

$\mathcal{Q}' = \mathcal{Q}^2 \cup \overline{\mathcal{Q}^2} \cup \mathcal{Q} \cup \{0 \ldots 2(k-1), \$\}$.

Premiers pas

`Initialise1`: $\delta'(a, b, \sharp) = (\delta(a, b, \sharp), \sharp)$,

`Compte0`: $\delta'(a, (b, \sharp), \sharp) = 0$,

`CompteI`: $\delta'((a, b), i, \sharp) = \delta'(\overline{(a, b)}, i, \sharp) = i + 1, (i \leq 2(k-1))$,

`CompteK`:
$\delta'(\overline{(a, b)}, 2(k-1), \sharp) = \delta'(\overline{(a, b)}, \$, \sharp) = \delta'(\overline{(a, b)}, \$, \$) = \delta'(a, \$, \sharp) = \delta'(a, \$, \$) = \$$,

`Règles de bord`: $\delta'(\$, \sharp, \sharp) = (\sharp, \delta'(\$, \$, \sharp)) = \delta'(\$, \$, \$) = \$$,

`Initialise2`: $\delta'((a, b), \overline{(c, d)}, i) = \delta(\delta(a, b, c), d, \sharp), \sharp)$,

`Initialise3`: $\delta'(\overline{(a, b)}, (c, d), i) = \overline{(\delta(\delta(a, b, c), c, d), \sharp)}$.

Diagonale (partie gauche)

`Couple`: $\delta'(a, b, (c, d)) = (\delta(a, b, c), \delta(b, c, d))$,

`Accélère1`: $\delta'(a, (b, c), \overline{(d, e)}) = \overline{(\delta(b, c, d), \delta(\delta(a, b, c), \delta(b, c, d), \delta(c, d, e)))}$.

Diagonale (partie centrale)

$\texttt{Accélère2}$: $\delta'((a,b),\overline{(c,d)},(e,f)) = \delta(\delta(a,b,c),d,e),\delta(d,e,f))$,

$\texttt{Accélère3}$: $\delta'(\overline{(a,b)},(c,d),\overline{(e,f)}) = \overline{\delta(c,d,e),\delta(\delta(b,c,d),\delta(c,d,e),f)}$.

Diagonale (partie droite)

$\texttt{Accélère4}$: $\delta'((a,b),\overline{(c,d)},e) = \delta(\delta(a,b,c),d,e)$,

$\texttt{Accélère5}$: $\delta'(\overline{(a,b)},d,e) = \delta(b,c,d)$.

Derniers pas (avant synchronisation)

$\texttt{Rapide1}$: $\delta'(\sharp,(a,b),\overline{(c,d)}) = (\delta(\sharp,\delta(\sharp,a,b),\delta(a,b,c)),\delta(\delta(\sharp,a,b),\delta(a,b,c),d))$,

$\texttt{Rapide2}$: $\delta'(\sharp,(a,b),(c,d)) = (\delta(\sharp,\delta(\sharp,a,b),\delta(a,b,c)),\delta(\delta(\sharp,a,b),\delta(a,b,c),\delta(b,c,d)))$,

$\texttt{Rapide3}$: $\delta'((a,b),(c,d),\overline{(e,f)}) =$
$(\delta(\delta(a,b,c),\delta(b,c,d),\delta(c,d,e)),\delta(\delta(b,c,d),\delta(c,d,e),f))$,

$\texttt{Rapide4}$: $\delta'((a,b),(c,d),(e,f)) =$
$(\delta(\delta(a,b,c),\delta(b,c,d),\delta(d,e,f)),\delta(\delta(b,c,d),\delta(c,d,e),\delta(d,e,f)))$.

Derniers pas (après synchronisation)

$\texttt{Lent1}$: $\delta'(\sharp,(a,b),(c,d)) = (\delta(\sharp,a,b),\delta(a,b,c))$,

$\texttt{Lent2}$: $\delta'((a,b),(c,d),(e,f)) = (\delta(b,c,d),\delta(c,d,e))$,

$\texttt{Lent3}$: $\delta'((a,b),(c,d),e) = (\delta(b,c,d),\delta(c,d,e))$.

Un phénomène intéressant a lieu lorsque la diagonale atteint le bord gauche du diagramme espace-temps: puisque la diagonale est épaisse, la partie initiale heurte déjà le bord gauche alors que la traîne doit encore voyager. Dès que le bord est atteint, les cellules concernées se mettent à calculer deux fois plus vite (elles ont maintenant assez d'information pour le faire). Le reste de la diagonale atteint alors cette zone de calcul rapide qui réagit comme un nouveau bord, et ainsi de suite...

La partie technique vient du fait que le calcul rapide doit se terminer *partout* au même temps $|u| + k$ pour que l'automate revienne à sa vitesse normale. C'est typiquement un problème de synchronisation, qui peut être résolu de manière classique : synchroniser en temps n et demander à chaque cellule de décompter k étapes avant de revenir à la vitesse normale. Cette méthode conduit à une solution en $\mathcal{O}(k.s^2)$ états, équivalente à ce que donnerait celle esquissée au paragraphe 4.8.

En fait on peut remarquer qu'il n'est pas nécessaire d'utiliser $k.s^2$ états pour lancer la diagonale étendue: $s^2 + k$ suffisent. Par conséquent, il serait intéressant de synchroniser la partie rapide (à gauche du diagramme) en utilisant seulement $s^2 + k$ états.

On peut le faire en lançant un signal à vitesse maximale, c'est-à-dire se déplaçant d'une cellule par unité de temps, vers la gauche au temps $\lceil k/2 \rceil$ à partir de la cellule d'indice $n - 1$ (voir point A sur la figure 4.14). L'intersection (au point B) de ce signal et d'une synchronisation au temps n particularise le point extrémal de la zone à synchroniser en k étapes. Il ne reste alors qu'à lancer une synchronisation en temps minimal à partir des points B et C sur la figure, ce qui ne nécessite qu'un nombre constant d'états (indépendant de k).

Remarque

D'autres techniques d'économie sont peut être envisageables pour réduire le nombre d'états auxiliaires et le sujet n'est pas clos [Hee96a].

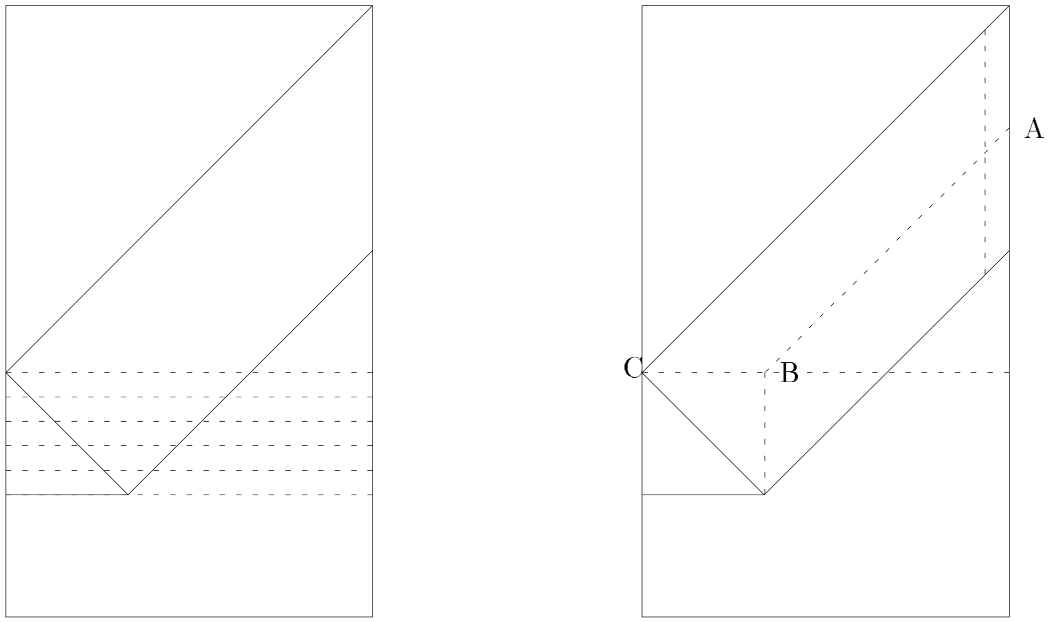


FIG. 4.14 – *À gauche : méthode à $k.s^2$ états. À droite : méthode à $k + s^2$ états.*

4.4 Cas des machines infinies

Les machines finies ne sont pas universelles puisqu'il n'y a pas plus de $card(Q)^{|u|}$ configurations différentes. En revanche les machines bi-infinies et semi-infinies le sont [LN90, Mar94, Smi71b]. Il peut donc être intéressant d'obtenir des résultats sur l'accélération dans ce contexte. Nous montrons comment quelques uns des théorèmes précédents s'étendent au cas semi-infini, tandis que le cas bi-infini est obtenu *via* le lemme technique 2.2 avec augmentation quadratique du nombre d'états.

L'idée consiste à lancer deux diagonales : une pour la partie finie (à gauche), l'autre pour la partie extensible (à droite). Puisque tout le diagramme espace-temps est « décalé » vers la gauche, un état de remplissage (\$) est nécessaire. Remarquons que le cas fini n'est pas tout à fait un cas particulier (la méthode de remplissage varie légèrement).

La figure 4.15 montre la répartition des sites du diagramme espace-temps de l'automate cellulaire simulé dans le diagramme espace-temps de l'automate cellulaire simulant et illustre la méthode pour $k = 1$, $n = 5$ et t variant de 0 à 6.

Pour l'ensemble des états et la diagonale gauche les transformations sont les mêmes que dans le cas fini. Les règles supplémentaires permettant de gérer la croissance de la ligne vers la droite sont les suivantes :

Premier pas et diagonale de droite

$\delta'(a, \sharp, \sharp) = \overline{(\sharp, \sharp)}$,

$\delta'((a, b), \overline{(\sharp, \sharp)}, \sharp) = \overline{\delta(\delta(a, b, \sharp), \delta(b, \sharp, \sharp), \sharp)}$,

$\delta'(\overline{(\sharp, \sharp)}, \sharp, \sharp) = \$$,

$\delta'(\overline{(a, b)}, c, \$) = \delta(b, c, \sharp)$,

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\sharp$	$\sharp$	$\sharp$	$\sharp$	$\sharp$
$\sharp$	$\langle 0,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,5\mid 1\rangle$	$\sharp$ $\sharp$	$\sharp$	$\sharp$	$\sharp$	$\sharp$
$\sharp$	$\langle 0,2\rangle$	$\langle 1,2\rangle$	$\langle 2,2\rangle$	$\langle 3,4\mid 2\rangle$	$\langle 5,2\rangle$ $\langle 5,3\rangle$	$\langle 6,3\rangle$	\$	$\sharp$	$\sharp$	$\sharp$
$\sharp$	$\langle 0,3\rangle$	$\langle 1,3\rangle$	$\langle 2,3\mid 3\rangle$	$\langle 4,3\rangle$ $\langle 4,4\rangle$	$\langle 5,4\rangle$	$\langle 6,4\rangle$	$\langle 7,4\rangle$	\$	$\sharp$	$\sharp$
$\sharp$	$\langle 0,4\rangle$	$\langle 1,2\mid 4\rangle$	$\langle 3,4\rangle$ $\langle 3,5\rangle$	$\langle 4,5\rangle$	$\langle 5,5\rangle$	$\langle 6,5\rangle$	$\langle 7,5\rangle$	$\langle 8,5\rangle$	\$	$\sharp$
$\sharp$	$\langle 0,1\mid 5\rangle$	$\langle 2,5\rangle$ $\langle 2,6\rangle$	$\langle 3,6\rangle$	$\langle 4,6\rangle$	$\langle 5,6\rangle$	$\langle 6,6\rangle$	$\langle 7,6\rangle$	$\langle 8,6\rangle$	$\langle 9,6\rangle$	\$
$\sharp$	$\langle 0,1\rangle$	$\langle 2,7\rangle$	$\langle 3,7\rangle$	$\langle 4,7\rangle$	$\langle 5,7\rangle$	$\langle 6,7\rangle$	$\langle 7,7\rangle$	$\langle 8,7\rangle$	$\langle 9,7\rangle$	$\langle 10,7\rangle$

FIG. 4.15 – *Accélération d'un pas sur un reconnaisseur semi-infini.*

$$\delta'(a, b, \$) = \delta(a, b, \sharp) \ ,$$
$$\delta'(a, \$, \sharp) = \delta(a, \sharp, \sharp) \ ,$$
$$\delta'(\$, \sharp, \sharp) = \$ \ .$$

Ceci conduit au résultat suivant :

Proposition 4.9 (acc. d'une constante pour reconnaisseurs semi-infinis)
Soient k un entier strictement positif et u un mot d'entrée.

Tout reconnaisseur semi-fini à s états travaillant en temps $|u| + k + t(u)$ peut être simulé par un reconnaisseur semi-infini à $s^{k+1} + s + 1$ états travaillant en temps $|u| + t(u)$, sans synchronisation et avec une densité linéaire.

Conclusions

Les méthodes spécifiques développées dans ce chapitre montrent que l'accélération d'une constante n'est pas seulement un cas particulier d'accélération linéaire : la synchronisation n'est pas indispensable, la densité peut être rendue linéaire. Pourtant, nous ne savons toujours pas si l'on peut réaliser l'accélération d'une constante *en version forte*, avec moins de nouveaux états que n'en nécessite l'accélération linéaire.

En pratique, la version faible suffit dans presque tous les cas, puisque le surcroît de temps nécessaire ne dépend que de k et de c (et aucunement de la longueur du mot d'entrée).

Chapitre 5

Accélération d'une constante sur calculateurs

Résumé

La première preuve du théorème d'accélération d'une constante pour calculateurs nécessitait un accroissement exponentiel du nombre d'états. Nous proposons dans ce chapitre deux techniques de preuve qui rendent l'accroissement polynomial. De plus, la seconde technique n'utilise pas de synchronisation et la densité d'utilisation des nouveaux états est faible.

Remarquons que la facilité consistant à utiliser un théorème d'accélération linéaire en version forte pour aboutir au résultat n'est plus possible dans le cadre des calculateurs puisqu'un tel théorème n'est pas encore connu.

Introduction

Nous énonçons tout d'abord le théorème, sans prendre en compte l'accroissement du nombre d'états:

Théorème 5.1 (accélération d'une constante, calculateurs finis)

Soient k un entier strictement positif et u un mot d'entrée.
Première forme : *tout calculateur fini travaillant en temps $|u| + t(u) + k$ peut être simulé (à renommage près des sorties) par un calculateur fini travaillant en temps $|u| + t(u)$.*
Deuxième forme : *tout calculateur fini travaillant en temps $|u| + t(u)$ peut être simulé (à renommage près des sorties) par un calculateur fini travaillant en temps $|u| + \sup\{0, t(u) - k\}$.*

Du point de vue de l'accélération, le cas des calculateurs est plus difficile que celui des reconnaisseurs. En effet, la machine simulante doit être, à la fin du calcul, dans le même état que la machine simulée *sur toute la longueur de la bande* (et non pas seulement sur la cellule zéro).

Les méthodes de regroupement simples développées au chapitre précédent ne sont donc pas directement applicables ici. On pourrait imaginer d'adapter un « dégroupeur » Mis en marche dès que la cellule zéro décrète la fin du calcul. De tels automates existent mais ils doivent être démarrés uniformément, ce qui suppose une synchronisation à temps. De plus, ils nécessitent $|u|$ étapes pour opérer. Cette idée conduirait éventuellement à un temps de simulation $2|u| + t(u)$ pour une machine travaillant initialement en temps $|u| + t(u) + k$, et donc à une version faible du théorème. Cette méthode est donc à écarter.

Nous pouvons néanmoins adapter partiellement les principes mis en œuvre dans l'accélération linéaire des reconnaisseurs, ce qui conduit à la méthode par redondance, « vorace » en ressources, que nous décrivons en section 2.

De toutes les techniques évoquées dans le chapitre précédent, celle qui perturbe le moins la configuration d'arrivée est la méthode de la diagonale. En effet, à partir du temps n, seule la cellule zéro est k-groupée, les autres évoluant comme dans la machine initiale. Cette idée est développée en section 3 et c'est elle qui aboutit au meilleur résultat.

1 Méthode de Choffrut et Čulik

Dans [CČ83] Choffrut et Čulik donnent une preuve du théorème d'accélération d'un pas pour les calculateurs finis (et *a fortiori* pour les reconnaisseurs).

Un calculateur fini $C = ((\mathcal{Q}, \delta), \mathcal{T})$ est simulé ainsi : chaque cellule simulante gère son état courant et tient à jour une fonction $G : \mathcal{Q} \times \mathcal{Q} \to \mathcal{Q}$ qui sert « d'oracle » : au temps t, si x et y valaient $\langle i-t-2, 0\rangle$ et $\langle i+t+2, 0\rangle$ alors $G(x, y)$ vaudrait $\langle i, t+1\rangle$ (en d'autres termes, G aurait prédit le nouvel état de la cellule).

Bien entendu, dans le cas général, une cellule ne peut pas deviner les valeurs adéquates de x et y, mais pour des temps t suffisamment grands on a $\langle i-t-2, 0\rangle = \langle i-t+2, 0\rangle = \sharp$. Puisqu'il y a un nombre fini d'états dans $\mathcal{Q}$ (disons s), il y a aussi un nombre fini de fonctions G de $\mathcal{Q}^2$ vers $\mathcal{Q}$ (précisément s^{s^2}).

Un décompte attentif des états conduit au résultat suivant :

Proposition 5.2 (ACC, première forme) [CČ83]
Il existe une preuve du théorème 5.1 nécessitant un accroissement de s à $\mathcal{O}(s^3.s^{s^2})$ du nombre d'états.

Les sections suivantes exposent différentes techniques d'amélioration de cette borne.

2 Méthode par redondance

Un premier essai pourrait consister en l'utilisation d'une accélération linéaire avec synchronisation, comme au chapitre précédent, mais alors le résultat serait en configuration regroupée, ce qui n'est plus satisfaisant pour les calculateurs.

Quoiqu'il en soit, il est possible de fournir à chaque cellule la même information que dans une configuration regroupée, *sans* réduire la longueur de la ligne. Cette approche nous conduit à la méthode de redondance simple, suffisante pour accélérer un calculateur fini d'une constante (mais pas d'un facteur en revanche).

Notation

Afin de rendre les figures 5.1 et 5.2 plus lisibles nous utilisons la notation abrégée introduite au chapitre 2 concernant les cellules qui contiennent plusieurs états. Nous rappelons ici cette notation :

$$\langle c_1, c_2, c_3, \ldots | t\rangle = (\langle c_1, t\rangle, \langle c_2, t\rangle, \langle c_3, t\rangle, \ldots)$$
$$\langle a..b|t\rangle = (\langle a, t\rangle, \langle a+1, t\rangle, \ldots, \langle b-1, t\rangle, \langle b, t\rangle).$$

La figure 5.1 illustre le fonctionnement de la méthode par redondance dans le cas des lignes de longueur paire (nous avons choisi $|u| = 7$ et t variant entre 0 et 10). Les informations « avant », « sync. » et « après » dans les cases de droite ne sont pas des états. Ce sont des indications quant au déroulement du processus de synchronisation en temps minimal concernant toutes les cellules de la ligne. Notons que cette synchronisation est essentielle puisque c'est elle qui ordonne aux cellules d'effectuer deux pas de calcul en une unité de temps au temps $|u|$. Par la suite, le processus continue à vitesse normale et devient donc équivalent au processus initial, à un seul détail près : toutes les cellules utilisent des états renommés puisqu'elles possèdent une information indiquant que la synchronisation à déjà eu lieu. Cette précaution permet au processus de regroupement de ne pas redémarrer aux temps $|u|$, $2.|u|$, $3.|u|$ ... (voir la remarque suivant la définition 2.6 au chapitre 2).

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\langle 5,0\rangle$	$\langle 6,0\rangle$	$\langle 7,0\rangle$	$\sharp$	avant
$\sharp$	$\sharp\ \langle 0,1\rangle \bullet$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\langle 5,1\rangle$	$\langle 6,1\rangle$	$\bullet\ \langle 7,1\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\ \langle 0,2\rangle \bullet$	$\langle 0,1\mid 2\rangle \bullet$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$	$\langle 5,2\rangle$	$\bullet\ \langle 6,7\mid 2\rangle$	$\bullet\ \langle 7,2\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\ \langle 0,3\rangle \bullet$	$\langle 0,1\mid 3\rangle \bullet$	$\langle 1,2\mid 3\rangle \bullet$	$\langle 3,3\rangle$	$\langle 4,3\rangle$	$\bullet\ \langle 5,6\mid 3\rangle$	$\bullet\ \langle 6,7\mid 3\rangle$	$\bullet\ \langle 7,3\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\ \langle 0,4\rangle \bullet$	$\langle 0,1\mid 4\rangle \bullet$	$\langle 1,2\mid 4\rangle \bullet$	$\langle 2,3\mid 4\rangle \bullet$	$\bullet\ \langle 4,5\mid 4\rangle$	$\bullet\ \langle 5,6\mid 4\rangle$	$\bullet\ \langle 6,7\mid 4\rangle$	$\bullet\ \langle 7,4\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\ \langle 0,5\rangle \bullet$	$\langle 0,1\mid 5\rangle \bullet$	$\langle 1,2\mid 5\rangle \bullet$	$\langle 2..4\mid 5\rangle$	$\langle 3..5\mid 5\rangle$	$\bullet\ \langle 5,6\mid 5\rangle$	$\bullet\ \langle 6,7\mid 5\rangle$	$\bullet\ \langle 7,5\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\ \langle 0,6\rangle \bullet$	$\langle 0,1\mid 6\rangle \bullet$	$\langle 1..3\mid 6\rangle$	$\langle 2..4\mid 6\rangle$	$\langle 3..5\mid 6\rangle$	$\langle 4..6\mid 6\rangle$	$\bullet\ \langle 5,6\mid 6\rangle$	$\bullet\ \langle 7,6\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\ \langle 0,7\rangle \bullet$	$\langle 0..2\mid 7\rangle$	$\langle 1..3\mid 7\rangle$	$\langle 2..4\mid 7\rangle$	$\langle 3..5\mid 7\rangle$	$\langle 4..6\mid 7\rangle$	$\langle 5..7\mid 7\rangle$	$\bullet\ \langle 7,7\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\ \langle 0,1\mid 8\rangle$	$\langle 0..2\mid 8\rangle$	$\langle 1..3\mid 8\rangle$	$\langle 2..4\mid 8\rangle$	$\langle 3..5\mid 8\rangle$	$\langle 4..6\mid 8\rangle$	$\langle 5..7\mid 8\rangle$	$\langle 6,7\mid 8\rangle\ \sharp$	$\sharp$	sync.
$\sharp$	$\langle 0,10\rangle$	$\langle 1,10\rangle$	$\langle 2,10\rangle$	$\langle 3,10\rangle$	$\langle 4,10\rangle$	$\langle 5,10\rangle$	$\langle 6,10\rangle$	$\langle 7,10\rangle$	$\sharp$	après
$\sharp$	$\langle 0,11\rangle$	$\langle 1,11\rangle$	$\langle 2,11\rangle$	$\langle 3,11\rangle$	$\langle 4,11\rangle$	$\langle 5,11\rangle$	$\langle 6,11\rangle$	$\langle 7,11\rangle$	$\sharp$	après

FIG. 5.1 – *Gain d'un pas de calcul sur un calculateur par la méthode de la redondance : cas d'une configuration initiale de longueur paire.*

Nous donnons également un exemple de fonctionnement dans le cas des lignes de longueur impaire. La figure 5.2 montre le diagramme espace-temps obtenu pour $|u| = 9$ et t variant entre 0 et 12. Remarquons que seul le *comportement* de l'automate cellulaire diffère en fonction de la parité de la ligne ; les règles de transitions, en revanche, sont toujours les mêmes. En particulier, la parité n'a pas besoin d'être connue à l'avance, le contexte autour de la cellule centrale suffit à distinguer les deux cas.

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\langle 5,0\rangle$	$\langle 6,0\rangle$	$\langle 7,0\rangle$	$\langle 8,0\rangle$	$\sharp$	avant
$\sharp$	$\sharp\langle 0,1\rangle \bullet$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\langle 5,1\rangle$	$\langle 6,1\rangle$	$\langle 7,1\rangle$	$\bullet\langle 8,1\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\langle 0,2\rangle \bullet$	$\langle 0,1\mid 2\rangle \bullet$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$	$\langle 5,2\rangle$	$\langle 6,2\rangle$	$\bullet\langle 7,8\mid 2\rangle$	$\bullet\langle 8,2\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\langle 0,3\rangle \bullet$	$\langle 0,1\mid 3\rangle \bullet$	$\bullet\langle 1,2\mid 3\rangle$	$\langle 3,3\rangle$	$\langle 4,3\rangle$	$\langle 5,3\rangle$	$\bullet\langle 6,7\mid 3\rangle$	$\bullet\langle 7,8\mid 3\rangle$	$\bullet\langle 8,3\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\langle 0,4\rangle \bullet$	$\langle 0,1\mid 4\rangle \bullet$	$\bullet\langle 1,2\mid 4\rangle$	$\bullet\langle 2,3\mid 4\rangle$	$\langle 4,4\rangle$	$\bullet\langle 4,5\mid 4\rangle$	$\bullet\langle 6,7\mid 4\rangle$	$\bullet\langle 7,8\mid 4\rangle$	$\bullet\langle 8,4\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\langle 0,5\rangle \bullet$	$\langle 0,1\mid 5\rangle \bullet$	$\bullet\langle 1,2\mid 5\rangle$	$\bullet\langle 2,3\mid 5\rangle$	$\langle 3..5\mid 5\rangle$	$\bullet\langle 4,5\mid 5\rangle$	$\bullet\langle 6,7\mid 5\rangle$	$\bullet\langle 7,8\mid 5\rangle$	$\bullet\langle 8,5\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\langle 0,6\rangle \bullet$	$\langle 0,1\mid 6\rangle \bullet$	$\bullet\langle 1,2\mid 6\rangle$	$\langle 2..4\mid 6\rangle$	$\langle 3..5\mid 6\rangle$	$\langle 4..6\mid 6\rangle$	$\bullet\langle 6,7\mid 6\rangle$	$\bullet\langle 7,8\mid 6\rangle$	$\bullet\langle 8,6\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\langle 0,7\rangle \bullet$	$\langle 0,1\mid 7\rangle \bullet$	$\langle 1..3\mid 7\rangle$	$\langle 2..4\mid 7\rangle$	$\langle 3..5\mid 7\rangle$	$\langle 4..6\mid 7\rangle$	$\langle 5..7\mid 7\rangle$	$\bullet\langle 7,8\mid 7\rangle$	$\bullet\langle 8,7\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\langle 0,8\rangle \bullet$	$\langle 0..2\mid 8\rangle$	$\langle 1..3\mid 8\rangle$	$\langle 2..4\mid 8\rangle$	$\langle 3..5\mid 8\rangle$	$\langle 4..6\mid 8\rangle$	$\langle 5..7\mid 8\rangle$	$\langle 6..8\mid 8\rangle$	$\bullet\langle 8,8\rangle\ \sharp$	$\sharp$	avant
$\sharp$	$\sharp\langle 0,1\mid 9\rangle$	$\langle 0..2\mid 9\rangle$	$\langle 1..3\mid 9\rangle$	$\langle 2..4\mid 9\rangle$	$\langle 3..5\mid 9\rangle$	$\langle 4..6\mid 9\rangle$	$\langle 5..7\mid 9\rangle$	$\langle 6..8\mid 9\rangle$	$\langle 7,8\mid 9\rangle\ \sharp$	$\sharp$	sync.
$\sharp$	$\sharp\langle 0,11\rangle$	$\langle 1,11\rangle$	$\langle 2,11\rangle$	$\langle 3,11\rangle$	$\langle 4,11\rangle$	$\langle 5,11\rangle$	$\langle 6,11\rangle$	$\langle 7,11\rangle$	$\langle 8\mid 11\rangle\ \sharp$	$\sharp$	après
$\sharp$	$\sharp\langle 0,12\rangle$	$\langle 1,12\rangle$	$\langle 2,12\rangle$	$\langle 3,12\rangle$	$\langle 4,12\rangle$	$\langle 5,12\rangle$	$\langle 6,12\rangle$	$\langle 7,12\rangle$	$\langle 8\mid 12\rangle\ \sharp$	$\sharp$	après

FIG. 5.2 – *Gain d'un pas de calcul sur un calculateur par la méthode de la redondance : cas d'une configuration initiale de longueur impaire.*

L'automate simulant est le produit d'un synchroniseur en temps minimal (décrit dans [MR92]) et de la machine dont nous indiquons ci-après les règles de transition.

Remarque

Puisque le processus démarre à partir des deux extrémités de la ligne, l'ensemble de la construction est symétrique ; nous n'indiquons que la version gauche-droite des règles.

Ensemble des états

$$\mathcal{Q}' = \mathcal{Q}^3 \cup \{\bullet\} \times \mathcal{Q}^2 \cup \mathcal{Q}^2 \times \{\bullet\} \cup \mathcal{Q} \ .$$

Constitution des paires d'états

$$\begin{aligned}\delta'((a,b,\bullet),c,d) &= \delta'((a,b,\bullet),(b,c,\bullet),d)\\ &= \delta'((a,b,\bullet),(b,c,\bullet),(c,d,\bullet))\\ &= (\delta(a,b,c),\delta(b,c,d),\bullet).\end{aligned}$$

Constitution des triplets d'états

$$\begin{aligned}\delta'((a,b,\bullet),(b,c,\bullet),(\bullet,d,e)) &= \delta'((a,b,\bullet),c,(\bullet,d,e))\\ &= \delta'((a,b,\bullet),(b,c,\bullet),(c,d,e))\\ &= \delta'((a,b,\bullet),(b,c,d),(c,d,e))\\ &= \delta'((a,b,c),(b,c,d),(c,d,e))\\ &= (\delta(a,b,c),\delta(b,c,d),\delta(c,d,e)).\end{aligned}$$

Étape de Synchronisation

$$\delta'((a,b,c),(b,c,d),(c,d,e)) = \delta(\delta(a,b,c),\delta(b,c,d),\delta(c,d,e)) \ .$$

Cette méthode peut s'étendre trivialement au cas d'une accélération de k pas en regroupant $k+1$ puis $2k+1$ états dans chaque cellule.

Proposition 5.3 (ACC, deuxième forme)

Il existe une preuve du théorème 5.1 nécessitant un accroissement de s à $\mathcal{O}(s^{2k+1})$ du nombre d'états.

L'interprétation des principales règles de simulation en terme de regroupement est donnée dans le diagramme de la figure 5.3. Les règles sont présentées dans l'ordre « chronologique » (ordre de mise en oeuvre dans le diagramme espace-temps), celles de la première colonne ont un analogue par symétrie puisque l'ensemble de la solution est symétrique.

[DETAILLER]

3 Diagonale double pour les calculateurs finis

Nous modifions maintenant la méthode de la diagonale double décrite au chapitre précédent de manière à restituer au temps $|u|$ une configuration non regroupée.

La figure 5.4 donne un exemple de fonctionnement de la méthode. Là encore, le comportement de l'automate cellulaire varie en fonction de la parité de la ligne (sans pour autant que celle-ci soit connue au départ).

Ensemble des états

$$\mathcal{Q}' = \{+,\bullet\} \times \mathcal{Q}^2 \cup \mathcal{Q}^2 \times \{+,\bullet\} \cup \mathcal{Q} \times \{+\} \times \mathcal{Q} \cup \mathcal{Q}^2 \cup \mathcal{Q} \cup \{\$\} \ .$$

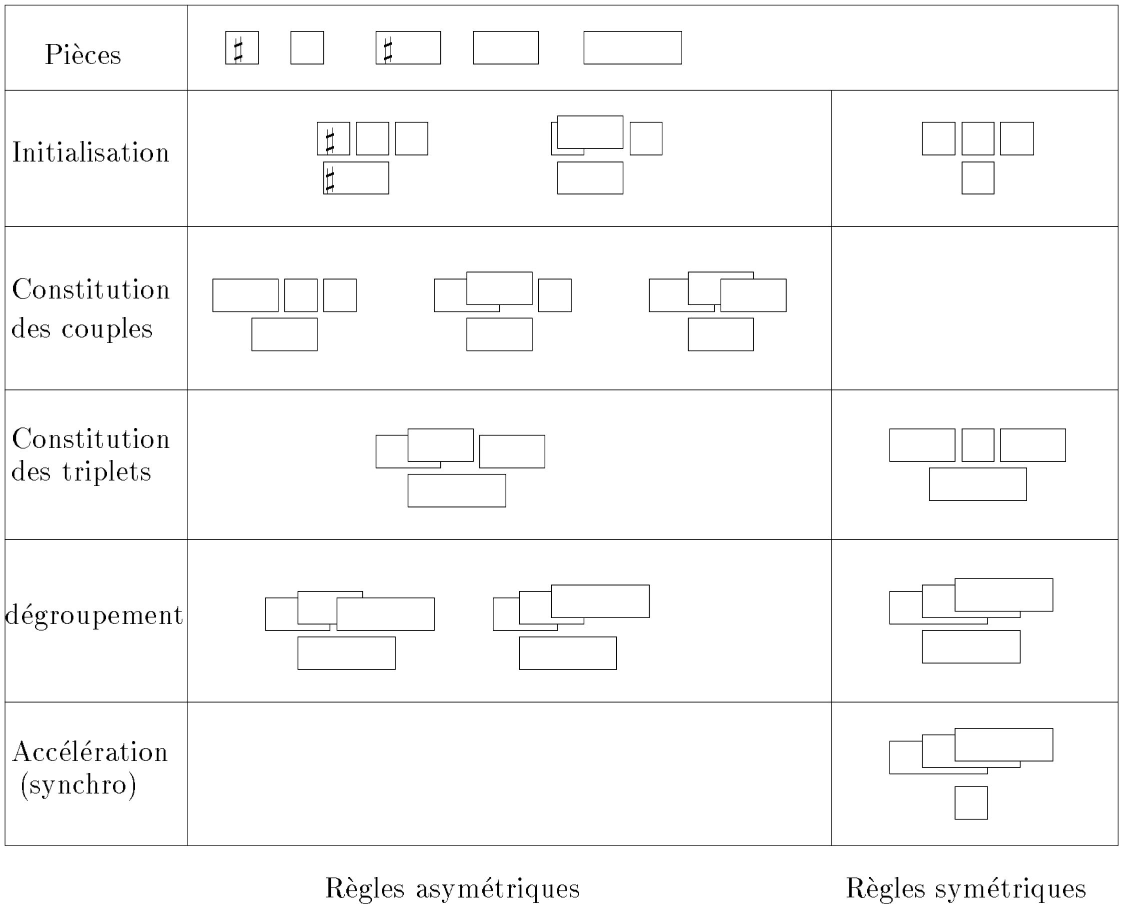


FIG. 5.3 – *Interpretation en terme de regroupement des principales règles de la méthode par redondance.*

Premiers pas

$\delta'(\sharp, a, b) = (\bullet, \sharp, \delta(\sharp, a, b))$,
$\delta'(\sharp, (\bullet, a, \sharp), b) = (\sharp, \delta(\sharp, a, b))$,
$\delta'((\bullet, \sharp, a), b, c) = (\bullet, \delta(\sharp, a, b), \delta(a, b, c))$,
$\delta'(\sharp a, \bullet(a, b), c) = (\delta(\sharp, a, b), \delta(a, b, c))$,
$\delta'(\sharp, (\sharp, a), (\bullet, a, b)) = (\sharp, \sharp)$,
$\delta'(\sharp, (\sharp, \sharp), a) = \sharp$,
$\delta'(\sharp, \$, a) = \$$.

Diagonale (avant le temps $\lceil n/2 \rceil$)

$\delta'((\bullet, a, b), c, d) = (\bullet, \delta(a, b, c), \delta(b, c, d))$,
$\delta'((a, b), (\bullet, b, c), d) = (\delta(a, b, c), \delta(b, c, d))$,
$\delta'((a, b), (c, d), (\bullet, d, e)) = (\delta(b, c, d), \delta(a, \delta(b, c, d), \delta(c, d, e)))$,
$\delta'(a, c, (d, e)) = \delta(a, b, \delta(c, d, e))$,
$\delta'(a, b, (c, d)) = \delta(a, b, c)$.

« **Collision** » **(au temps $\lceil n/2 \rceil$)**

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\langle 5,0\rangle$	$\langle 6,0\rangle$	$\langle 7,0\rangle$	$\sharp$
$\sharp$	$\bullet\ \sharp\ \langle 0,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\langle 5,1\rangle$	$\langle 6,1\rangle$	$\langle 7,1\rangle\sharp\ \bullet$	$\sharp$
$\sharp$	$\sharp\ \langle 0,2\rangle$	$\bullet\ \langle 0,2\rangle\langle 1,2\rangle$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$	$\langle 5,2\rangle$	$\langle 6,2\rangle\langle 7,2\rangle\bullet$	$\langle 7,2\rangle\sharp$	$\sharp$
$\sharp$	$\sharp\ \sharp$	$\langle 0,3\rangle\langle 1,3\rangle$	$\bullet\ \langle 1,3\rangle\langle 2,3\rangle$	$\langle 3,3\rangle$	$\langle 4,3\rangle$	$\langle 5,3\rangle\langle 6,3\rangle\bullet$	$\langle 6,3\rangle\langle 7,3\rangle$	$\sharp\ \sharp$	$\sharp$
$\sharp$	$\$$	$\langle 0,5\rangle\ \langle 0,4\rangle$	$\langle 1,4\rangle\langle 2,4\rangle$	$\bullet\ \langle 2,4\rangle\langle 3,4\rangle$	$\langle 4,4\rangle\langle 5,4\rangle\bullet$	$\langle 5,4\rangle\langle 6,4\rangle$	$\langle 7,4\rangle\ \langle 7,5\rangle$	$\$$	$\sharp$
$\sharp$	$\$$	$\langle 0,6\rangle$	$\langle 1,6\rangle\ \langle 1,5\rangle$	$+\ \langle 2,5\rangle\langle 3,5\rangle$	$\langle 4,5\rangle\langle 5,5\rangle\ +$	$\langle 6,5\rangle\ \langle 6,6\rangle$	$\langle 7,6\rangle$	$\$$	$\sharp$
$\sharp$	$\$$	$\langle 0,7\rangle$	$+\ \langle 1,7\rangle\langle 2,6\rangle$	$\langle 2,7\rangle\langle 3,7\rangle$	$\langle 4,7\rangle\langle 5,7\rangle$	$\langle 5,6\rangle\langle 6,7\rangle\ +$	$\langle 7,7\rangle$	$\$$	$\sharp$
$\sharp$	$\$$	$+\ \langle 0,8\rangle\langle 1,7\rangle$	$\langle 1,8\rangle\langle 2,8\rangle$	$\langle 3,8\rangle$	$\langle 4,8\rangle$	$\langle 5,8\rangle\langle 6,8\rangle$	$\langle 6,7\rangle\langle 7,8\rangle\ +$	$\$$	$\sharp$
$\sharp$	$+\ \sharp\ \langle 0,8\rangle$	$\langle 0,9\rangle\langle 1,9\rangle$	$\langle 2,9\rangle$	$\langle 3,9\rangle$	$\langle 4,9\rangle$	$\langle 5,9\rangle$	$\langle 6,9\rangle\langle 7,9\rangle$	$\langle 7,8\rangle\sharp\ +$	$\sharp$
$\sharp$	$+\ \sharp\ \langle 0,10\rangle$	$\langle 1,10\rangle$	$\langle 2,10\rangle$	$\langle 3,10\rangle$	$\langle 4,10\rangle$	$\langle 5,10\rangle$	$\langle 6,10\rangle$	$\langle 710\rangle,\ \sharp\ +$	$\sharp$

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\langle 5,0\rangle$	$\langle 6,0\rangle$	$\langle 7,0\rangle$	$\langle 8,0\rangle$	$\sharp$
$\sharp$	$\bullet\ \sharp\ \langle 0,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\langle 5,1\rangle$	$\langle 6,1\rangle$	$\langle 7,1\rangle$	$\langle 8,1\rangle\sharp\ \bullet$	$\sharp$
$\sharp$	$\sharp\ \langle 0,2\rangle$	$\bullet\ \langle 0,2\rangle\langle 1,2\rangle$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$	$\langle 5,2\rangle$	$\langle 6,2\rangle$	$\langle 7,2\rangle\langle 8,2\rangle\bullet$	$\langle 8,2\rangle\sharp$	$\sharp$
$\sharp$	$\sharp\ \sharp$	$\langle 0,3\rangle\langle 1,3\rangle$	$\bullet\ \langle 1,3\rangle\langle 2,3\rangle$	$\langle 3,3\rangle$	$\langle 4,3\rangle$	$\langle 5,3\rangle$	$\langle 6,3\rangle\langle 7,3\rangle\bullet$	$\langle 7,3\rangle\langle 8,3\rangle$	$\sharp\ \sharp$	$\sharp$
$\sharp$	$\$$	$\langle 0,5\rangle\langle 0,4\rangle$	$\langle 1,4\rangle\langle 2,4\rangle$	$\bullet\ \langle 2,4\rangle\langle 3,4\rangle$	$\langle 4,4\rangle$	$\langle 5,4\rangle\langle 6,4\rangle\bullet$	$\langle 6,4\rangle\langle 7,4\rangle$	$\langle 8,4\rangle\langle 8,5\rangle$	$\$$	$\sharp$
$\sharp$	$\$$	$\langle 0,6\rangle$	$\langle 1,6\rangle\langle 1,5\rangle$	$\langle 2,5\rangle\langle 3,5\rangle$	$\langle 4,5\rangle\ +\ \langle 4,6\rangle$	$\langle 5,5\rangle\langle 6,5\rangle$	$\langle 7,5\rangle\langle 7,6\rangle$	$\langle 8,6\rangle$	$\$$	$\sharp$
$\sharp$	$\$$	$\langle 0,7\rangle$	$\langle 1,7\rangle$	$+\ \langle 2,7\rangle\langle 3,7\rangle$	$\langle 4,7\rangle$	$\langle 5,7\rangle\langle 6,7\rangle\ +$	$\langle 7,7\rangle$	$\langle 8,7\rangle$	$\$$	$\sharp$
$\sharp$	$\$$	$\langle 0,8\rangle$	$+\ \langle 1,8\rangle\langle 2,7\rangle$	$\langle 2,8\rangle\langle 3,8\rangle$	$\langle 4,8\rangle$	$\langle 5,8\rangle\langle 6,8\rangle$	$\langle 6,7\rangle\langle 7,8\rangle\ +$	$\langle 8,8\rangle$	$\$$	$\sharp$
$\sharp$	$\$$	$+\ \langle 0,9\rangle\langle 1,8\rangle$	$\langle 1,9\rangle\langle 2,9\rangle$	$\langle 3,9\rangle$	$\langle 4,9\rangle$	$\langle 5,9\rangle$	$\langle 6,9\rangle\langle 7,9\rangle$	$\langle 7,8\rangle\langle 8,9\rangle\ +$	$\$$	$\sharp$
$\sharp$	$+\ \sharp\ \langle 0,9\rangle$	$\langle 0,10\rangle\langle 1,10\rangle$	$\langle 2,10\rangle$	$\langle 3,10\rangle$	$\langle 4,10\rangle$	$\langle 5,10\rangle$	$\langle 6,10\rangle$	$\langle 7,10\rangle\langle 8,10\rangle$	$\langle 8,9\rangle\sharp\ +$	$\sharp$
$\sharp$	$+\ \sharp\ \langle 0,11\rangle$	$\langle 1,11\rangle$	$\langle 2,11\rangle$	$\langle 3,11\rangle$	$\langle 4,11\rangle$	$\langle 5,11\rangle$	$\langle 6,11\rangle$	$\langle 7,11\rangle$	$\langle 8,11\rangle\sharp\ +$	$\sharp$

FIG. 5.4 – *Méthode de la diagonale appliquée aux calculateurs : cas pair en haut, impair en bas.*

Si n est pair

$$\delta'((a,b),(\bullet,c,d),(d,e,\bullet)) = (+,\delta(a,b,c),\delta(b,c,d))$$
$$\delta'((a,b),(+,c,d),(e,f,+)) = (\delta(a,\delta(b,c,d),\delta(c,d,e)),\delta(\delta(b,c,d),\delta(c,d,e),\delta(d,e,f)))\ .$$

Si n est impair

$$\delta'((\bullet,a,b),c,(d,e,\bullet)) = (\delta(b,c,d),(+,\delta(\delta(a,b,c),\delta(b,c,d),\delta(c,d,e))))\ ,$$
$$\delta'((a,b),(c,+,d),(e,f)) = \delta(\delta(a,b,c),d,\delta(c,e,f))\ ,$$
$$\delta'((a,b),(c,d),(e,+,f)) = (+,\delta(a,\delta(b,c,d),\delta(c,d,e)),\delta(\delta(b,c,d),\delta(c,d,e),f))\ ,$$
$$\delta'(+(a,b),c,(d,e,+)) = \delta(b,c,d)\ .$$

Diagonale (après le temps $\lceil n/2 \rceil$)

$$\delta'(a,b,(+,c,d)) = (+,\delta(a,b,c),c)\ ,$$
$$\delta'(a,(+,b,c),(d,e)) = (\delta(a,b,c),\delta(b,d,e))\ ,$$
$$\delta'((+,a,b),(c,d),e) = \delta(c,d,e)\ ,$$
$$\delta'((a,b),c,d) = \delta(b,c,d)\ .$$

Derniers pas

$$\delta'(\sharp,\$,(+,a,b)) = (+,\sharp,a)\ ,$$
$$\delta'(\sharp,(+,\sharp,a),(b,c)) = (+,\sharp,\delta(\sharp,b,c))\ ,$$
$$\delta'(\sharp,(+,\sharp,a),b) = (+,\delta(\sharp,a,b))\ .$$

La figure 5.5 donne l'interprétation en termes de regroupement de la déformation du diagramme espace-temps initial (pour le cas pair).

Remarques

– lors de la définition des automates cellulaires regroupeurs au chapitre 3 nous n'avons pas interdit les cas de recouvrements : un site de la machine initiale géré par plusieurs sites

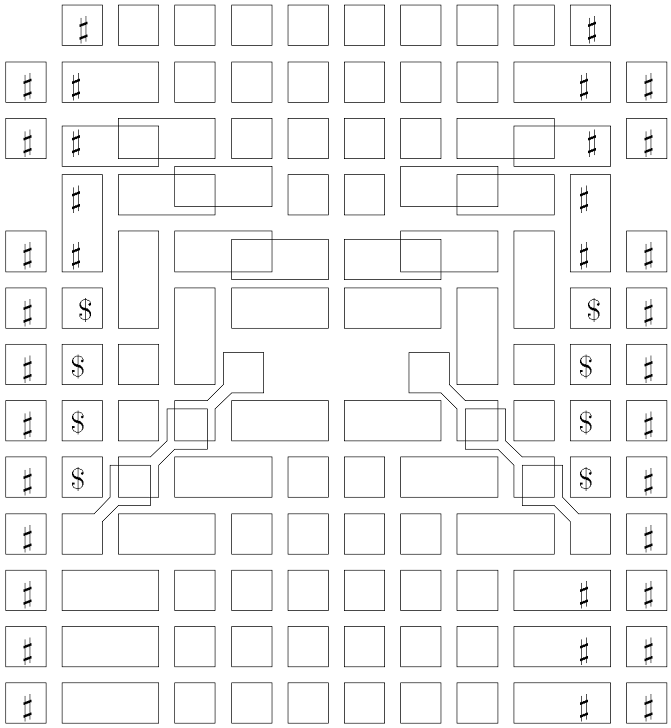


FIG. 5.5 – *Plongement du regroupeur dans le plan discret.*

du regroupeurs. C'est le cas ici, *cf.* diagramme espace-temps ci avant. Ce recouvrement permet de restaurer une configuration « normale » du calculateur simulant (c'est-à-dire identique à renommage près à la configuration du calculateur simulé) sans recourir à des triplets d'états;

- puisque l'ensemble du processus est symétrique chacune des règles possède une image miroir;
- aucune synchronisation n'est requise;
- la diagonale est distribuée sur trois cellules à cause du cas des lignes de longueur impaire où, sinon, il aurait fallu un triplet à la rencontre des deux diagonales;
- la méthode s'étend à l'accélération de k pas grâce aux techniques de généralisation de la méthode de la diagonale double décrites au chapitre précédent.

Proposition 5.4 (ACC, troisième forme)
Il existe une preuve du théorème 5.1 nécessitant un accroissement de s à $\mathcal{O}(s^{k+1})$ du nombre d'états, sans synchronisation et avec une densité linéaire.

Conclusion

Nous disposons maintenant d'un moyen d'accélérer les calculateurs d'un nombre constant de pas au prix d'un accroissement polynomial des ressources et avec une densité linéaire.

Peut on faire mieux? L'intuition et quelques arguments simples de la théorie de l'information suggèrent qu'il est « plus facile » d'accélérer d'une constante que d'un facteur. Nous continuons donc à chercher un théorème d'accélération d'une constante pour calculateur, *en version forte*, qui ne nécessite pas un accroissement polynomial des ressources.

Chapitre 6

Accélération linéaire des reconnaisseurs

Résumé

Nous proposons une nouvelle méthode d'accélération linéaire pour les reconnaisseurs qui ne fait pas appel à la synchronisation (comme c'est déjà le cas dans [IKM85]) ni à une machine séquentielle intermédiaire (comme c'est le cas dans [MR92]). Comme « effet de bord » le nombre d'états nécessaires à l'accélération tombe à $s + 2.s^k$.

La méthode de simulation est prouvée deux fois : dans le formalisme usuel tout d'abord (pour l'essentiel, cette première partie est la traduction d'un article en cours de relecture [Hee95b]) puis à l'aide d'un regroupeur.

Introduction

Énonçons tout d'abord la nouvelle version du théorème :

Théorème 6.1 (accélération linéaire pour reconnaisseurs)
Soit k un entier positif et u un mot d'entrée. Tout reconnaisseur à s états, travaillant en temps $|u| + 1 + t(u)$ sur l'entrée u, peut être simulé par un reconnaisseur à $s + 2.s^k$ états travaillant en temps $|u| + 1 + \lceil t(u)/k \rceil$ sur la même entrée.

En section 1 nous rappelons brièvement les différentes étapes dans l'élaboration de ce résultat et l'évolution des techniques de preuve entre 1969 et 1992. Notre nouvelle simulation utilise certaines de ces techniques comme composants.

La preuve elle même est en section 2 : nous illustrons tout d'abord la méthode sur un exemple simple avant de détailler le cas général.

Nous concluons sur des remarques concernant l'accélératrion des reconnaisseurs infinis et la généralisation aux dimensions supérieures.

1 Historique

Le premier résultat concernant l'accélération linéaire des reconnaisseurs est obtenu par Cole dans un papier de 1969 [Col69]. Il prouve une version faible du théorème : chaque cellule simulante code l'information de k cellules initiales (on dit que la configuration est k regroupée) ; la vitesse apparente des communications est donc k fois plus grande. Si l'automate initial possède s états le nouvel automate ainsi construit en possède s^k (ce qui est considéré comme optimal).

Toutes les preuves directes (c'est-à-dire n'utilisant que des automates cellulaires) connues à ce jour utilisent cette technique de regroupement comme composant.

La solution de Cole présente néanmoins un défaut puisque la machine accélérée ne reconnaît pas exactement le même langage que la machine initiale mais plutôt sa version k-regroupée.

En 1972, A. R. Smith [Smi72] remédie à ce problème grâce à l'astuce suivante : le reconnaisseur simulant regroupe *lui même* le mot d'entrée avant d'effectuer le calcul accéléré de Cole. Bien entendu une synchronisation en temps $|u|$ est nécessaire pour assurer une transition uniforme de la phase de regroupement à celle de calcul. Ceci conduit au résultat suivant : tout langage accepté en temps $|u| + 1 + t(u)$ par un reconnaisseur à s états est accepté en temps $|u|+1+\lceil(|u|+1+t(u))/k\rceil$ par un reconnaisseur à $s+5.s^k$ états (le facteur 5 provient de la synchronisation). C'est plus satisfaisant que le résultat de Cole, mais insuffisant dans le cas des langages reconnus très rapidement (c'est-à-dire quand la fonction $t(u)$ croît très lentement, par exemple $t(u) = |u| + 1$).

En 1985, Ibarra, Kim et Moran [IKM85] atteignent le temps de simulation optimal ($|u| + 1 + \lceil t(u)/k\rceil$) *via* un type particulier de machine de Turing (dite à balayage), équivalent en temps aux reconnaisseurs. Aucune synchronisation n'est requise mais la preuve est compliquée par l'intervention d'un modèle séquentiel. Le nombre d'états, quant à lui, passe de s à $\mathcal{O}(s^{12.k})$.

La première preuve directe est celle de J. Mazoyer et N. Reimen en 1992 dans [MR92]. On peut la voir comme une amélioration de la méthode de A. R. Smith. L'automate simulant est capable d'assurer le regroupement et le calcul à vitesse normale *en parallèle* ; ainsi, il ne perd plus de temps. Ceci conduit directement au temps de simulation optimal. Le nombre d'états passe de s à $s+5.s^k$ (une synchronisation est nécessaire). Cette solution est « purement automate cellulaire » et s'étend de façon très naturelle au cas grands voisinages.

2 Accélération linéaire sans synchronisation

Nous présentons notre nouvelle méthode comme une amélioration de celle de J. Mazoyer et N. Reimen dans [MR92] dont nous rappelons les deux principales étapes : à partir du temps 0 jusqu'au temps $|u| + 1$, le calcul est simulé à vitesse normale (1) en parallèle avec un processus de regroupement (démarré depuis l'extremité droite de la ligne). À la fin de cette étape, l'automate simulant contient une version k-regroupée de la configuration normalement obtenue au temps $|u|+1$. Dès le temps $|u|+1$ et jusqu'à la fin du calcul, c'est le calcul accéléré de Cole qui est mis en œuvre. La configuration reste k-regroupée et le résultat est disponible sur la cellule de gauche.

Un point essentiel dans cette construction est la transition uniforme de la première étape à la seconde pour toutes les cellules de numéros 1 à $\lceil |u|/k\rceil$. Pour cela, une synchronisation en temps minimal à partir des deux extrémités de la ligne est nécessaire (voir [Bal67a, Maz87, MR92]).

Remarques conduisant à notre méthode :

1. en un certain sens, la configuration partiellement k-regroupée construite au cours de

la première étape n'est pas utilisée de façon optimale. En effet, on pourrait utiliser l'information déjà regroupée pour commencer à calculer plus vite (au moins localement) ;

2. c'est la cellule d'indice 1 qui contient le résultat, elle doit donc être accélérée au plus vite (c'est-à-dire dès les temps $|u| + 1$). Les autres cellules, en revanche, peuvent être accélérées plus tard, tant que cela ne perturbe pas le calcul de la cellule 1. Ainsi, il serait imaginable de se passer de la synchronisation.

La première remarque va dans le sens d'une utilisation maximale par les cellules de l'information disponible dans leur voisinage. La seconde remarque s'avère utile pour restaurer un état stable (c'est-à-dire uniformément accéléré) à la fin du processus de regroupement/accélération.

Avant de décrire la méthode en détail, nous illustrons l'idée générale sur un fragument de diagramme espace-temps (avec $k = 2$, $|u| = 7$ et $t \in \{0, \ldots, 14\}$).

$\sharp$	$\langle 0,0\rangle$	$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\langle 5,0\rangle$	$\langle 6,0\rangle$	$\sharp$
$\sharp$	$\langle 0,1\rangle$	$\langle 1,1\rangle$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\langle 5,1\rangle$	$\langle 6,1\rangle$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,2\rangle$	$\langle 1,2\rangle$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$	$\langle 5,2\rangle$ $\langle 6,2\rangle$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,3\rangle$	$\langle 1,3\rangle$	$\langle 2,3\rangle$	$\langle 3,3\rangle$	$\langle 4,3\rangle$ $\langle 5,3\rangle$	$\langle 6,3\rangle$ $\langle 6,4\rangle$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,4\rangle$	$\langle 1,4\rangle$	$\langle 2,4\rangle$	$\langle 3,4\rangle$ $\langle 4,4\rangle$	$\langle 5,4\rangle$ $\langle 5,5\rangle$	$\langle 6,5\rangle$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,5\rangle$	$\langle 1,5\rangle$	$\langle 2,5\rangle$ $\langle 3,5\rangle$	$\langle 4,5\rangle$ $\langle 4,6\rangle$	$\langle 5,6\rangle$ $\langle 6,6\rangle$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,6\rangle$	$\langle 1,6\rangle$ $\langle 2,6\rangle$	$\langle 3,6\rangle$ $\langle 3,7\rangle$	$\langle 4,7\rangle$ $\langle 5,7\rangle$	$\langle 6,7\rangle$ $\langle 6,8\rangle$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,7\rangle$ $\langle 1,7\rangle$	$\langle 2,7\rangle$ $\langle 2,8\rangle$	$\langle 3,8\rangle$ $\langle 4,8\rangle$	$\langle 5,8\rangle$ $\langle 5,9\rangle$	$\langle 6,9\rangle$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,9\rangle$ $\langle 1,9\rangle$	$\langle 2,9\rangle$ $\langle 3,9\rangle$	$\langle 4,9\rangle$ $\langle 4,10\rangle$	$\langle 5,10\rangle$ $\langle 6,10\rangle$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,11\rangle$ $\langle 1,11\rangle$	$\langle 2,11\rangle$ $\langle 3,11\rangle$	$\langle 4,11\rangle$ $\langle 5,11\rangle$	$\langle 6,11\rangle$ $\langle 6,12\rangle$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,13\rangle$ $\langle 1,13\rangle$	$\langle 2,13\rangle$ $\langle 3,13\rangle$	$\langle 4,13\rangle$ $\langle 5,13\rangle$	$\langle 6,13\rangle$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,15\rangle$ $\langle 1,15\rangle$	$\langle 2,15\rangle$ $\langle 3,15\rangle$	$\langle 4,15\rangle$ $\langle 5,15\rangle$	$\langle 6,15\rangle$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,17\rangle$ $\langle 1,17\rangle$	$\langle 2,17\rangle$ $\langle 3,17\rangle$	$\langle 4,17\rangle$ $\langle 5,17\rangle$	$\langle 6,17\rangle$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,19\rangle$ $\langle 1,19\rangle$	$\langle 2,19\rangle$ $\langle 3,19\rangle$	$\langle 4,19\rangle$ $\langle 5,19\rangle$	$\langle 6,19\rangle$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$
$\sharp$	$\langle 0,21\rangle$ $\langle 1,21\rangle$	$\langle 2,21\rangle$ $\langle 3,21\rangle$	$\langle 4,21\rangle$ $\langle 5,21\rangle$	$\langle 6,21\rangle$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$ $\sharp$	$\sharp$

FIG. 6.1 – *Accélération d'un facteur deux sur un reconnaisseur.*

Rappelons que la notation $\langle i, t\rangle$ désigne l'état de la cellule numéro i au temps t lors de l'évolution du reconnaisseur initial. L'exemple simple ci-avant met en évidence les différences entre cette méthode de simulation et les méthodes plus « classiques » basées sur l'utilisation d'une synchronisation [MR92, Smi71b].

La figure 6.2 montre une approximation continue de la géométrie du processus. Une approximation continue peut être vue comme la limite du diagramme espace-temps quand $|u|$ tend vers l'inifini. Les déplacements d'informations sont alors représentés par des droites de pentes rationnelles.

Comme nous l'indiquions plus haut, la méthode fonctionne essentiellement en simulant « au plus tôt » le comportement des cellules initiales. Il y a donc une accélération précoce

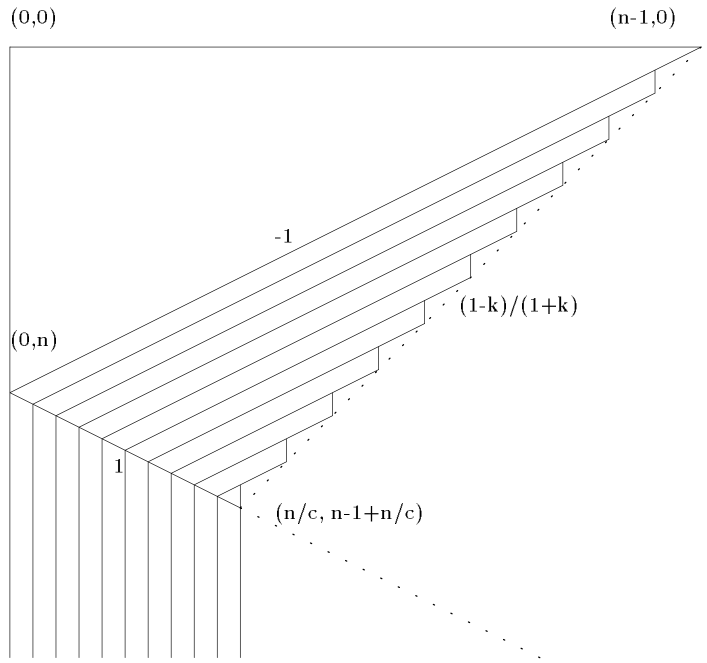


FIG. 6.2 – *Géométrie du nouveau procédé d'accélération linéaire.*

dans certaines parties du diagramme espace-temps, alors que d'autres parties calculent encore lentement (à vitesse 1). La figure 6.3 indique de quelle façon les lignes de temps sont « tordues » au cours du processus. Elle permet également de constater que cette simulation est un regroupement au sens du chapitre 3.

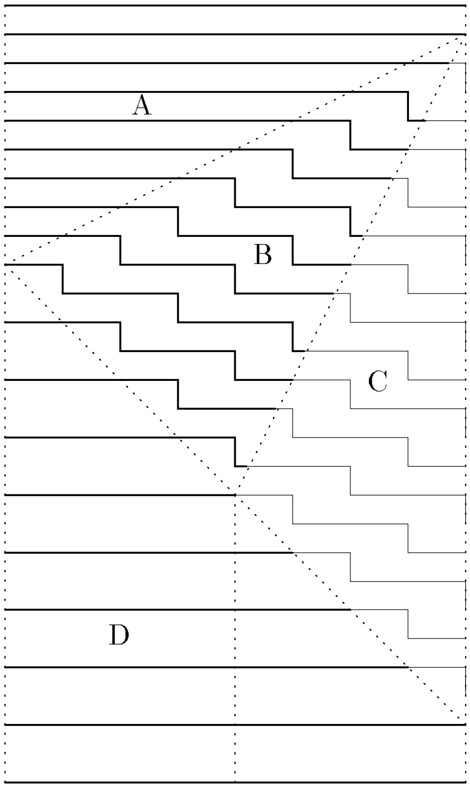
Il est temps de décrire formellement la méthode employée et notamment le procédé de regroupement/accélération : une cellule simulante contient soit un seul état (cellule simple) soit un k-uplet d'états (cellule composée). Il est assez naturel, et en fait très utile, de supposer qu'une fois regroupée, l'information d'une cellule ne sera jamais dégroupée. En d'autres termes, une cellule composée ne redevient pas une cellule simple. Sachant que le processus de regroupement démarre de l'extremité droite de la ligne, les contextes qui peuvent exister autour d'une cellule sont alors les suivants :

- une cellule simple entourée de deux cellules simples ne peut rein faire d'autre que calculer son nouvel état en simulant un pas de la fonction de transition initiale (f) ;

- une cellule simple entourée d'une cellule simple à gauche et d'une cellule composée à droite peut utiliser le surcroît d'information venu de droite pour calculer l'état de k cellules initiales ; elle devient alors elle même une cellule composée ;

- enfin, une cellule composée ayant pour voisine de droite une cellule composée également utilise toute l'information disponible pour calculer k pas dans le futur.

La figure 6.4 montre comment les cellules réagissent à leur contexte ainsi que le déroulement du processus d'accélération/regroupement. En haut de cette figure on observe le dia-



A : calcul à vitesse normale
B : accélération « dès que possible »
C : produits d'états quiescents seulement
D : calcul accéléré

FIG. 6.3 – *Distorsion des lignes de temps dans le nouveau processus d'accélération.*

gramme espace-temps de l'automate simulant au voisinage des cellules A, B, et C. En bas on voit ce que les cellules A, B, et C simulent.

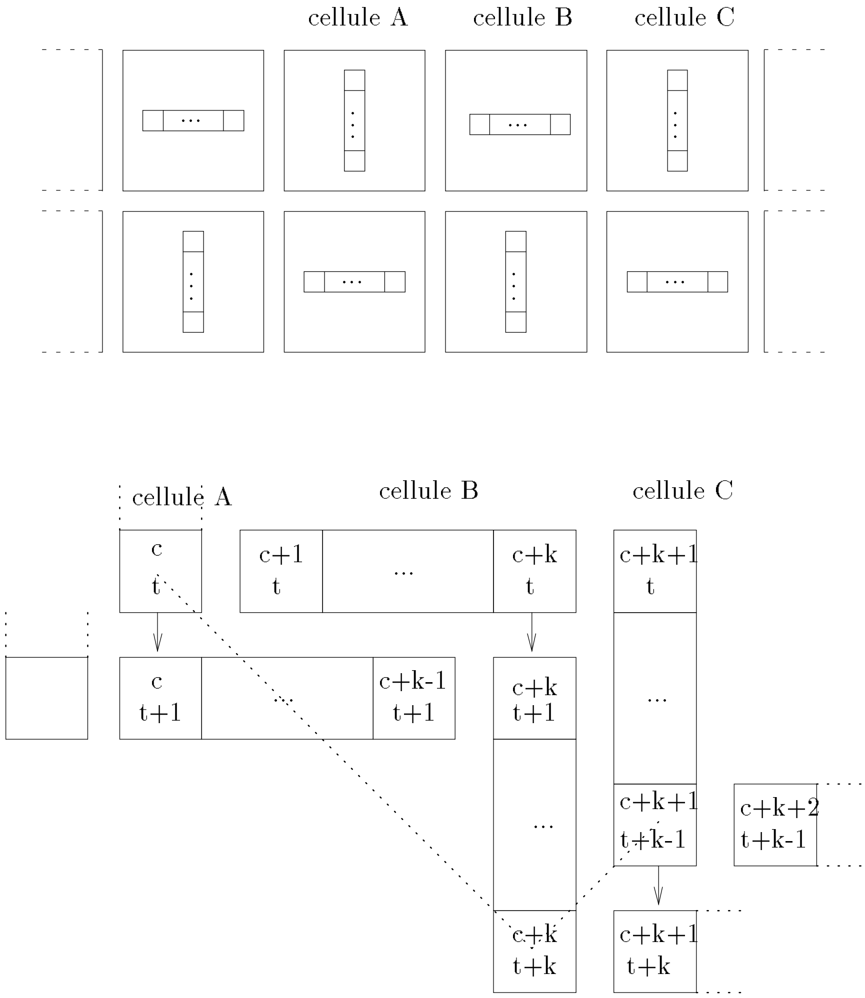
Notons que l'avancement dans le temps des cellules simulées n'est pas partout le même : les cellules de droite (B et C) sont plus accélérées que la cellule de gauche (A). Une contrainte essentielle est que cette accélération non uniforme n'interfère pas avec la simulation de la fonction de transition initiale δ.

Remarquons également qu'une cellule doit pouvoir faire la différence entre un k-uplet de valeurs uniformément accélérées, tel que $\langle i,t\rangle \cdots \langle i+k-1,t\rangle$ (un tel k-uplet est dit horizontal) et un k-uplet de de valeurs de plus en plus accélérées tel que $\langle i,t\rangle \cdots \langle i,t+k-1\rangle$ (dit vertical). En conséquence, l'ensemble des états simulants doit être $\mathcal{Q}' = \mathcal{Q}^k \cup \overline{\mathcal{Q}^k} \cup \mathcal{Q}$. S'il y a s états dans l'automate initial, il y en a $2.s^k + s$ dans l'automate simulant.

Notation

La terminologie que nous utilisons dans la preuve ci dessous est celle des mots : ε désigne le mot vide et « $\cdot$ » le produit de concaténation. L'ensemble $\mathcal{Q}^*$ est celui des mots finis sur l'alphabet $\mathcal{Q}$. Le k-uplet d'états $(x_1, x_2, \ldots, x_k)$ est représenté par le mot $x_1 \cdot x_2 \cdots \cdot x_k$ si c'est un k-uplet horizontal et par le mot $\overline{x_1 \cdot x_2 \cdots \cdot x_k}$ si c'est un k-uplet vertical.

Nous utilisons également certaines notations introduites au chapitre 2 que nous rappelons brièvement : une lettre minuscule est une variable d'état ; les lettres $a, b, c \ldots$ prennent des valeurs non frontières, les lettres $\ldots x, y, z$ prennent des valeurs quelconques. Les suites d'états sont notées xs et ys.



Les valeurs c et t dans les cellules désignent $\langle c, t\rangle$

FIG. 6.4 – *Phase d'accélération/regroupement. Relativement à la machine simulante (en haut) et à la machine simulée (en bas).*

Rappelons enfin la notation usuelle pour l'itération de la composition : soit g une fonction et i un entier positif, g^{i+1} est la composition de $g(g^i)$ et g^0 est l'identité.

Nous pouvons maintenant prouver le théorème 6.1 en o construisant explicitement la fonction de simulation δ'.

Preuve

Tout d'abord, et pour clarifier l'expression de δ' nous introduisons deux fonctions auxiliaires importantes : g et h. Toutes deux dépendent de la fonction initiale δ. Intuitivement, g est la restriction de la fonction de transition globale aux configurations finies, h est la clôture de g par composition.

Remarquons qu'ici la fonction h se comporte un peu différemment selon que la lettre la plus à gauche de son entrée vaut $\sharp$ ou non.

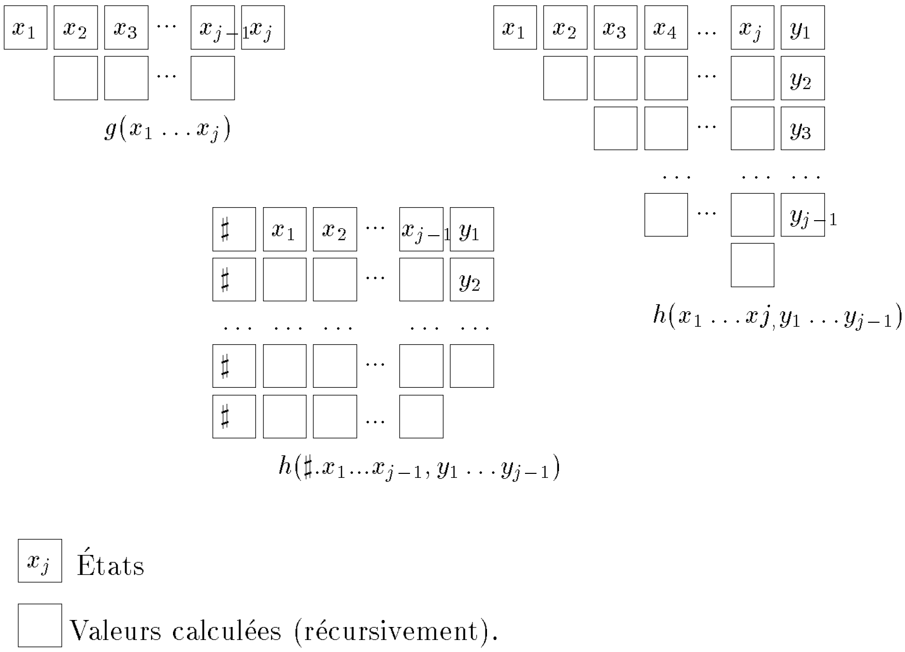


FIG. 6.5 – *Fonctionnement des fonctions auxiliaires g et h.*

$$
\begin{gathered}
g : \mathcal{Q}^* \to \mathcal{Q}^* \\
g(\varepsilon) = g(x) = g(x_1 \cdot x_2) = \varepsilon \\
g(x_1 \cdot x_2 \cdot x_3 \cdot xs) = \delta(x_1, x_2, x_3) \cdot g(x_2 \cdot x_3 \cdot xs)
\end{gathered}
$$

$$
\begin{gathered}
h : \mathcal{Q}^* \times \mathcal{Q}^* \to \mathcal{Q}^* \\
h(xs, \varepsilon) = g(xs),\ h(\varepsilon, ys) = \varepsilon \\
h(a \cdot xs, y \cdot ys) = h(g(a \cdot xs \cdot y), ys) \\
h(\sharp \cdot xs, y \cdot ys) = h(\sharp \cdot g(\sharp \cdot xs \cdot y), ys)
\end{gathered}
$$

Grâce à δ, g et h nous donnons une expression de la fonction $\delta' : \mathcal{Q}' \to \mathcal{Q}'$, où $\mathcal{Q}' = \mathcal{Q}^k \cup \overline{\mathcal{Q}^k} \cup \mathcal{Q}$.

`Début` $\delta'(x, a, \sharp) = \delta(x, a, \sharp) \cdot \sharp^{k-1}$ (ne sert qu'une fois) , `Norm` $\delta'(x,a,b) = \delta(x,a,b)$,

`Droite1` $\delta'(a, x \cdot \sharp^{k-1}, \sharp) = \overline{\sharp^k}$,

`Droite2` $\delta'(a \cdot x_1 \cdots x_{k-1}, \overline{\sharp^k}, \sharp) = \sharp^k$,

`Droite3` $\delta'(\overline{x_1 \cdots x_k}, \sharp^k, \sharp) = \overline{\sharp^k}$,

`Groupe1` $\delta'(a, x_0, x_1 \cdots x_k) = g(a \cdot x_0 \cdots x_k)$,

`Groupe2` $\delta'(x_1 \cdots x_k, \overline{y_1 \cdots y_k}, z_1 \cdots z_k) = g(h(x_1 \cdots x_k, y_1 \cdots y_{k-1}) \cdot y_k \cdot z_1 \cdots z_k)$,

`Axl1` $\delta'(a, x_1 \cdots x_k, \overline{y_1 \cdots y_k}) = \overline{h(x_{k-1} \cdot x_k \cdot y_1) \cdots h(a \cdot x_1 \cdots x_k, y_1 \cdots y_k)}$,

`Axl2` $\delta'(\overline{x_1 \cdots x_k}, y_1 \cdots y_k, \overline{z_1 \cdots z_k}) = \delta'(x_k, y_1 \cdots y_k, \overline{z_1 \cdots z_k})$
(voir `Axl1` si $x_1 \neq \sharp$ et `Rebond` sinon) ,

`Rebond` $\delta'(\sharp, x_1 \cdots x_k, \overline{y_1 \cdots y_k}) = h(\sharp \cdot x_1 \cdots x_k, y_1 \cdots y_k)$ (ne sert qu'une fois) ,

`Restaure` $\delta'(x_1 \cdots x_k, y_1 \cdots y_k, \overline{z_1 \cdots z_k}) = g^k(x_1 \cdots x_k \cdot y_1 \cdots y_k \cdot z_1 \cdots z_k)$,

`Cole1` $\delta'(\sharp, x_1 \cdots x_k, y_1 \cdots y_k) = g^k(\sharp^k \cdot x_1 \cdots x_k \cdot y_1 \cdots y_k)$,

`Cole2` $\delta'(x_1 \cdots x_k, y_1 \cdots y_k, z_1 \cdots z_k) = g^k(x_1 \cdots x_k \cdot y_1 \cdots y_k \cdot z_1 \cdots z_k)$,

`Fin` Lorsqu'un état terminal τ apparaît sur la cellule zéro au cours du calcul d'une règle `Rebond` ou `Cole1`, le calcul s'arrêteet le résultat est τ.

Notons que τ peut apparaître pendant l'un des appels récursifs nécessaires au calcul de `Rebond` ou `Cole1`, pas nécessairement le dernier.

□

3 Améliorations

La technique ci-dessus s'adapte aisément au cas des reconnaisseurs semi-infinis, conduisant au résultat suivant :

Théorème 6.2 (accélération linéaire pour reconnaisseurs semi-infinis)
Tout reconnaisseur semi-infini à s états travaillant en temps $|u|+1+t(u)$ peut être simulé pas un reconnaisseur semi-infini à $s+2.s^k$ états travaillant en temps $|u|+1+\lceil t(u)/k \rceil$.

Preuve
Les règles `Droite1`, `Droite2` et `Droite3` ne sont plus nécessaires puisque la partie droite du diagramme n'est plus une frontière. Les règles `Début`, `Groupe` et `Axl` peuvent être prolongées en considérant que l'état $\sharp$ est équivalent aux regroupements d'états $\sharp^k$ et $\overline{\sharp^k}$. Remarquons que la règle `Restaure` est perpétuellement utilisée dans la partie droite du diagramme qui ne cesse de s'étendre.

`Début2` $\delta'(a, \sharp, \sharp) = h(a \cdot \sharp^k, \sharp^k)$ (ne sert qu'une fois) ,

`Groupe3` $\delta'(x_1 \cdots x_k, \overline{y_1 \cdots y_k}, \sharp) = \delta'(x_1 \cdots x_k, \overline{y_1 \cdots y_k}, \sharp^k)$

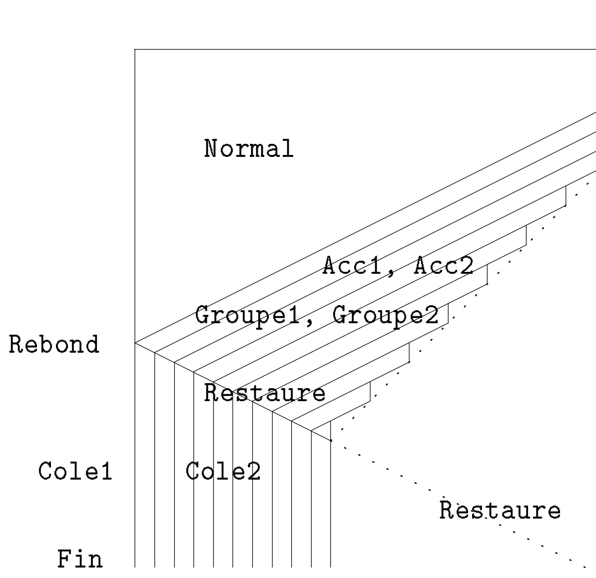


FIG. 6.6 – *Règles de simulation : quand et où sont elles utilisées ?*

`Ax13` $\delta'(\overline{x_1 \cdots x_k}, \sharp, \sharp) = \delta'(\overline{x_1 \cdots x_k}, \sharp^k, \overline{\sharp^k})$

□

Le cas bi-infini est traité par simulation sur un automate semi-infini (grâce au lemme 2.2, chapitre 2). Bien entendu cela provoque une augmentation quadratique du nombre d'états.

4 Preuve par la méthode de regroupement

La simulation utilisée est en fait un pur regroupement tel qu'il est décrit au chapitre 3. Dans ce formalisme, la preuve suit les étapes suivantes :

1. écriture des 3-agencements, de la fonction de transition δ opérant sur les triplets de pièces et de la fonction de placement θ ;
2. vérification de la faisabilité pour chacune des règles de transition ;
3. vérification de la cohérence faible et de la propriété d'accélération.

La figure 6.9 résume à elle seule l'ensemble des deux premières étapes on y observe :

- toutes les pièces utilisées,
- les 3-agencement de pièces positionnées les unes par rapport aux autres (à l'aide des fonction ω_1 et ω_2) ;
- la fonction de transition δ pour chaque 3-agencement ;
- la fonction de placement θ pour chaque 3-agencement.

- la preuve de la faisabilité ; en effet, l'ensemble des conséquences de chaque 3-agencement apparaît en pointillés et il suffit de *vérifier* que la pièce calculée par δ et placée par θ n'en sort pas (en pratique, la vérification de la faisabilité est automatisable).

La cohérence faible et la propriété d'accélération font l'objet du lemme 6.3.

Enfin, le plongement représenté en figure 6.7 et le diagramme espace-temps représenté en figure 6.8 ne sont pas indispensable pour la preuve formelle de la méthode de simulation, en revanche ils sont d'une grande aide pour la compréhension.

Notation

Avant d'indiquer le lemme de correction, nous introduison les notations suivantes :

- le symbole ξ désigne n'importe quel mot sur $\{\boxed{\substack{\text{D}\\\text{D}}}, \boxed{\text{D D}}\}$, nous l'utiliserons pour désigner une partie de la configuration dont il n'est pas nécessaire de calculer la longueur ;
- nous notons γ le mot infini $(\boxed{}.\boxed{\substack{\\}})^{\infty}$ et $\gamma_{|p}$ le préfixe de longueur p de ce mot. Ainsi, $\gamma_{|3} = \boxed{}.\boxed{\substack{\\}}.\boxed{}$.

Lemme 6.3 (correction) .
Notons $c_0 = \boxed{\text{G}}.\square^n.\boxed{\text{D}}$ la configuration initiale du regroupeur. Au temps $t > 0$, l'image - plongée dans le demi-plan discret - de c_0 est de la forme :

Si t vaut 1

$$c_1 = \boxed{\text{G}}.\square^{n-1}.\boxed{\text{D}}.\boxed{\text{D}} \ \textit{avec}\ v_{0,1} = \vec{s}$$

Si t est compris entre 2 **et** n

$$\textit{Si } t = 3.q + 1,\ c_t = \boxed{\text{G}}.\square^{n-t}.\gamma_{|q}.\boxed{\text{D}}.\xi.\boxed{\text{D}} \ \textit{avec}\ v_{0,t} = t.\vec{s}$$

$$\textit{Si } t = 3.q + 2,\ c_t = \boxed{\text{G}}.\square^{n-t}.\gamma_{q+1}.\xi.\boxed{\text{D}} \ \textit{avec}\ v_{0,t} = t.\vec{s}$$

$$\textit{Si } t = 3.q,\ c_t = \boxed{\text{G}}.\square^{n-t}.\gamma_{q+2}.\xi.\boxed{\text{D}} \ \textit{avec}\ v_{0,t} = t.\vec{s}$$

Si t est compris entre $n+1$ et $\lfloor 2n/3 \rfloor$

$$c_t = \boxed{\text{G}}.\boxed{}^{t-n}.\gamma_{2n-\lfloor 4t/3 \rfloor}.\xi.\boxed{\text{D}} \ \textit{avec}\ v_{0,t} = (n + 2.t).\vec{s}$$

Si t est supérieur à $\lfloor 2n/3 \rfloor$

$$c_t = \boxed{\text{G}}.\boxed{}^{\lceil n/2 \rceil}.\xi.\boxed{\text{D}} \ \textit{avec}\ v_{0,t} = (n + 2.t).\vec{s}$$

Rappelons que le mot ξ n'a aucune influence sur la simulation puisqu'il se projette dans une zone perpétuellement quiescente, c'est la raison pour laquelle nous ne calculons pas sa longueur.

Rappelons également les grandes lignes de la méthode de preuve : pour une configuration c_t annoncée, on vérifie qu'elle est atteinte au moins une fois (cas de base) puis on recense les règles applicables afin de calculer $\Delta(c_t)$, il reste à vérifier que $\Delta(c_t)$ vaut bien c_{t+1} (étape de récurrence). Dans certains cas, cette récurrence est très simple et nous ne la détaillons pas.

Preuve

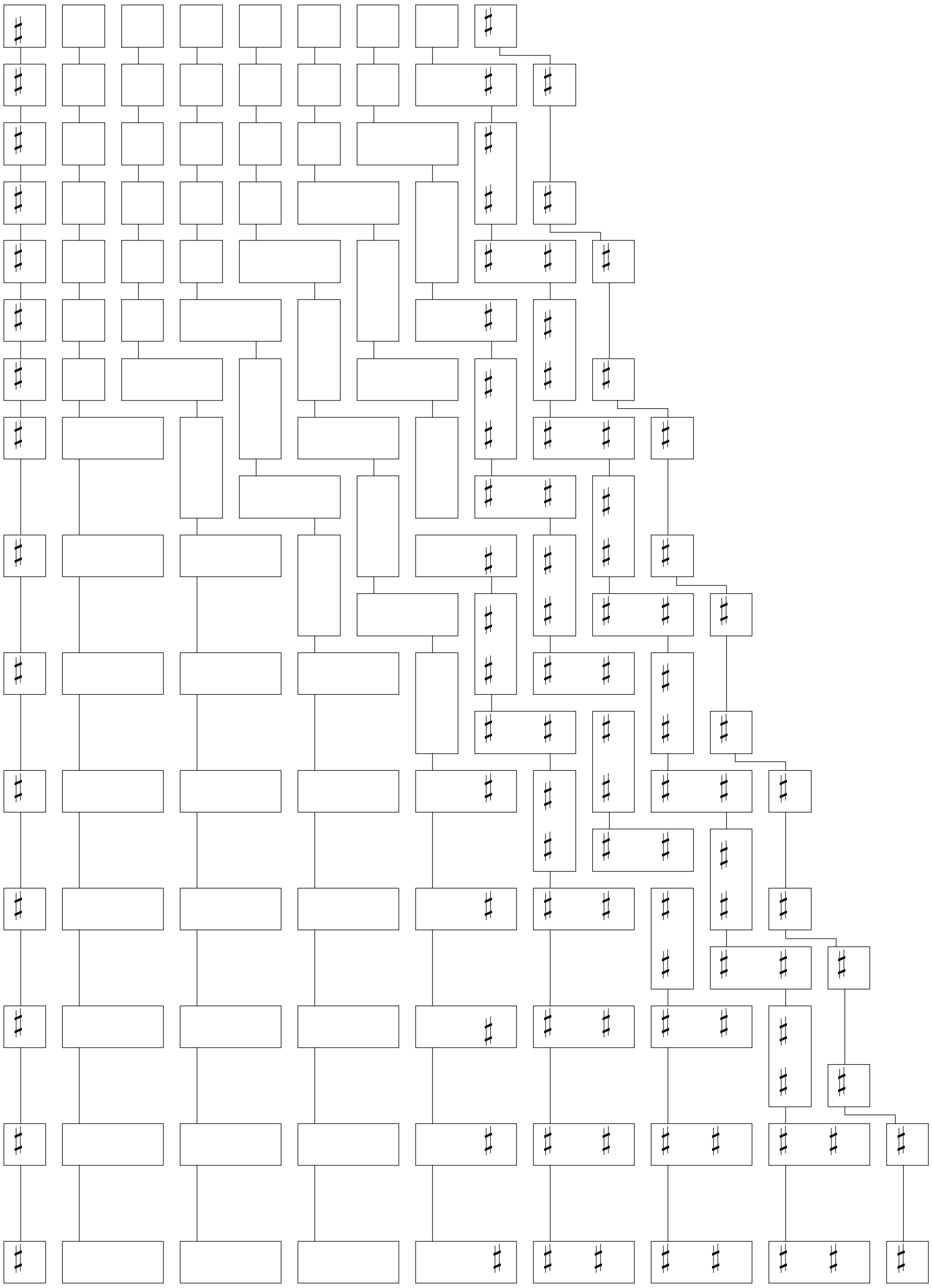


FIG. 6.7 – *Accélération linéaire des reconnaisseurs : plongement dans le demi-plan discret.*

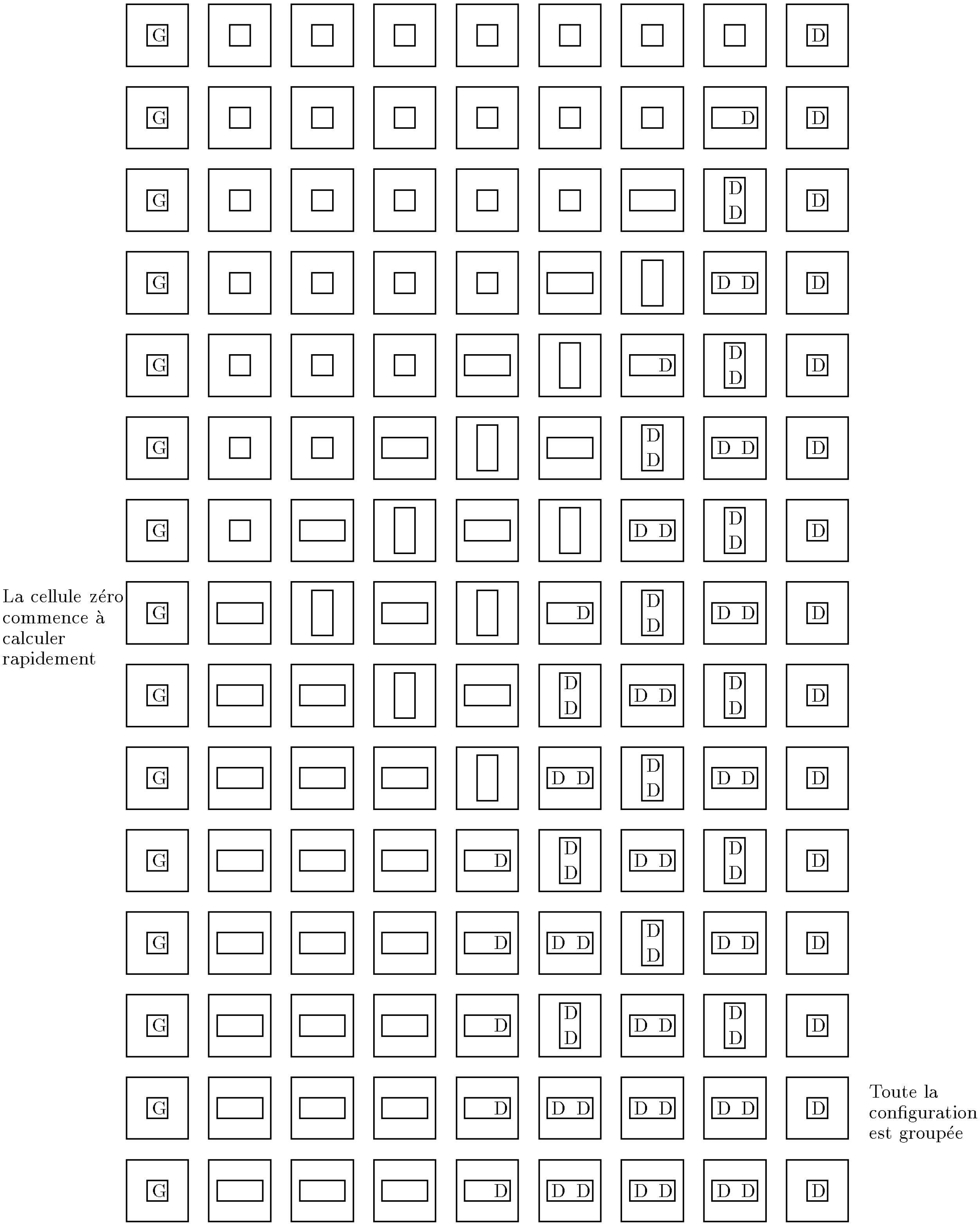


FIG. 6.8 – *Accélération linéaire des reconnaisseurs : fonctionnement du regroupeur.*

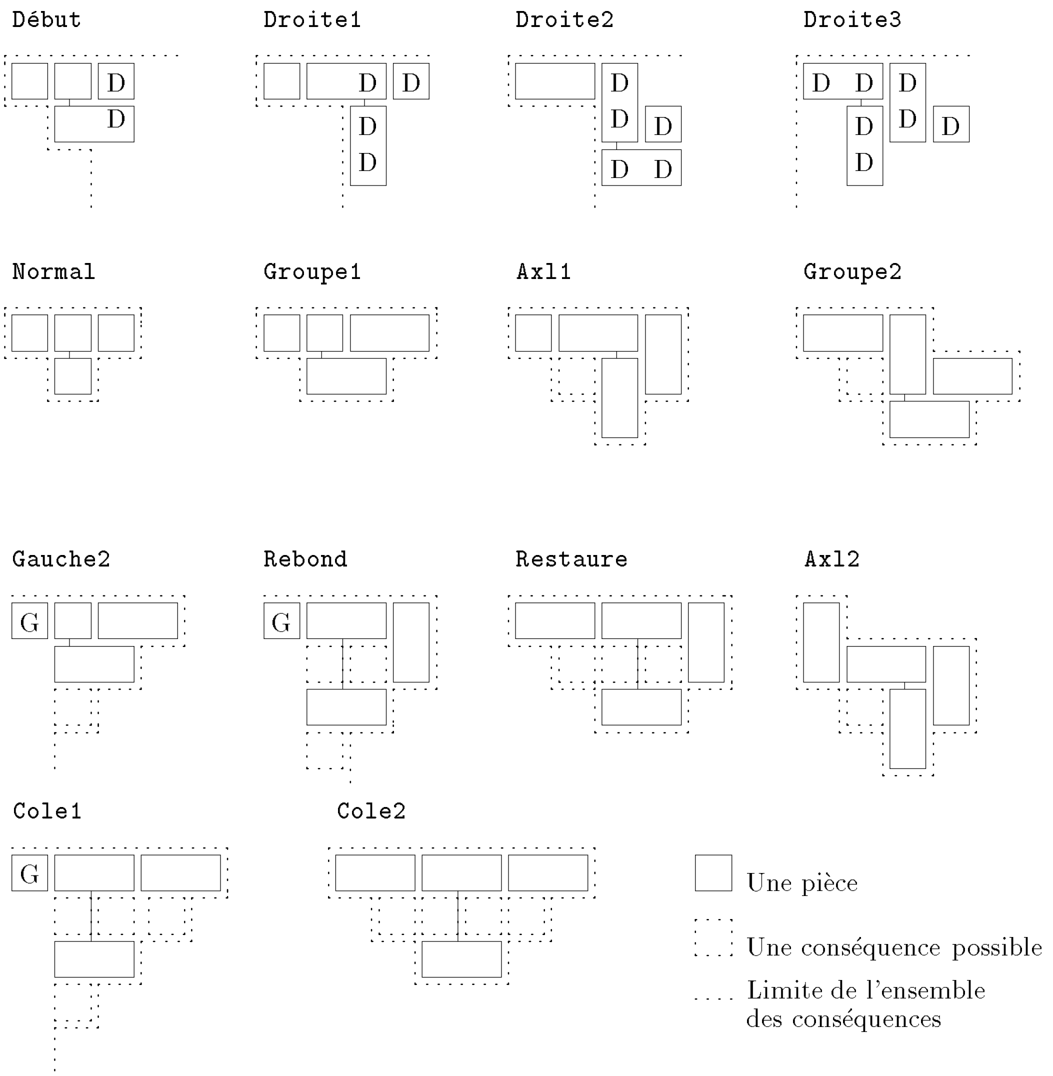


FIG. 6.9 – *Accélération linéaire des reconnaisseurs: paramètres $\mathcal{Q}$, F, δ et θ du regroupeur.*

Si t vaut 1

La configuration annoncée dans le lemme est atteinte grâce aux seules règles `Normal` et `Début`.

Si t est compris entre 2 **et** n

Tout d'abord, remarquons que la configuration au temps 2 a bien la forme générale pour $t = 3.q+1$ (avec $q = 0$ et ξ réduit au mot vide). Pour une tell configuration, faisons opérer une première fois la fonction de transition ; les règles applicables sont `Normal`, `Groupe1,2`, `Axl1,2` et des règles opérant sur ξ que nous ne précisons pas. On aboutit alors à la configuration suivante :

$$c_t = \boxed{\text{G}}.\boxed{}^{n-t}.\gamma_{q+1}.\xi.\boxed{\text{D}}$$

qui a bien la forme annoncée pour $t = 3.q + 2$.

Le même travail peut être effectué pour le passage d'une configuration type $3.q + 2$ à une configuration type $3.q'$.

À partir du temps n

Jusqu'à maintenant, toutes les configurations étaient placées de proche en proche (voir propriétés de rigidité au chapitre 3) à partir de la cellule 0 positionnée dans le demi-plan par le vecteur $v_{0,t} = t.\vec{s}$.

Dès le temps $n + 1$, la situation change car les règles `Rebond` puis `Cole1` deviennent utilisables (sur la cellule 0 donc). Pour ces deux règles, la fonction θ vaut $2.\vec{s}$ au lieu de $\vec{s}$ (ce qui traduit le fait que le calcul sur la cellule 0 est deux fois plus rapide).

La preuve de cette partie du lemme est simple et se fait également par recensement des règles applicables. Le seul point à expliciter est la longueur du préfixe de γ dans la configuration :

$$c_t = \boxed{\text{G}}.\boxed{}^{t-n}.\gamma_{2n-\lfloor 4t/3\rfloor}.\xi.\boxed{\text{D}}$$

Il est assez facile de voir que la règle `Restaure` incrémente la longueur du mot $\boxed{}^{t-n}$ à chaque application. Par ailleurs, les règles `Groupe1,2` et `Axl1,2` continuent d'opérer dans la partie droite du diagramme espace-temps, y compris sur les portions quiescentes, et font augmenter la longueur du mot ξ à raison d'une cellule toutes les trois unités de temps. Au total, le préfixe du mot γ passe d'une longueur initiale de $\lfloor 2n/3\rfloor$ cellules quand t vaut n à une longueur nulle dès que t vaut $\lfloor 3n/2\rfloor$, d'où la forme générale annoncée dans le lemme.

Pour n supérieur à $\lfloor 2n/3\rfloor$

La configuration est « stabilisée » et uniformément regroupée. En effet, seules les règles `Cole1,2` sont applicables et elles transforme la pièce $\boxed{}$ en elle même. La calcul se poursuit à vitesse 2 (car $\theta(x, \boxed{}, \boxed{}) = 2.sud$).

récapitulatif

Tout au cours de la preuve (et lors d'une vérification par programme) nous n'avons rencontré aucune pièce non recenssée, aucun 3-agencement non recenssé. Ceci prouve donc la cohérence faible. Par ailleurs, l'examen des valeurs de $v_{0,t}$ a montré que la propriété d'accélération est vraie dès le temps n pour la cellule 0.

□

Chapitre 7

Accélération linéaire des synchroniseurs

Résumé

Les synchroniseurs (déjà utilisés comme outils dans les chapitres précédents) sont des calculateurs particuliers puisqu'ils n'utilisent qu'un seul symbole de sortie. Pour cette sous classe, nous obtenons un théorème d'accélération linéaire complet (c'est-à-dire fonctionnant pour tous les synchroniseurs à partir du temps n) en utilisant des méthodes proches de celles de J. Mazoyer et N. Reimen dans [MR92]. Par ailleurs, il est connu que tout polynôme à coefficients entiers positifs et plus grand que l'identité est un temps de synchronisation [Maz90]. Une application directe de l'accélération linéaire et de l'accélération d'une constante pour les synchroniseurs est l'extension de ce résultat au polynômes à coefficients rationnels quelconques (non nécessairement positifs) et plus grands que l'identité. Plus généralement, l'ensemble des temps de synchronisation est clos par composition avec de tels polynômes.
Note : pour l'essentiel, ce chapitre est la version française de l'article [Hee96b].

1 Accélération linéaire pour synchroniseurs

Cette section est consacrée à la preuve du théorème 7.1. Ses conséquences sur l'ensemble des temps de synchronisation sont couvertes en section 2.

Théorème 7.1 (accélération linéaire pour synchroniseurs)
Si k un entier positif et $n+t(n)$ un temps de synchronisation. Alors $n+\lceil t(n)/k\rceil$ est aussi un temps de synchronisation.

Avant de détailler la preuve, nous émettons quelques remarques à propos des temps de synchronisation (en particulier nous précisons certaines remarques du chapitre 2).

Tout d'abord, notons que la définition 2.5 des synchroniseurs diffère quelque peu des définitions classiques trouvées dans la littérature (voir [Bal67a, Maz87, Yun93]). En effet, ici il n'y a pas explicitement de « général » chargé de démarrer la synchronisation. Celui-ci est créé par les règles de transitions elles-mêmes : $\delta_0(\$,\sharp,\sharp)=\delta_0(\sharp,\sharp,\$)=$ "général". Il faut donc un pas de plus pour terminer une synchronisation en temps minimal, soit n étapes au lieu de $n-1$ (où n est le nombre de cellules à synchroniser).

Cette caractéristique autorise une expression plus directe de la composition d'automates cellulaires. En effet, le résultat fourni par une synchronisation - une ligne de cellules partageant toutes le même état - peut être envoyé tel quel en entrée d'une nouvelle synchronisation.

Mis à part cette modification, la fonction de transition est celle décrite par J. Mazoyer et N. Reimen [MR92], avec un état supplémentaire pour éviter l'émergence de configurations du type $\$\sharp\sharp\ldots$ ou $\ldots\sharp\sharp\$$ une fois que la synchronisation est démarrée.

Observons par ailleurs qu'au sens du lemme 2.1, une synchronisation en moins de n étapes est impossible. Tout d'abord il est facile de voir qu'il n'est pas possible de synchroniser en temps inférieur à $\lfloor n/2 \rfloor$: en effet, jusqu'au temps $\lfloor n/2 \rfloor$, chaque configuration contient au moins une cellule quiescente. D'autre part, supposons qu'il existe un entier n_0 et un temps de synchronisation $t(n_0)$ inférieur à n_0. Alors, toutes les cellules d'indice supérieur à n_0 sont trop loin pour avoir une influence quelconque sur le processus de synchronisation (voir figure 7.1), et donc la cellule de gauche atteint l'état **feu** au temps $t(n_0)$, pour tout $n > t(n_0)$. Ce qui revient à dire : $\forall n > t(n_0)$, $t(n) = t(n_0)$. C'est bien entendu absurde dès que n dépasse la valeur $2.t(n_0)$.

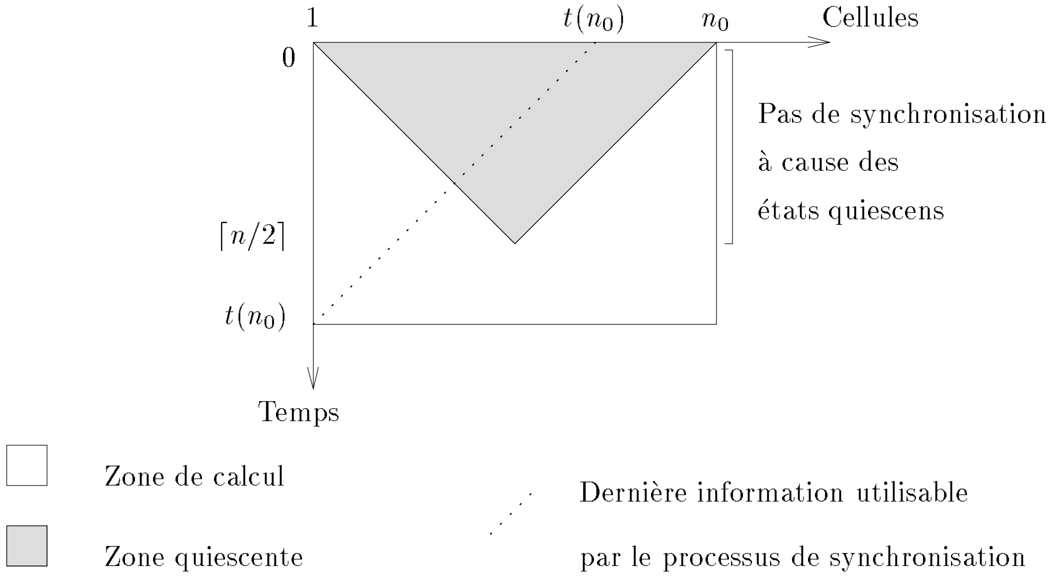


FIG. 7.1 – *La synchronisation en temps sous-linéaire est impossible.*

La preuve du théorème 7.1 nécessite plusieurs étapes qui font l'objet des paragraphes suivants.

1.1 Comment désigner des cellules

Lemme 7.2 (signal gauche-droite)

Pour tout entiers positifs a et b tels que $b \geq a$, il existe un automate cellulaire $\mathcal{L}$ à b états non quiescents pour lequel, à partir de la configuration infinie $\$\sharp^\omega$, l'ensemble des sites non quiescents du diagramme espace-temps est $\{(\lceil ta/b \rceil, t), t \geq 1\}$.

Preuve

L'ensemble des états d'un tel automate est $\mathcal{Q} = \{\sharp, 1, \ldots, b\}$, et sa fonction de transition est $f_{\mathcal{L}}$ décrite ainsi :

`Début :` $f_{\mathcal{L}}(\$, \sharp, \sharp) = 1$

`Attends :` si $\lceil at/b \rceil = \lceil (t+1)a/b \rceil$ alors
$f_{\mathcal{L}}(t, \sharp, \sharp) = \sharp$ et $f_{\mathcal{L}}(\$, t, \sharp) = f_{\mathcal{L}}(\sharp, t, \sharp) = t+1$

`Déplace :` si $\lceil at/b \rceil \neq \lceil (t+1)a/b \rceil$ alors
$f_{\mathcal{L}}(t, \sharp, \sharp) = t+1$ et $f_{\mathcal{L}}(\$, t, \sharp) = f_{\mathcal{L}}(\sharp, t, \sharp) = \sharp$

`Recommence :` $f_{\mathcal{L}}(\$, b, \sharp) = f_{\mathcal{L}}(\sharp, b, \sharp) = 1$ et $f_{\mathcal{L}}(b, \sharp, \sharp) = \sharp$

Une simple récurrence montre qu'à tout temps t, la cellule d'indice $\lceil ta/b \rceil$ est la seule non quiescente. Elle a pour état $1 + (t - 1 \bmod b)$ (*cf.* partie gauche de la figure 7.2).

$\square$

Plus loin, nous devons également particulariser des cellules à partir de l'extrémité droite de la ligne :

Lemme 7.3 (signal droite-gauche)

Pour tout entiers positifs n, c et d tels que $d \geq c$, il existe un automate cellulaire $\mathcal{R}$ à d états non quiescents pour lequel, à partir de la configuration $\$\sharp^{\$}$, l'ensemble des sites non quiescents du diagramme espace-temps est $\{(n - \lfloor tc/d \rfloor, t), 0 \leq \lfloor tc/d \rfloor \leq n-1\}$

Preuve

« L'approximation continue » du problème consiste à trouver deux nombres rationnels α et β tels que l'équation $i = \alpha t + \beta$ soit vraie à la fois pour le site $(n, 0)$ et pour le site $(n - \lfloor ct/d \rfloor, t)$. Les valeurs correctes des coefficients sont $\alpha = -c/d$ et $\beta = n$, ce qui conduit à l'équation discrète $i = \lfloor -ct/d \rfloor + n$ ou, sous une forme légèrement différente, $i = n - \lceil ct/d \rceil$. Cette dernière forme permet une programmation du signal $\mathcal{R}$ par recopie d'un motif (exactement comme dans le cas de $\mathcal{L}$). Le motif est défini par $\{(c - \lceil ct/d \rceil, t), 1 \leq t \leq d\}$ (*cf.* partie droite de la figure 7.2).

$\square$

À tout signal gauche-droite caractérisé par a et b on associe un signal droite-gauche, dit conjugué, caractérisé par $c = b - a$ et $d = b$. Tous deux atteignent la cellule $\lceil an/b \rceil$ au temps n.

Remarque

Dans le paragraphe suivant nous devons particulariser les cellules $\lceil an/b \rceil$ et $\lceil an/b \rceil + 1$. Il nous faut donc modifier l'un des deux signaux conjugués (disons le signal droite-gauche) afin qu'il atteigne la cellule $\lceil an/b \rceil + 1$ au temps n. Cela se fait facilement en introduisant un délai : le premier déplacement du signal vers la gauche (et seulement celui-ci) est annulé. Ce délai initial nécessite d états supplémentaires dans le pire des cas.

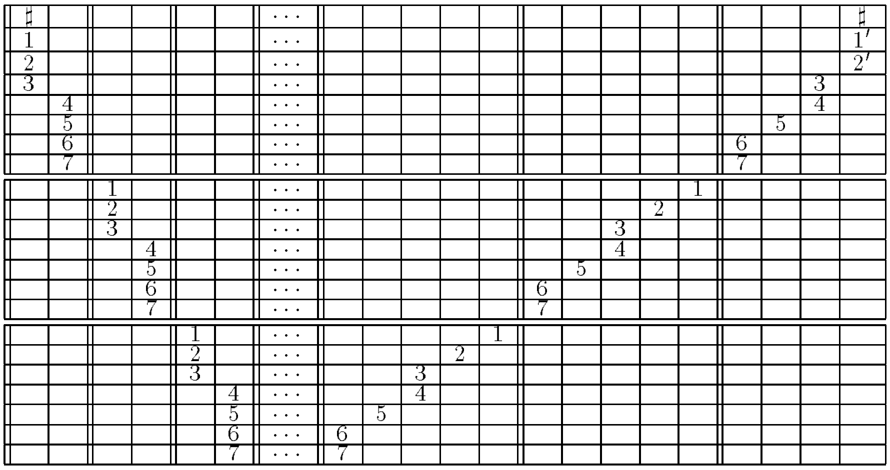


FIG. 7.2 – *Signal gauche-droite $a = 2, b = 7$ et son conjugué $c = b - a = 5, d = 7$ avec délai.*

1.2 Regroupement autour d'une cellule

Nous exposons maintenant la méthode de construction d'une configuration k-regroupée centrée autour d'une cellule désignée (*cf.* [MR92] pour plus de détails sur le k-regroupement en général).

Définition 7.1 (configurations k-groupées)
Soit une configuration $c = \$a_1 \ldots a_n\$$ sur un ensemble d'états $\mathcal{Q}$. Les configurations k-groupées de c sont les configurations $\$A_1 \ldots A_m\$$ sur $\mathcal{Q}^k$ de la forme :

$$\$.\sharp^k \ldots \sharp^k.\underbrace{(\sharp \ldots \sharp.a_1 \ldots a_p)}_{\alpha}.(a_{p+1} \ldots a_{p+k}) \ldots \underbrace{(a_{n-l} \ldots a_n, \sharp \ldots \sharp)}_{\beta}.\sharp^k \ldots \sharp^k.\$$$

Nous désignons par α (resp. β) le rang de la première (resp. dernière) cellule qui n'est pas dans l'état $\sharp^k$. On dit qui la configuration k-groupée s'étend entre α et β.

Remarquons que $\beta - \alpha$ appartient à $\{\lfloor n/k \rfloor - 1, \lfloor n/k \rfloor, \lfloor n/k \rfloor - 1\}$ et qu'il existe plusieurs configurations k-groupées pour des mêmes valeurs de α et β.

Dans le cas de l'accélération des synchroniseurs, nous avons besoin d'un *double* processus de regroupement : la partie gauche de la ligne (qui s'étend entre les cellules 1 et $\lceil na/b \rceil$) doit être regroupée le long d'un signal gauche-droite (obtenu à partir du lemme 7.2) et, symétriquement, la partie droite de la ligne (qui s'étend entre les cellules $\lceil na/b \rceil + 1$ et n) doit être regroupée le long d'un signal droite-gauche (avec délai, *cf.* lemme 7.3). De cette façon, toute la ligne peut être groupée autour des cellules $\lceil an/b \rceil$ et $\lceil an/b \rceil + 1$.

Le temps n marquant la fin de la phase de regroupement est particularisé par une synchronisation auxiliaire (à partir des deux extrémités de la ligne, *cf.* lemme 2.1).

La géométrie du processus apparaît sur la figure 7.3.

Une condition essentielle est que le calcul du synchroniseur à accélérer ne soit pas modifié par ce processus de regroupement. En d'autres termes, il faut regrouper et calculer *en parallèle*.

Puisque nous utilisons déjà une synchronisation auxiliaire en temps n, nous pouvons adapter assez facilement la technique développée par J. Mazoyer et N. Reimen dans [MR92].

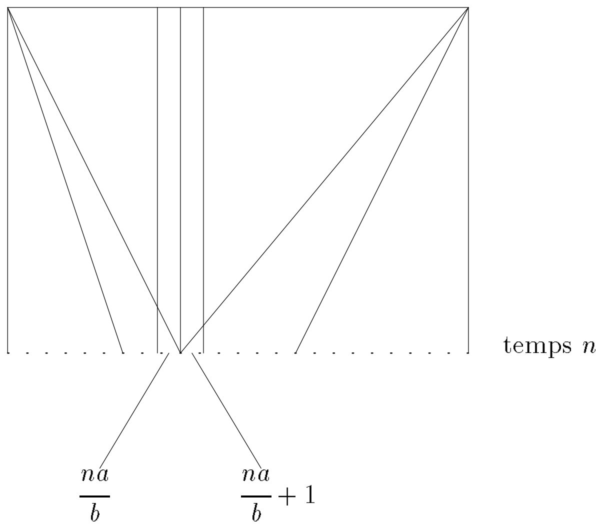


FIG. 7.3 – *Géométrie du double processus de regroupement.*

Lemme 7.4 (regroupement gauche)
Soit $\mathcal{A} = (\mathcal{Q}, f)$ un automate cellulaire, et a et b deux entiers tels que $b \geq a$. Il existe un automate cellulaire $\mathcal{B} = (\mathcal{Q} \cup \{1, \ldots, b\} \times \mathcal{Q}^k, f')$ tel que, pour toute configuration initiale c_0 et pour tout temps $t \geq 1$, la partie de la configuration de $\mathcal{B}$ qui s'étend de la cellule $1 + \lceil at/b \rceil - \lceil \lceil at/b \rceil /b \rceil$ à la cellule $\lceil at/b \rceil$ au temps t est une version k-groupée de la configuration de $\mathcal{A}$ entre la cellule 1 et la cellule $\lceil at/b \rceil$ au même temps t.

Preuve
Soit π la première projection de $\{1, \ldots . b\} \times \mathcal{Q}^k$ vers $\{1, \ldots, b\}$. Nous décrivons la fonction de transition f' de l'automate $\mathcal{B}$. La partie de f' qui génère le signal le long duquel à lieu le regroupement est $\pi(f')$. C'est simplement une modification du signal gauche-droite (*cf.* lemme 7.2) afin que toutes les cellules à gauche du signal « sachent » quand il y a déplacement vers la droite. Les règles de transition utilisées sont les suivantes :
$\pi(f')(\$, 1, \sharp) = 2$ (ou 1 si $a = b = 1$), $\pi(f')(j, \sharp, \sharp) = j + 1$, $\pi(f')(j, j, \sharp) = j + 1$, $\pi(f')(j, j, j) = j + 1$.
Le signal à été prouvé correct précédemment et il est facile de vérifier que toute cellule à gauche du signal connaît la valeur du compteur (j).

La deuxième partie de la preuve est la description du processus de regroupement (adapté de [MR92]). La valeur du compteur pour le signal est j et les lettres x, y, z (ou x_i, y_i, z_i) désignent des variables pour les états des cellules pendant le calcul de $\mathcal{A}$.

`Début` : $f'(\sharp, x, y) = (1, \sharp, \ldots, \sharp, f(\sharp, x, y))$, $f'(x, y, z) = f(x, y, z)$

`Déplace1` : quand $\lceil aj/b \rceil \neq \lceil a(j-1)/b \rceil$, $0 < j < b$,
$f'((j, x_1, \ldots, x_k), y, z) = (j + 1, f(x_2, x_3, x_4), \ldots, f(x_{k-1}, x_k, y), f(x_k, y, z))$,
$f'((j, x_1, \ldots, x_k), (j, y_1, \ldots, y_k), z) = (j + 1, f(x_{k-1}, x_k, y_1), \ldots, f(y_{k-2}, y_k, z))$,
$f'((j, x_1, \ldots, x_k), y_1, \ldots, y_k, z_1, \ldots, z_k) = (j + 1, f(x_{k-1}, x_k, y_1), \ldots, f(z_{k-2}, z_{k-1}, z_k))$.

`Déplace2:` quand $j = b$,

$$f'((b, x_1, \ldots, x_k), y, z) = (1, f(x_2, x_3, x_4), \ldots, f(x_k, y, z))$$

`Attends:` quand $\lceil aj/b \rceil = \lceil a(j-1)/b \rceil, 0 < j < b$,

$$f'((j, x_1, \ldots, x_k), y, z) = (j+1, f(x_k, y, z))$$
$$f'((j, x_1, \ldots, x_k), (j, y_1, \ldots, y_k), z) = (j+1, f(x_k, y_1, y_2), \ldots, f(y_{k-1}, y_k, z)),$$
$$f'((j, x_1, \ldots, x_k), y_1, \ldots, y_k, z_1, \ldots, z_k) = (j+1, f(x_{k-1}, x_k, y_1), \ldots, f(z_{k-2}, z_{k-1}, z_k)).$$

$\square$

$\langle 1,0\rangle$	$\langle 2,0\rangle$	$\langle 3,0\rangle$	$\langle 4,0\rangle$	$\langle 5,0\rangle$	$\langle 6,0\rangle$
$(\mathbf{1},\sharp,\langle 1,1\rangle)$	$\langle 2,1\rangle$	$\langle 3,1\rangle$	$\langle 4,1\rangle$	$\langle 5,1\rangle$	$\langle 6,1\rangle$
$(\mathbf{2},\sharp,\langle 1,2\rangle)$	$\langle 2,2\rangle$	$\langle 3,2\rangle$	$\langle 4,2\rangle$	$\langle 5,2\rangle$	$\langle 6,2\rangle$
$(\mathbf{3},\sharp,\langle 1,3\rangle)$	$\langle 2,3\rangle$	$\langle 3,3\rangle$	$\langle 4,3\rangle$	$\langle 5,3\rangle$	$\langle 6,3\rangle$
$(\mathbf{4},\sharp,\sharp)$	$(\mathbf{4},\langle 1,4\rangle,\langle 2,4\rangle)$	$\langle 3,4\rangle$	$\langle 4,4\rangle$	$\langle 5,4\rangle$	$\langle 6,4\rangle$
$(\mathbf{5},\sharp,\sharp)$	$(\mathbf{5},\langle 1,5\rangle,\langle 2,5\rangle)$	$\langle 3,5\rangle$	$\langle 4,5\rangle$	$\langle 5,5\rangle$	$\langle 6,5\rangle$
$(\mathbf{6},\sharp,\sharp)$	$(\mathbf{6},\langle 1,6\rangle,\langle 2,6\rangle)$	$\langle 3,6\rangle$	$\langle 4,6\rangle$	$\langle 5,6\rangle$	$\langle 6,6\rangle$
$(\mathbf{7},\sharp,\sharp)$	$(\mathbf{7},\langle 1,7\rangle,\langle 2,7\rangle)$	$\langle 3,7\rangle$	$\langle 4,7\rangle$	$\langle 5,7\rangle$	$\langle 6,7\rangle$
$(\mathbf{1},\sharp,\sharp)$	$(\mathbf{1},\sharp,\langle 1,8\rangle)$	$(\mathbf{1},\langle 2,8\rangle,\langle 3,8\rangle)$	$\langle 4,8\rangle$	$\langle 5,8\rangle$	$\langle 6,8\rangle$
$(\mathbf{2},\sharp,\sharp)$	$(\mathbf{2},\sharp,\langle 1,9\rangle)$	$(\mathbf{2},\langle 2,9\rangle,\langle 3,9\rangle)$	$\langle 4,9\rangle$	$\langle 5,9\rangle$	$\langle 6,9\rangle$
$(\mathbf{3},\sharp,\sharp)$	$(\mathbf{3},\sharp,\langle 1,10\rangle)$	$(\mathbf{3},\langle 2,10\rangle,\langle 3,10\rangle)$	$\langle 4,10\rangle$	$\langle 5,10\rangle$	$\langle 6,10\rangle$
$(\mathbf{4},\sharp,\sharp)$	$(\mathbf{4},\sharp,\sharp)$	$(\mathbf{4},\langle 1,11\rangle,\langle 2,11\rangle)$	$(\mathbf{4},\langle 3,11\rangle,\langle 4,11\rangle)$	$\langle 5,11\rangle$	$\langle 6,11\rangle$
$(\mathbf{5},\sharp,\sharp)$	$(\mathbf{5},\sharp,\sharp)$	$(\mathbf{5},\langle 1,12\rangle,\langle 2,12\rangle)$	$(\mathbf{5},\langle 3,12\rangle,\langle 4,12\rangle)$	$\langle 5,12\rangle$	$\langle 6,12\rangle$
$(\mathbf{6},\sharp,\sharp)$	$(\mathbf{6},\sharp,\sharp)$	$(\mathbf{6},\langle 1,13\rangle,\langle 2,13\rangle)$	$(\mathbf{6},\langle 3,13\rangle,\langle 4,13\rangle)$	$\langle 5,13\rangle$	$\langle 6,13\rangle$
$(\mathbf{7},\sharp,\sharp)$	$(\mathbf{7},\sharp,\sharp)$	$(\mathbf{7},\langle 1,14\rangle,\langle 2,14\rangle)$	$(\mathbf{7},\langle 3,14\rangle,\langle 4,14\rangle)$	$\langle 5,14\rangle$	$\langle 6,14\rangle$
$(\mathbf{1},\sharp,\sharp)$	$(\mathbf{1},\sharp,\sharp)$	$(\mathbf{1},\sharp,\langle 1,15\rangle)$	$(\mathbf{1},\langle 2,15\rangle,\langle 3,15\rangle)$	$(\mathbf{1},\langle 4,15\rangle,\langle 5,15\rangle)$	$\langle 6,15\rangle$
$(\mathbf{2},\sharp,\sharp)$	$(\mathbf{2},\sharp,\sharp)$	$(\mathbf{2},\sharp,\langle 1,16\rangle)$	$(\mathbf{2},\langle 2,16\rangle,\langle 3,16\rangle)$	$(\mathbf{2},\langle 4,15\rangle,\langle 5,15\rangle)$	$\langle 6,16\rangle$
$(\mathbf{3},\sharp,\sharp)$	$(\mathbf{3},\sharp,\sharp)$	$(\mathbf{3},\sharp,\langle 1,17\rangle)$	$(\mathbf{3},\langle 2,17\rangle,\langle 3,17\rangle)$	$(\mathbf{3},\langle 4,15\rangle,\langle 5,15\rangle)$	$\langle 6,17\rangle$
$(\mathbf{4},\sharp,\sharp)$	$(\mathbf{4},\sharp,\sharp)$	$(\mathbf{4},\sharp,\sharp)$	$(\mathbf{4},\langle 1,18\rangle,\langle 2,18\rangle)$	$(\mathbf{4},\langle 3,18\rangle,\langle 4,18\rangle)$	$(\mathbf{4},\langle 5,18\rangle,\langle 6,18\rangle)$
$(\mathbf{5},\sharp,\sharp)$	$(\mathbf{5},\sharp,\sharp)$	$(\mathbf{5},\sharp,\sharp)$	$(\mathbf{5},\langle 1,19\rangle,\langle 2,19\rangle)$	$(\mathbf{5},\langle 3,19\rangle,\langle 4,19\rangle)$	$(\mathbf{5},\langle 5,19\rangle,\langle 6,19\rangle)$
$(\mathbf{6},\sharp,\sharp)$	$(\mathbf{6},\sharp,\sharp)$	$(\mathbf{6},\sharp,\sharp)$	$(\mathbf{6},\langle 1,20\rangle,\langle 2,20\rangle)$	$(\mathbf{6},\langle 3,20\rangle,\langle 4,20\rangle)$	$(\mathbf{6},\langle 5,20\rangle,\langle 6,20\rangle)$
$(\mathbf{7},\sharp,\sharp)$	$(\mathbf{7},\sharp,\sharp)$	$(\mathbf{7},\sharp,\sharp)$	$(\mathbf{7},\langle 1,21\rangle,\langle 2,21\rangle)$	$(\mathbf{7},\langle 3,21\rangle,\langle 4,21\rangle)$	$(\mathbf{7},\langle 5,21\rangle,\langle 6,21\rangle)$

FIG. 7.4 – *Exemple de regroupement à gauche pour $k = 2, a = 2, b = 7$.*

De la même façon, le regroupement à partir de l'extrémité droite de la ligne utilise une version modifiée du signal donné par le lemme 7.3.

Lemme 7.5 (Regroupement à droite)
Soient un automate cellulaire $\mathcal{A} = (\mathcal{Q}, f)$, et deux entiers c et d tels que $d \geq c$. Il existe un automate cellulaire $\mathcal{C} = (\mathcal{Q} \cup \{1, \ldots, d\} \times \mathcal{Q}^k, f'')$ tel que, pour toute configuration initiale c_0 de longueur n et pour tout temps $1 \leq t \leq n$, la partie de la configuration de $\mathcal{C}$ s'étendant entre la cellule $n - \lfloor ct/d \rfloor$ à la cellule $n - \lceil \lfloor ct/d \rfloor /k \rceil$ est une version k-groupée de la portion de configuration de l'automate $\mathcal{A}$ qui s'étend entre les cellules $n - \lfloor ct/d \rfloor$ et n au même temps t.

La preuve est celle du lemme 7.4 à ceci près que le signal droite-gauche utilisé subit un retard au démarrage.

1.3 Calcul accéléré

À partir d'une configuration k-groupée, il est très simple d'effectuer un calcul k fois plus rapide (c'est l'idée originelle de Cole [Col69]). Ce calcul est mis en place dès que tous les regroupement sont terminés, c'est-à-dire au temps n (matérialisé par une synchronisation en temps minimal). Afin de décrire dans le détail la fonction de calcul rapide utilisée, nous avons besoins des fonction auxiliaires suivantes (le symbole . représente la concaténation et w un mot) :

$$mot : \mathcal{Q}^k \to \mathcal{Q}^*, mot(x_1, x_2, \dots, x_k) = x_1.x_2 \dots x_k$$

$$f_{mot} : \mathcal{Q}^* \to \mathcal{Q}^*$$
$$f_{mot}(\varepsilon) = f_{mot}(x) = f_{mot}(x_1.x_2) = \varepsilon$$
$$f_{mot}(x_1.x_2.x_3.w) = f(x_1, x_2, x_3).f_{mot}(x_2.x_3.w)$$

La fonction *mot* transforme les vecteurs d'états en mots et la fonction f_{mot} est l'extension naturelle de la fonction de transition initiale f aux mots. Maintenant nous pouvons donner une expression pour la fonction de calcul rapide (f_r) :
$f_r(L, M, R) = {f_{mot}}^k(mot(L).mot(M).mot(R))$ où L, M, R sont des k-uplets d'états.

1.4 Regrouper et accélérer partout

En lançant simultanément plusieurs processus de regroupement autour de certaines cellules (bien choisies) le long de la ligne, il est possible d'obtenir une configuration *uniformément regroupée*. En d'autres mots, une configuration telle que chaque cellule appartienne au moins à une des sections regroupées.
Plus précisément, il y a $K + 1$ processus de regroupement, autour des cellules d'indices $\lceil i.n/K \rceil$, $(0 \leq i \leq K)$.
Soient L_i (resp. R_i) l'indice de la cellule la plus à gauche (resp. à droite) impliquée dans le i ème processus du regroupement.

En appliquant le lemme 7.4 pour $a = in, b = K$ et le lemme 7.5 pour $n - \lfloor ct/d \rfloor = \lceil \frac{in}{K} \rceil$, on obtient :

$$L_i = 1 + \left\lceil \frac{in}{K} \right\rceil - \left\lceil \frac{\lceil \frac{in}{K} \rceil}{k} \right\rceil \quad \text{et} \quad R_i = \left\lceil \frac{in}{K} \right\rceil + \left\lceil \frac{n - \lceil \frac{in}{K} \rceil}{k} \right\rceil$$

La ligne est globalement regroupée si la relation $L_{i+1} \leq R_i + 1$ est vérifiée pour tout $0 \leq i \leq K$.
Montrons qu'il suffit de choisir $K = k$ pour satisfaire la condition :

$$L_{i+1} - R_i + 1 = \left\lceil \frac{(i+1)n}{K} \right\rceil - \left\lceil \frac{in}{K} \right\rceil - \left\lceil \frac{\lceil \frac{(i+1)n}{K} \rceil}{k} \right\rceil - \left\lceil \frac{n - \lceil \frac{in}{K} \rceil}{k} \right\rceil$$

$$L_{i+1} - R_i + 1 = \left\lceil \frac{in}{K} + \frac{n}{K} \right\rceil - \left\lceil \frac{in}{K} \right\rceil - \left\lceil \frac{in}{kK} + \frac{n}{kK} \right\rceil - \left\lceil \frac{n}{k} - \frac{\lceil \frac{in}{K} \rceil}{k} \right\rceil$$

(puisque $\lceil \lceil x/p \rceil /q \rceil = \lceil x/pq \rceil$, *cf.* [Knu73, p. 42])

$$L_{i+1} - R_i + 1 \leq \left\lceil \frac{n}{K} \right\rceil - \left\lceil \frac{n}{k} \right\rceil - \left\lceil \frac{n}{kK} \right\rceil + 1$$

(car $\lceil a + b \rceil - \lceil a \rceil \leq \lceil b \rceil$, $\lceil c + d \rceil \geq \lceil c \rceil + \lceil d \rceil - 1$, $\lceil e - f \rceil \geq \lceil e \rceil - \lceil f \rceil$)

$$\text{Ainsi, } L_{i+1} - R_i + 1 \leq 0 \text{ dès que } K \geq k \text{ puisque } \lceil n/kK \rceil \geq 1$$

Remarque

Une fois le processus terminé (c'est-à-dire au temps n), il est possible que certaines parties regroupées se chevauchent (en une ou plusieurs cellules). Qui plus est, avant le temps n, certains sites comme $(1,1)$ ou $(n,1)$ doivent contenir des informations relatives aux $k+1$ regroupements en cours. Puisque nous ne voulons pas que ces informations interfèrent, l'automate résultant doit être le produit des $k+1$ automates effectuant chacun un des regroupements. Ceci nous conduit à un automate à $\mathcal{O}(s^{k(k+1)})$ états si l'automate initial en nécessitait s. La constante multiplicative caché dans le $\mathcal{O}$ provient de la synchronisation en temps n, indispensable pour matérialiser la fin du regroupement (environ 5 états) et de l'information supplémentaire pour coder le déplacement des signaux droite-gauche et gauche-droite décrits plus haut. La figure 7.5 illustre la structure d'une cellule du nouvel automate. La figure 7.6 montre les différents composants produit global.

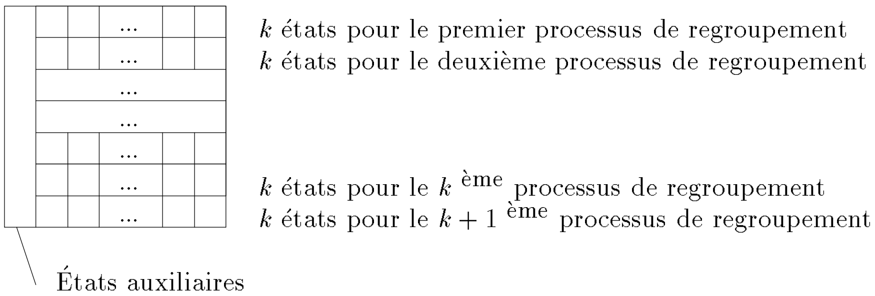


FIG. 7.5 – *Structure d'une cellule de l'automate effectuant le regroupement global.*

1.5 Terminer la synchronisation

La dernière étape de notre construction concerne l'apparition de l'état **feu** sur toute la longueur de la ligne. La figure 7.5 montre que chaque cellule simulante contient $k+1$ « couches » de k états chacunes. En pratique, à la fin du processus de regroupement, au temps n, la plupart de ces couches sont entièrement quiescentes (leur état vaut $\sharp^k$). Seules une ou deux couches, selon que la cellule se trouve sur une ou deux zones regroupées, contiennent des états significatifs et simulent véritablement un calcul. L'état **feu** apparaîtra bien partout en même temps (puisqu'aucun processus de regroupement ne prend de retard sur les autres) mais *pas nécessairement dans les mêmes couches.*

Appelons c_f la première configuration pour laquelle des états **feu** ont été simulés. En aucun cas elle ne peut être considérée comme une configuration globalement synchronisée puisque tous les états ne sont pas *exactement* les mêmes (ils simulent bien des états **feu** mais pas dans les mêmes couches). Or, c'est une des conditions fortes de la définition 2.1, l'état **feu** doit être unique (sans ça, la composition des synchroniseurs est plus compliquée, *cf.* section suivante et généralisation).

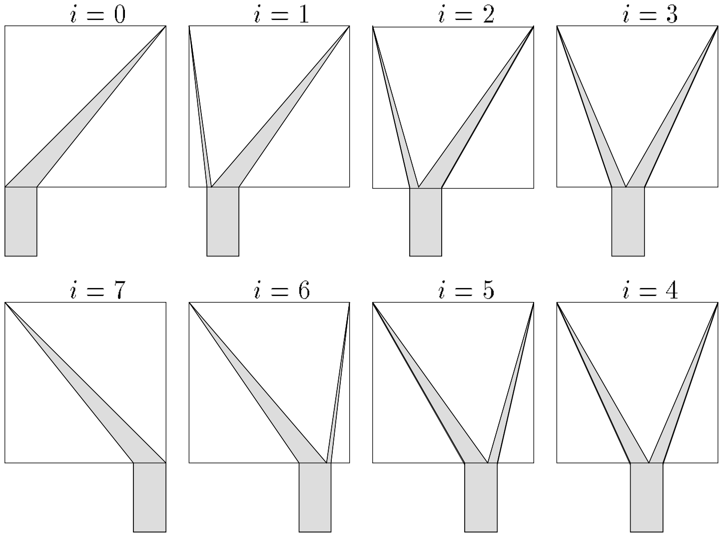


FIG. 7.6 – *Les $k+1$ regroupements élémentaires pour l'accélération d'un facteur (ici $k = 7$).*

Pour remédier à ce problème, il suffit d'identifier, parmi les $\mathcal{O}(\mathcal{Q}^{k(k+1)})$ états possibles, tous ceux qui contiennent au moins une occurrence de l'état **feu**.

Ceci termine la construction d'un automate simulant le synchroniseur initial en gagnant un nombre linéaire de pas de calcul et prouve le théorème 7.1. L'accélération d'une constante s'exprime comme un corollaire direct :

Corrolaire 7.6 (accélération d'une constante)
Soit k un entier positif. Si $n+t(n)$ est un temps de synchronisation, alors $n+\sup\{0, t(n)-k\}$ est également un temps de synchronisation.

Preuve
Pour gagner k pas de calcul, une accélération linéaire d'un facteur $k+1$ pendant *un seul pas* suffit : elle est mise en place pendant les étapes 0 à n, effective pendant l'étape n à $n+1$ seulement, et inopérante pendant les étapes suivantes (où le calcul se poursuit à vitesse normale).

En fait, il existe une preuve moins spécifique utilisant le théorème général d'accélération d'une constante pour les calculateurs (rappelons que les synchroniseurs sont des calculateurs particuliers), *cf.* chapitre 5 et aussi [CČ83, Hee95a].

□

2 Polynômes et temps de synchronisation

2.1 Combiner des temps de synchronisation

Rappelons tout d'abord les opérations arithmétiques usuelles sur les temps de synchronisation. Pour la somme et le produit, les preuves sont des versions simplifiées et adaptées à nos définitions de celles formulées par J. Mazoyer dans [Maz90]. L'expression de la (pseudo-)différence symétrique est nouvelle.

Lemme 7.7 (somme)
Considérons deux synchroniseurs $\mathcal{S}_1$ et $\mathcal{S}_2$ travaillant en temps $n + t_1(n)$ et $n + t_2(n)$ respectivement. Il est possible de construire un nouveau synchroniseur $\mathcal{S}_1 + \mathcal{S}_2$ travaillant en temps $n + t_1(n) + t_2(n)$.

Preuve
L'idée est celle de J. Mazoyer ([Maz90], §8.1, prop. 4). La simulation se fait en 4 étapes :

1. $\mathcal{S}_1$ et $\mathcal{S}_2$ démarrent en même temps et en parallèle avec une synchronisation en temps minimal ;
2. après n pas de calcul, $\mathcal{S}_2$ est « gelée » (chaque cellule mémorise l'information courante concernant $\mathcal{S}_2$ et ne la modifie plus jusqu'à nouvel ordre) ;
3. l'ordre de « dégel » est lancé dès que la synchronisation $\mathcal{S}_1$ est terminée ;
4. la simulation de $\mathcal{S}_2$ se poursuit normalement jusqu'à apparition de l'état **feu** de $\mathcal{S}_2$, servant également d'état **feu** pour le nouvel automate $\mathcal{S}_1 + \mathcal{S}_2$.

□

Lemme 7.8 (produit)
Considérons deux synchroniseurs $\mathcal{S}_1$ et $\mathcal{S}_2$ travaillant en temps $n + t_1(n)$ et $n + t_2(n)$ respectivement. Il est possible de construire un nouveau synchroniseur $\mathcal{S}_1 \times \mathcal{S}_2$ travaillant en temps $n + t_1(n) \times t_2(n)$.

Preuve
La preuve détaillée apparaît dans [Maz90] (§8.1, prop. 7 et cor. 8). Nous indiquons seulement l'intuition : le processus mis en œuvre rappelle celui de la somme mais, ici, les étapes 2 et 3 sont répétées $t_1(n)$ fois.

□

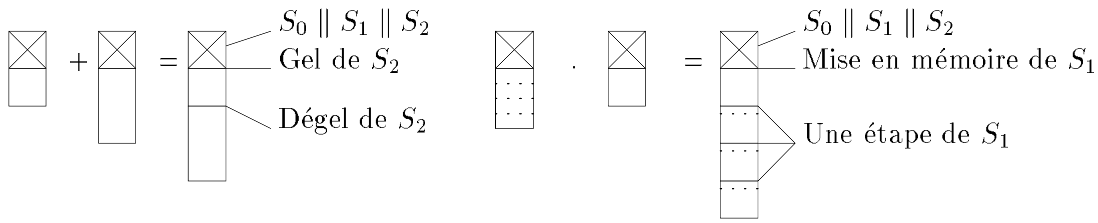


FIG. 7.7 – *Constructions pour la somme et le produit de temps de synchronisation.*

Lemme 7.9 (pseudo-différence)
Considérons deux synchroniseurs $\mathcal{S}_1$ et $\mathcal{S}_2$ travaillant en temps $n + t_1(n)$ et $n + t_2(n)$

respectivement. Pour tout $p \geq 1$, il est possible de construire un nouveau synchroniseur $\mathcal{S}_1 \triangle \mathcal{S}_2$ travaillant en temps :
$n + \sup\{|t_1(n) - t_2(n)|, t_1(n)/p, t_2(n)/p\}$.

Preuve

Grâce au théorème 7.1 et a l'utilisation de délais, nous construisons tout d'abord un synchroniseur fonctionnant en temps $n + \sup\{\lceil t_2(n)/p \rceil, t_1(n) - t_2(n)\}$ (pour tout $p \geq 1$). Le déroulement du calcul comporte les quatre phases suivantes :

1. l'automate réalise un $(p+1)$-regroupement, tout en simulant les n premiers pas du synchroniseur $\mathcal{S}_1$ et les n premiers pas du synchroniseur $\mathcal{S}_2$;

2. puis, grâce à une version $(p+1)$-accélérée de $\mathcal{S}_1$ et une version p-accélérée de $\mathcal{S}_2$, utilisées pendant $\lceil t_2(n)/p \rceil$ unités de temps, l'automate simule $p \lceil t_2(n)/p \rceil$ pas de calcul de $\mathcal{S}_2$ et $n + \inf\{t_1(n), (p+1) \lceil t_2(n)/p \rceil\}$ pas de $\mathcal{S}_1$; notons qu'à l'issue de cette phase, la valeur de $\Theta = p \lceil t_2(n)/p \rceil - t_2(n)$ $(\in \{0, \ldots, p-1\})$ est connue de toutes les cellules ; cette valeur s'avère utile à la fin de la quatrième étape ;

3. cette phase n'a lieu que si la simulation de $\mathcal{S}_1$ n'est pas déjà terminée ; les derniers pas de $\mathcal{S}_1$ sont alors simulés à vitesse normale (rappelons qu'une configuration $(p+1)$-groupée peut simuler des calculs à n'importe quelle vitesse entre 0 et $p+1$) ;

4. si la troisième phase à eu lieu, l'automate cellulaire effectue Θ pas supplémentaires pendant lesquels ni $\mathcal{S}_1$ ni $\mathcal{S}_2$ ne sont simulés (en fait, ils ont déjà été entièrement simulés) ; ces pas supplémentaires représentent juste un délai avant la synchronisation finale, permettant d'obtenir une meilleure expression de la différence (voir calcul ci-après).

Soit Γ le temps total écoulé, on a :

$$\begin{aligned}
\Gamma &= n + \lceil t_2(n)/p \rceil + t_1(n) - \inf\{t_1(n), (p+1) \lceil t_2(n)/p \rceil + \Theta\} \\
&= n + \sup\{\lceil t_2(n)/p \rceil, t_1(n) - p \lceil t_2(n)/p \rceil + p \lceil t_2(n)/p \rceil - t_2(n)\} \\
&= n + \sup\{\lceil t_2(n)/p \rceil, t_1(n) - t_2(n)\}
\end{aligned}$$

En intervertissant les rôles de $\mathcal{S}_1$ et $\mathcal{S}_2$ dans la construction ci-dessus, on obtient une expression symétrique de la différence : $n + \sup\{\lceil t_1(n)/p \rceil, t_2(n) - t_1(n)\}$. Le lemme 7.9 est obtenu en lançant en parallèle la construction ci-dessus et sa version symétrisée.

□

2.2 Fermeture pour les polynômes à coefficients rationnels

Définition 7.2 *La famille Sync des* délais *de synchronisation par rapport au temps de synchronisation minimal (n) est définie par :*

$$Sync = \{t(n), n + t(n) \text{ est un temps de synchronisation } \}$$

Grâce aux opérations définies ci-avant et au lemme 7.11 entre autres on prouve le théorème suivant :

Théorème 7.10 *Si $(n \mapsto t(n))$ est un délai de synchronisation et $P \in \mathbb{Q}[n]$ est un polynôme à coefficients rationnels et valeurs positives, alors $(n \mapsto \lceil P(t(n)) \rceil)$ est aussi un délai de synchronisation.*

En d'autres termes, l'ensemble *Sync* est clos pour la composition avec les polynômes à coefficients rationnels et valeurs positives.

La preuve détaillée de ce théorème utilise le lemme suivant qui stipule qu'une fonction différant très rarement d'un temps de synchronisation est en fait un temps de synchronisation :

Lemme 7.11 (modifications finies)

L'ensemble Sync est clos pour les modifications finies, c'est-à-dire : si $(n \mapsto t(n)) \in Sync$, *si* $\forall n \geq 1,\ t'(n) \geq 0$ *et si de plus l'ensemble* $E = \{m : t'(m) \neq t(n)\}$ *est fini, alors* $(n \mapsto t'(n)) \in Sync$.

Preuve

Dès le démarrage du processus de synchronisation en $n+t(n)$ étapes, chaque cellule peut vérifier, en mois de n étapes si n est dans l'ensemble E ou non (ce test nécessite de matérialiser le temps n par une synchronisation auxiliaire en temps minimal). Si c'est le cas, toutes les cellules décomptent $t'(n)$ étapes et entrent dans l'état **feu**. Sinon, la synchronisation en $n+t(n)$ étapes continue normalement.

□

Avant de prouver le théorème 7.10, voici un récapitulatif des opérations applicables aux délais de temps de synchronisation : si $t(n), t_1(n), t_2(n) \in Sync$ et $c \geq 0, p \geq 1$ alors

- $t(n) + c \in Sync$ (ralentissement d'une constante, trivial).
- $\sup\{0, t(n) - c\} \in Sync$ (accélération d'une constante, corollaire 7.6 ou [CČ83, Hee95a]).
- $c.t(n) \in Sync$ (ralentissement linéaire, lemme 7.7).
- $\lceil t(n)/p \rceil \in Sync$ (accélération linéaire, théorème 7.1).
- $t_1(n) + t_2(n) \in Sync$ (somme, lemme 7.7, [Maz90]).
- $t_1(n).t_2(n) \in Sync$ (produit, lemme 7.8, [Maz90])
- $\sup\{t_1(n)/(p+1), t_2(n)/(p+1), |t_1(n) - t_2(n)|\} \in Sync$ (pseudo-différence, lemme 7.9).

Nous disposons maintenant de tous les éléments pour prouver le théorème 7.10.

Preuve

Soient $(n \mapsto t(n)) \in Sync$ un délai de synchronisation et $P \in \mathbb{Z}[X]$, $P(X) = a_0 X^d + a_1 X^{d-1} + \cdots + a_{d-1} X + a_d$ $(a_0 \neq 0)$ un polynôme à valeurs positives et coefficients entiers (éventuellement négatifs). Nous établissons tout d'abord que $P(t(n))$ est dans l'ensemble *Sync*.

Considérons la forme de Horner de P :

$$P(X) = (\ldots(((a_0 X + a_1)X + a_2)X + \ldots) + a_{k-1})X + a_k$$

On sait que a_0 est strictement positif puisque $\forall X \geq 1, P(X) > 0$, par conséquent $(n \mapsto a_0 t(n))$ est un délai de synchronisation. Mais *a priori*, l'expression $a_0 X + a_1$ peut prendre des valeurs négatives (si $a_1 < 0$), et rien n'indique que $(n \mapsto a_0 t(n) + a_1)$ est un délai de synchronisation. En revanche, grâce aux théorèmes d'accélération et décélération d'une constante on sait que : $(n \mapsto \sup\{0, a_0 t(n) + a_1\}) \in Sync$.

Ceci nous conduit à introduire une forme modifiée du polynôme P :

$$P'(X) = (\ldots(((a_0X \oplus a_1)X \oplus a_2)X \oplus \ldots) \oplus a_{k-1})X \oplus a_k \text{ où } x \oplus y = \sup\{0, x+y\}$$

Grâce au lemme 7.8 (produit) et à l'accélération ou décélération d'une constante, on a $P'(t(n)) \in Sync$.

Il est facile de vérifier que $P(t(n)) = P'(t(n))$ sauf pour un nombre *fini* de valeurs de n. Or l'ensemble $Sync$ est clos pour les modifications finies (lemme 7.11), d'où $(n \mapsto P(t(n))) \in Sync$.

Le cas général d'un polynôme appartenant à $\mathbb{Q}[X]$ et à valeurs positives est obtenu à partir du cas précédent grâce à une accélération linéaire. En effet, il existe toujours $P \in \mathbb{Z}[X]$ à valeurs positives et $q \in \mathbb{N}$ tels que $\forall X \in \mathbb{N}, Q(X) = P(X)/q$.

□

Corrolaire 7.12

Les polynômes à coefficients rationnels (et plus grands que l'identité) sont des temps de synchronisation.

C'est un cas particulier du théorème 7.10, pour $t(n) = n, \forall n \in \mathbb{N}$.

3 Généralisations et conclusions

Le théorème d'accélération linéaire reste vrai pour les synchroniseurs démarrant sur une configuration quelconque (et non plus uniformément quiescente comme le suggère la définition 2.5). En effet, le calcul de l'automate initial est entièrement inclus dans celui de l'automate simulant (qui reproduit donc la sensibilité par rapport à la configuration de départ).

Remarquons toutefois qu'accepter des configurations d'entrée quelconques autorise des synchronisations en temps constant pour tout n (elles sont dites triviales). Bien entendu, la construction donnée pour l'accélération linéaire ne s'applique pas à ces synchronisations puisqu'elle ne satisfont pas toutes les hypothèses requises : elles n'excèdent pas un certain minimum (n ou $2n$ selon que les synchronisations sont autorisées à partir des deux extrémités de la ligne ou non).
Par ailleurs, dans la définition des synchroniseurs il est bien spécifié que l'état **feu** ne doit apparaître nulle part avant la fin du calcul (sans cette condition, le problème de la synchronisation perd beaucoup d'intérêt). Mais notre construction pour l'accélération linéaire n'utilise jamais cette hypothèse : elle reste correcte même si l'état **feu** apparaît prématurément pour certaines cellules (du moment qu'à la fin du calcul *toutes* les cellules sont bien dans l'état **feu**).

En fait, il faut tout de même modifier très légèrement la fin de la construction à cause du fait suivant : si l'état **feu** peut apparaître au cours des calculs, il risque de servir à la poursuite dudit calcul et il n'est plus possible d'identifier directement tous les états simulant qui contiennent une occurrence de l'état **feu** (sous peine d'interférer dans les simulations de la synchronisation initiale). Ce que l'on obtient alors, c'est un synchroniseur possédant non pas un seul mais plusieurs états **feu** (moralement équivalents entre eux).

Les deux remarques précédentes conduisent à une reformulation un peu plus générale du théorème 7.1 :

Théorème 7.13 (accélération linéaire pour les calculateurs unaires)
Tout calculateur non trivial possédant un seul symbole de sortie peut subir une accélération linéaire.

Nous proposons également une généralisation du théorème 7.10 aux polynômes à plus d'une variable. Soit P un polynôme de $\mathbb{Q}[X_1, \ldots, X_v]$ où tous les monômes de plus haut degré ont des coefficients positifs, alors P peut être séparé en deux parties, l'une positive, l'autre négative : $P = P^+ - P^-$. On prouve facilement que P^+ et P^- sont des temps de synchronisation dès lors que $X_1, \ldots, X_v$ sont des temps de synchronisation. Par ailleurs, si P^+ et P^- sont suffisamment séparés, c'est-à-dire s'il existe $\varepsilon > 0$ tel que $\forall n\ P^+(n) \geq (1+\varepsilon)P^-(n)$, on peut utiliser la pseudo-différence (lemme 7.9) pour établir que $(n \mapsto P(n))$ est un temps de synchronisation.

Chapitre 8

Codage économique de signaux élémentaires

Résumé

Les signaux élémentaires (c'est-à-dire de pente p entière) représentent le moyen le plus simple pour véhiculer de l'information dans un diagramme espace-temps d'automate cellulaire. Notre étude vise à minimiser le nombre d'états nécessaires à la réalisation de tels signaux. Le résultat théorique essentiel est une reformulation du théorème de lacune pour les signaux fortement croissants, tandis que le résultat pratique est la réalisabilité de tous les signaux « carrés » (dont la pente est identique à l'épaisseur) à l'aide 5 états au plus.

1 Définitions

Tout d'abord nous rappelons la définition de signal ; intuitivement c'est un déplacement d'information dans l'espace et dans le temps.

Définition 8.1 (signaux, voir [MT93])

- *un **signal** S est un ensemble de sites $\{(c(t), t) : t \in I\!N\}$ avec $c : I\!N \setminus \{0\} \to I\!N$ telle que $c(t+1) \in \{c(t)-1, c(t), c(t)+1\}$;*
- *une fonction $f : I\!N \to I\!N$ croissante est **Fischer-constructible** s'il existe un automate cellulaire et un sous-ensemble Q_f d'états tels que $< 0, t > \in Q_f \iff \exists i / \delta(i) = t$;*
- *une fonction $g : I\!N \to I\!N$ est **AC-constructible** si $\{(c,t)/g(c) \leq t < g(c+1)\}$ est un signal.*

Lemme 8.1

Toute fonction f croissante AC-constructible est Fischer-constructible.

La preuve est donnée dans [Ter91]. Par la suite nous utiliserons essentiellement la notion d'AC-constructibilité. Notons que la réciproque est fausse.

La définition suivante ne s'applique qu'aux signaux AC-construits.

Définition 8.2 (demi-espaces limités par un signal)

Considérons le signal S,

- *$Dans(S) = \{(i,t)/n \in I\!N, 0 \leq i < n\}$ est l'intérieur de S ;*

– $Hors(S) = \{(i, \delta(n))/n \in I\!N, i > n\}$ *est l'extérieur de* S.

Pour les signaux AC-construits il existe une notion de mouvement. En effet, si (n, t) est un site du signal S, alors le prochain site est nécessairement $(n-1, t+1)$, $(n, t+1)$ ou $(n+1, t+1)$, c'est-à-dire obtenu à partir du précédent par un mouvement à gauche, sur place ou à droite. Une nouvelle façon de définir un signal est de donner la suite des mouvements qui le composent (codés par G, S, D ou -1, 0, 1 selon le contexte).

Un signal *va à droite* quand il ne contient pas de mouvement gauche. De tels signaux jouent un rôle particulier par la suite.

Définition 8.3 (signal périodique, élémentaire)
Un signal est dit ***périodique*** *quand la suite des mouvements qui le compose devient ultimement périodique. Si de plus la phase transitoire est nulle, alors le signal est dit* ***élémentaire****. Soient* T *la taille de la partie périodique et* M *la somme des mouvements élémentaires sur une période, alors la* pente *du signal est le rapport* T/M.

Remarque
Bien entendu, la pente est toujours supérieure ou égale à 1, car l'information ne peut pas avancer de plus d'une cellule par unité de temps (sur un automate cellulaire simple, donc à voisinage de plus proches voisins).

La section suivante est consacrée à l'étude des signaux élémentaires qui vont à droite. Ils représentent, en un sens, le mode de transmission d'information le plus simple pour les automates cellulaires.

En pratique, on utilise souvent des signaux aller/retour, c'est-à-dire des paires de signaux, l'un allant à droite et l'autre possédant les mêmes propriétés de construction mais se déplaçant dans le sens inverse. Les résultats annoncés en section 3 sont valables pour des signaux aller/retour.

2 Exemples de signaux, propriétés

Cette section est consacrée à l'étude de quelques signaux classiques, dont certains jouent un rôle particulier dans la section suivante et de leurs propriétés fondamentales.

LOG_b En un certain sens, c'est le signal inverse de EXP_b, formellement $LOG_b = \{(n, n + \log_b(n)), n \in I\!N\}$. Nous n'utiliserons pas directement ce signal dans la section suivante, en revanche il joue un rôle important dans la littérature, notamment à propos du théorème de lacune, décrit dans [Ter91, MT93, Rei93] et dont nous rappelons l'énoncé plus loin.

$DROIT_p$ C'est un signal élémentaire de pente $p > 0$. Formellement, $DROIT_p = \{(i, t) : ip \leq t < (i+1)p\}$. De tels signaux abondent dans la littérature, en particulier pour la construction de solutions au problème de la synchronisation (*cf.* 5). On peut caractériser un signal $DROIT_p$ par un paramètre l, qui est la largeur maximale de la zone non quiescente laissée par le signal (on dit également son épaisseur). Par la suite nous noterons $DROIT_{l,p}$ le signal de pente p et d'épaisseur l : $DROIT_{l,p} = \{(i-j, t) : ip \leq t < (i+1)p, 0 \leq i < l\}$. Les définitions de S et $Hors(S)$ restent identiques, en revanche $Dans(S)$ est modifié : $Dans(S, l) = \{(i, \delta(n)), n \in I\!N, 0 \leq i < \delta(n) - l\}$. On définit symétriquement les signaux $GAUCHE_{l,p}$ et on utilise la notation $LIN_{l,p}$ pour parler de $DROIT_{l,h}$ et de $GAUCHE_{l,h}$ en même temps (signaux « aller et retour »). Ce chapitre est consacré à l'étude de ces signaux.

$QUAD$ De nombreuses constructions pour les signaux quadratiques on été décrites (*cf.* les travaux de J. Mazoyer ou V. Terrier). $QUAD$ est un exemple simple de signal compris entre LIN_p et EXP_b.

EXP_b C'est le signal exponentiel en base b, formellement $\{(n, b^n + n), n \in I\!N\}$. Une façon de réaliser un tel signal consiste à énumérer les entiers en base b successivement sur les diagonales. Chaque cellule doit pouvoir stocker un entier entre 0 et $b-1$ et une retenue éventuelle nécessaire au processus d'incrémentation, soir $2b$ valeurs distinctes. Il est à noter que ce type de construction laisse le demi-espace $Hors(EXP_b)$ entièrement quiescent. C'est la transposition aux automates cellulaires de la construction usuelle sur *one-way CA* (*cf.* [IJ87, Ter95]). D'autres méthodes de construction existent pour ce signal (*cf.* [CČ83]). On a $EXP_b \in AC(2b)$. La figure 8.1 donne un exemple de signal exponentiel. En fait, il existe des méthodes de construction plus économiques en nombre d'états mais elles ne laissent pas $Hors(EXP_b)$ entièrement quiescent. Nous développons cette remarque dans la section suivante.

$FACT$ C'est le signal factoriel, décrit notamment en [MT93]. On passe du signal Fischer-construit au signal AC-construit grâce au lemme 8.1. Ce signal croît plus vite que tout signal EXP_b. Il est a noter qu'on n'en connaît pas de construction laissant $Hors(FACT)$ entièrement quiescent (ce point est détaillé par la suite, voir lemme 8.6).

Le résultat essentiel connu sur les signaux est le théorème de lacune qui indique que certains signaux ne sont pas AC-constructibles.

Théorème 8.2 (théorème de lacune [MT93, Ter91])
Pour tout signal à droite $S = \{(n, n + \delta(n)), n \in I\!N\}$*, la fonction* $(n \mapsto \delta(n))$ *devient constante ou il existe un entier naturel* b *tel que* $\forall n \in I\!N, s(n) \geq \log_b(n)$.

3 Minimum d'états pour réaliser un signal élémentaire

Cette section est consacrée à l'étude de la fonction $E(l,p)$, informellement le nombre minimum d'états nécessaires à la réalisation des signaux élémentaires $LIN_{l,p}$.

Ce nombre est lié à $e(l,p)$, le cardinal de l'ensemble $\mathcal{Q}$ du plus petit automate cellulaire qui réalise $LIN_{l,p}$ en partant d'une configuration initiale ne dépendant que du signal lui même (pas obligatoirement la configuration $100\ldots$).

L'état quiescent, toujours présent dans $\mathcal{Q}$ pour un automate cellulaire simple, est compté dans $e(l,p)$.

En tout état de cause, $E(l,p) \leq 2e(l,p)$ et les deux fonctions présentent des caractéristiques communes. Dans le cas des signaux larges, des motivations pratiques (voir introduction) nous poussent à étudier $E(l,p)$. Dans le cas des signaux fins, les valeurs exactes nous intéressent moins et nous étudierons seulement $e(l,p)$, en nous contentant de la majoration ci-avant pour $E(l,p)$.

```
0>  -   -   -   -   -   -   -   -
1>  *   -   -   -   -   -   -   -
0>  0>  -   -   -   -   -   -   -
1>  1   *   -   -   -   -   -   -
0>  1>  *   -   -   -   -   -   -
1>  0   0>  -   -   -   -   -   -
0>  0>  0>  *   -   -   -   -   -
1>  1   1   *   -   -   -   -   -
0>  1>  1   *   -   -   -   -   -
1>  0   1>  *   -   -   -   -   -
0>  0>  0   0>  -   -   -   -   -
1>  1   0   0>  *   -   -   -   -
0>  1>  0   0>  *   -   -   -   -
1>  0   0>  0>  *   -   -   -   -
0>  0>  1   1   *   -   -   -   -
1>  1   1   1   *   -   -   -   -
0>  1>  1   1   *   -   -   -   -
1>  0   1>  1   *   -   -   -   -
0>  0>  0   1>  *   -   -   -   -
1>  1   0   0   0>  -   -   -   -
0>  1>  0   0   0>  *   -   -   -
1>  0   0>  0   0>  *   -   -   -
0>  0>  0>  0   0>  *   -   -   -
1>  1   1   0   0>  *   -   -   -
0>  1>  1   0   0>  *   -   -   -
1>  0   1>  0   0>  *   -   -   -
0>  0>  0   0>  0>  *   -   -   -
1>  1   0   1   1   *   -   -   -
0>  1>  0   1   1   *   -   -   -
1>  0   0>  1   1   *   -   -   -
0>  0>  1   1   1   *   -   -   -
1>  1   1   1   1   *   -   -   -
0>  1>  1   1   1   *   -   -   -
1>  0   1>  1   1   *   -   -   -
0>  0>  0   1>  1   *   -   -   -
1>  1   0   0   1>  *   -   -   -
0>  1>  0   0   0   0>  -   -   -
1>  0   0>  0   0   0>  *   -   -
0>  0>  0>  0   0   0>  *   -   -
1>  1   1   0   0   0>  *   -   -
0>  1>  1   0   0   0>  *   -   -
```

```
(0>,*  ,-  ) -> (*  )    (0>,1>,0  ) -> (0  )    (1>,*  ,-  ) -> (0>)
(0>,-  ,-  ) -> (*  )    (0>,1>,1  ) -> (0  )    (1>,0  ,0  ) -> (0>)
(1  ,*  ,-  ) -> (*  )    (1  ,0  ,0  ) -> (0  )    (1>,0  ,1>) -> (0>)
(*  ,-  ,-  ) -> (-  )    (1  ,0  ,0>) -> (0  )    (1>,0  ,0>) -> (0>)
(-  ,-  ,#  ) -> (-  )    (1  ,0  ,1  ) -> (0  )    (1>,0  ,1  ) -> (0>)
(-  ,-  ,-  ) -> (-  )    (#  ,0>,-  ) -> (1>)    (0  ,0>,0>) -> (1  )
(0  ,0  ,0  ) -> (0  )    (#  ,0>,1>) -> (1>)    (0  ,0>,1  ) -> (1  )
(0  ,0  ,1>) -> (0  )    (#  ,0>,0>) -> (1>)    (0  ,1  ,1  ) -> (1  )
(0  ,0  ,0>) -> (0  )    (1>,1  ,*  ) -> (1>)    (0>,0>,*  ) -> (1  )
(0  ,0  ,1  ) -> (0  )    (1>,1  ,0  ) -> (1>)    (0>,0>,-  ) -> (1  )
(0  ,1>,*  ) -> (0  )    (1>,1  ,1  ) -> (1>)    (0>,0>,0  ) -> (1  )
(0  ,1>,0  ) -> (0  )    (#  ,1>,*  ) -> (0>)    (0>,0>,0>) -> (1  )
(0  ,1>,1  ) -> (0  )    (#  ,1>,0  ) -> (0>)    (0>,0>,1  ) -> (1  )
(0>,0  ,0  ) -> (0  )    (#  ,1>,1  ) -> (0>)    (0>,1  ,1  ) -> (1  )
(0>,0  ,1>) -> (0  )    (0  ,0>,*  ) -> (0>)    (1  ,1  ,*  ) -> (1  )
(0>,0  ,0>) -> (0  )    (0  ,0>,-  ) -> (0>)    (1  ,1  ,0  ) -> (1  )
(0>,0  ,1  ) -> (0  )    (0  ,0>,0  ) -> (0>)    (1  ,1  ,1  ) -> (1  )
```

FIG. 8.1 – *Réalisation d'un signal exponentiel avec 6 états.*

Avant de commencer l'étude détaillée, emmettons quelques remarques de bon sens et quelques principes de base pour la construction de signaux élémentaires :

Lemme 8.3

Tout signal S, AC-constructible par un automate qui laisse $Dans(S, l)$ (pour un certain l) et $Hors(S)$ entièrement quiescents est ultimement périodique.

Preuve

Puisque $Dans(S, l)$ et $Hors(S)$ sont entièrement quiescents (pour un certain l), deux configurations de l'automate réalisant S diffèrent au plus de l cellules. Si l'automate concerné possède s états, le signal S contient au maximum s^l lignes distinctes. Au delà, il y a obligatoirement une répétition. La table de transition d'un automate cellulaire étant fixée et déterministe, le signal est ultimement périodique (la période et le régime transitoire sont tous deux plus petits que s^l).

□

Inversement :

Lemme 8.4

On peut construire $LIN_{l,p}$ en laissant $Dans(LIN_{l,p}, l)$ et $Hors(LIN_{l,p})$ entièrement quiescents.

Preuve

Il suffit de mettre au point l'automate à p états qui passe de la configuration $\ldots 0.0.i^l.0.0\ldots$ à la configuration $\ldots 0.0.(i+1)^l.0.0\ldots$, et aussi de la configuration $\ldots 0.0..p^l.0.0\ldots$ à la configuration $\ldots 0.0.0.1^l.0\ldots$ (il y a un décalage vers la droite).

Cet automate existe et sa table de transition est décrite par les règles suivantes :

Cas général

$\delta(0,0,i) = 0$, $\delta(0,i,i) = i+1$, $\delta(i,i,i) = i+1$, $\delta(i,i,0) = i+1$, $\delta(i,0,0) = 0$.

Mouvement vers la droite

$\delta(0,p,p) = 0$, $\delta(p,p,p) = 1$, $\delta(p,0,0) = 1$.

□

Ceci fourni une première approximation, grossière, de la valeur de $e(l,p)$:

$$\forall (l,p) \in I\!N^2, e(l,p) \leq p$$

En dupliquant tous les états, on obtient :

$$\forall (l,p) \in I\!N^2, E(l,p) \leq 2p$$

Les majorations indiquées ci-avant sont très larges et, dans bien des cas, on peut faire mieux. En particulier, l'intuition suggère que $E(l,p)$ décroît quand l croît (notion de *troc* entre espace et nombre d'états). Les sous-sections suivantes sont consacrées à l'étude de $e(l,p)$ en fonction du rapport p/l.

3.1 Cas des signaux larges

Nous dirons que le signal élémentaire $LIN_{l,p}$ est large si le rapport p/l est inférieur ou égal à un.

Lemme 8.5 (signaux élémentaires larges)

Si $p \leq l$, le nombre d'états nécessaires pour réaliser le signal $LIN_{l,p}$ ne dépend ni de p ni de l. On a $\exists c$ t.q. $\forall l,p,(p \leq l)$, $E(l,p) \leq c$.

Preuve

Nous donnons une preuve par construction : on veut réaliser le signal $LIN_{l,p}$, en partant d'une configuration presque partout quiescente au temps 0 (plus précisément, la zone non quiescente est de longueur au plus l). Soit n le numéro de la première case non quiescente. On sait construire $LIN_{l,p}$ si l'on sait *marquer* les sites (n,p) et $(n+l,p)$. Marquer un site c'est y réaliser une condition particulière, fausse pour les autres sites (typiquement lui attribuer un état particulier, inutilisé ailleurs). Pour cela, des signaux auxiliaires α_1 et α_{-1} doivent être mis en place dès le début et voyager à vitesse maximale vers les sites à distinguer (il s'agit ici de signaux auxiliaires *à l'intérieur* du signal $LIN_{b,l}$).

Les sites de départ de α_1 et α_{-1} sont respectivement $(n+p,0)$ et $(n+l-p,0)$. Ainsi, nous pouvons obtenir une configuration du type : $\ldots,0,0,\underbrace{a_1,?,\ldots,?,a_{-1}}_{l},0,0,\ldots$ où a_1 et a_{-1} sont les états particuliers correspondant à l'arrivée des signaux de construction α_1 et α_{-1}.

```
---1oooo2oo3oooo4---5xxxx6xx7xxxx8-----1ooo2oo3ooo4---5xxx6xx7xxx8---
---o\oo/|oo|\oo/o---x\xx/|xx|\xx/x-----o\o/|oo|\o/o---x\x/|xx|\x/x---
---oo\/o|oo|o\/oo---xx\/x|xx|x\/xx-----ooxo|oo|oxoo---xxox|xx|xoxx---
---oo/\o|oo|o/\oo---xx/\x|xx|x/\xx-----o/o\|oo|/o\o---x/x\|xx|/x\x---
---o/oo\|oo|/oo\o---x/xx\|xx|/xx\x-----aooobooccoood---AxxxBxxCxxxD---
---aoooobooccooood---AxxxxBxxCxxxxD----1ooo2oo3ooo4-----5xxx6xx7xxx8--
--1oooo2oo3oooo4-----5xxxx6xx7xxxx8---o\o/|oo|\o/o-----x\x/|xx|\x/x--
--o\oo/|oo|\oo/o-----x\xx/|xx|\xx/x---ooxo|oo|oxoo-----xxox|xx|xoxx--
--oo\/o|oo|o\/oo-----xx\/x|xx|x\/xx---o/o\|oo|/o\o-----x/x\|xx|/x\x--
```

FIG. 8.2 – *Signal large ($l > 2p$) réalisé avec 22 états.*

Pour pouvoir recommencer le processus lors de la période suivante, il faut aussi marquer les sites de lancement de α_1 et α_{-1}. Là encore nous utilisons des signaux de construction, β_1 et β_{-1}, mis en place à partir des sites $(n, 0)$ et $(n + l, 0)$ (qui sont déjà distingués).

L'élément clef de cette preuve est que, l étant plus grand que p, les pentes des signaux de construction (1 et -1) ne dépendent pas de p. Ces derniers peuvent donc être réalisés avec un nombre constant d'états.

□

Évaluation de la constante obtenue dans différents cas :

1. si $l > 2p$, alors $c \leq 9$. Ce résultat a été obtenu par améliorations successives d'une solution à 22 états. Le taux d'occupation de la table de transition, voisin de 18% seulement, laisse espérer des solutions en moins d'états (un autre argument, cité plus bas, va également dans ce sens). La figure 8.2 montre un fragment de diagramme espace-temps pour la solution initiale a 22 états, la figure 8.3 illustre la construction à 9 états ;

2. si $p < l < 2p$, alors les signaux de construction doivent pouvoir se croiser sans interférer. En première approximation 9^2 états suffisent. La figure 8.4 illustre (partiellement) la construction d'un tel signal ;

3. un résultat est très important en pratique : si $l = p$ alors la constante tombe à 5. Autrement dit, tous les signaux $LIN_{p,p}$, dits également signaux *carrés*, sont réalisables avec 5 états au plus. La figure 8.5 donne une des constructions a 5 états ;

4. pour certains cas particuliers (petites valeurs de p et l), 3 états suffisent. Le cas du signal carré $LIN_{3,3}$ en particulier est intéressant car il n'existe qu'une seule façon de le réaliser avec trois états (modulo symétries et renommages). Notre intuition est que tous les signaux carrés peuvent être réalisés avec trois états. On peut écrire des algorithmes de recherche systématique, utilisant éventuellement des heuristiques pour réduire l'explosion combinatoire. Quelques codages optimaux sont indiqué dans la figure 8.6.

Remarque

Nous pensons que les signaux $LIN_{3,3}$ peuvent aider à la construction de solutions synchronisantes de type Minsky avec très peu d'états (*cf.* [Min67, Yun93, Yun94]. Une solution partielle à 6 états utilisant ces signaux est décrite en section 4 de ce chapitre).

3.2 Cas des signaux fins

Un signal fin est un signal dont la largeur est strictement inférieure à la pente. Ce sont les plus utilisés dans la littérature, notamment les signaux de largeur 1 ou 2.

```
---\oooooo4oooo1oooooo/-----\xxxxxx4xxxx1xxxxxx/----
---o\oooo/4oooo1\oooo/o-----x\xxxx/4xxxx1\xxxx/x----
---oo\oo/14oooo14\oo/oo-----xx\xx/14xxxx14\xx/xx----
---ooo\/o14oooo14o\/ooo-----xxx\/x14xxxx14x\/xxx----
---ooo/\o14oooo14o/\ooo-----xxx/\x14xxxx14x/\xxx----
---oo/oo\14oooo14/oo\oo-----xx/xx\14xxxx14/xx\xx----
---o/oooo\4oooo1/oooo\o-----x/xxxx\4xxxx1/xxxx\x----
---/oooooobooooocoooooox-----oxxxxxxbxxxxcxxxxxx\----
--\oooooo4oooo1oooooo/-------\xxxxxx4xxxx1xxxxxx/---
--o\oooo/4oooo1\oooo/o-------x\xxxx/4xxxx1\xxxx/x---
--oo\oo/14oooo14\oo/oo-------xx\xx/14xxxx14\xx/xx---
```

```
---\ooooo4oooo1ooooo/-----\xxxxx4xxxx1xxxxx/---
---o\ooo/4oooo1\ooo/o-----x\xxx/4xxxx1\xxx/x---
---oo\o/14oooo14\o/oo-----xx\x/14xxxx14\x/xx---
---oooxo14oooo14oxooo-----xxxox14xxxx14xoxxx---
---oo/o\14oooo14/o\oo-----xx/x\14xxxx14/x\xx---
---o/ooo\4oooo1/ooo\o-----x/xxx\4xxxx1/xxx\x---
---/oooooboooocoooooox-----oxxxxxbxxxxcxxxxx\---
--\ooooo4oooo1ooooo/-------\xxxxx4xxxx1xxxxx/--
--o\ooo/4oooo1\ooo/o-------x\xxx/4xxxx1\xxx/x--
```

FIG. 8.3 – *Signal large ($l > 2p$) réalisé avec 9 états. À gauche le cas pair, à droite le cas impair.*

```
------1...........2......3...........4-----
------.\..........!>..../|..........<.-----
------..\.........!.>../.|.........<..-----
------...\........!..>/..|........<...-----
------....\.......!../>..|.......<....-----
------.....\......!./..>.|......<.....-----
------......\.....!/....>|.....<......-----
------.......\....(......]....<.......-----
------........\../!......|>..<........-----
------.........\/.!......|.><.........-----
------........./\.!......|.<>.........-----
------......../..\!......|<..>........-----
------......./....)......[....>.......-----
------....../.....!\....<|.....>......-----
------...../......!.\..<.|......>.....-----
------..../.......!..\<..|.......>....-----
------.../........!..<\..|........>...-----
------../.........!.<..\.|.........>..-----
------./..........!<....\|..........>.-----
------a...........b......c...........d-----
-----1...........2......3...........4------
----..\.........!.>../.|.........<..-------
----...\........!..>/..|........<...-------
----....\.......!../>..|.......<....-------
----.....\......!./..>.|......<.....-------
```

FIG. 8.4 – *Exemple de signal large d'épaisseur comprise entre p et $2p$.*

Nous nous proposons d'encadrer $e(l,p)$ entre un minorant, noté $m(l,p)$, et des majorants de plus en plus fins, $M_1(l,p)$ et $M_2(l,p)$. Tout d'abord, considérons un signal $LIN_{l,p}$ réalisé par un automate cellulaire à k états. Nécessairement $p \leq k^l$ (sinon il y aurait une répétition au bout de k^l étapes et le signal ne serait pas de pente p). Autrement dit, $m(l,p) \geq p^{1/l}$.

Remarque

Pour tout couple (l_1, p_1) on peut trouver un couple (l_2, p_2) tel que $m(l_2, p_2) > m(l_1, p_1)$. Donc tous les signaux de la zone définie par $p \leq k^l$ (pour un certain k) ne sont pas réalisables avec un nombre d'états fixé à l'avance (il y aura toujours un signal particulier nécessitant plus d'états). Cette situation est contraire à celle constatée pour les signaux larges.

Un premier majorant peut être obtenu facilement en appliquant les principes de construction des signaux carrés. Bien entendu, la pente des signaux de construction dépend maintenant de p. Plus précisément, puisque l'on a $p = l \times (p \ div \ l) + p \ mod \ l$, il faut $(p \ div \ l) + (p \ mod \ l)$ états pour construire le premier signal auxiliaire et autant pour le deuxième. En outre, ces deux signaux doivent se croiser sans interférer. ce qui peut demander des états supplémentaires et il faut distinguer les frontières du signal. Au total, $M_1(l,p) = 2 \times (p \ div \ l + p \ mod \ l) + \mathcal{O}(1)$.

Le théorème suivant nous permet d'obtenir une majoration beaucoup plus fine, proche en fait de la fonction m.

Théorème 8.6 (signaux fortement croissants)

```
-----\oooooo/----\xxxxxx/----     ----\ooooo/----\xxxxx/--------
-----o\oooo/o----x\xxxx/x-----     ---o\ooo/o----x\xxx/x--------
-----oo\oo/oo----xx\xx/xx-----     ---oo\o/oo----xx\x/xx--------
-----ooo\/ooo----xxx\/xxx----     ----oooxooo----xxxoxxx--------
-----ooo/\ooo----xxx/\xxx----     ----oo/o\oo----xx/x\xx--------
-----oo/oo\oo----xx/xx\xx----     ----o/ooo\o----x/xxx\x--------
-----o/oooo\o----x/xxxx\x----     ----/oooooox----oxxxxx\--------
-----/ooooooox----oxxxxxx\----    ---\ooooo/------\xxxxx/-------
----\oooooo/------\xxxxxx/---     ---o\ooo/o------x\xxx/x-------
----o\oooo/o------x\xxxx/x---     ---oo\o/oo------xx\x/xx-------
----oo\oo/oo------xx\xx/xx---     ---oooxooo------xxxoxxx-------
----ooo\/ooo------xxx\/xxx---     ---oo/o\oo------xx/x\xx-------
----ooo/\ooo------xxx/\xxx---     ---o/ooo\o------x/xxx\x-------
----oo/oo\oo------xx/xx\xx---     ---/oooooox------oxxxxx\-------
----o/oooo\o------x/xxxx\x---     --\ooooo/--------\xxxxx/------

                Cas pair                   Cas impair
```

FIG. 8.5 – *Réalisation d'un signal « carré » ($l = p$) avec cinq états.*

```
-----oox---oxx------           --------ooooo...ooooxo----oxxxx...xxx-ooo--------
-----o-------x------           --------xoooo...oooo-o----o-xxx...xxxxxox--------
-----xx-----oo------           -------ooooo...ooooxo------oxxxx...xxx-ooo-------
----oox-----oxx-----           -------xoooo...oooo-o------o-xxx...xxxxxox-------
----o---------x-----           ------ooooo...ooooxo--------oxxxx...xxx-ooo------
----xx-------oo-----           ------xoooo...oooo-o--------o-xxx...xxxxxox------
---oox-------oxx----
---o-----------x----

Signal carré 3x3                Signal large 2xl (l>3)

- | - o x     o | - o x     x | - o x     - | - o x     o | - o x     x | - o x
--+------     --+------     --+------     --+------     --+------     --+------
- | - - o     - | x o -     - | o ? o     - | - - o     - | - x o     - | ? o x
o | x ? ?     o | x ? -     o | - ? -     o | - o o     o | x o o     o | o - -
x | - ? ?     x | ? ? ?     x | x ? ?     x | o x ?     x | o o o     x | x - x
```

FIG. 8.6 – *Quelques codages optimaux.*

Soit $S \in AC(k)$ (S est construit à partir d'une fonction $f : \mathbb{N} \to \mathbb{N}$ sur un automate cellulaire à k états) et croissant plus vite que EXP_k (c'est-à-dire $(n \mapsto n + k^n) \in o(f)$). Alors il existe une infinité de sites non quiescents dans $Hors(S)$.

Preuve

Considérons la situation du point de vue de la cellule h_0 : le signal EXP_k l'atteint au bout du temps $h_0 + k^{h_0}$, et le signal S au bout du temps $\delta(h_0)$. Pour réaliser S jusqu'à h_0, il faut plus de k^{h_0} configurations différentes.

Un certain nombre de sites de $Hors(S) \cap \{(i, j), i \in \mathbb{N}, j \leq h_0\}$ ont donc été utilisés. Après le passage par la cellule h_0 la situation est similaire : le reste de S croit toujours plus vite que EXP_k, on refait donc le même raisonnement avec une cellule d'indice $n_1 > h_0$. Par induction on constate donc que le signal passe une infinité de fois dans $Hors(EXP_k)$ (*cf.* figure 8.7).

□

Remarque

Ce théorème est la contrepartie du théorème de lacune (8.2). En effet, les signaux S fortement croissants constructibles en laissant $Hors(S)$ entièrement quiescent sont soit le signal vertical $VERT$ (le signal qui reste toujours sur la cellule 0), soit des exponentielles EXP_b. Il y a donc une lacune (un signal du type FACT n'est pas constructible en respectant $Hors(FACT)$ quiescent). Ici, la condition $Hors(S)$ quiescent est l'équivalent du fait qu'une information ne peut pas remonter dans le temps dans le théorème de lacune.

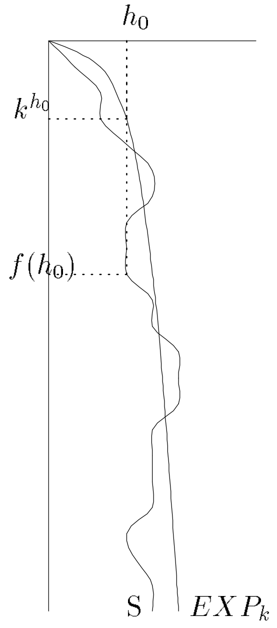


FIG. 8.7 – *Construction pour le théorème des signaux fortement croissants.*

Nous pouvons maintenant étudier en détail la construction des signaux fins qui génère un minorant plus précis dans le cas où $p = l + k^l$ (pour un certain k). Soit n le numéro de la première cellule non quiescente dans la configuration initiale. Il s'agit là encore de mettre en place des signaux auxiliaires pour singulariser les sites (n, p) et $(n + l, p)$. La différence essentielle est que nous utilisons des signaux de type EXP_k (nécessitant donc chacun $2.k - 1$ états). En outre, ces signaux doivent pouvoir se croiser sans interférer (voir figure 8.8) et les frontières du signal élémentaire doivent être particularisées. Au total, il faut donc $(2.k - 1)^2 + \mathcal{O}(1)$ états, ou encore, puisque $k = (p-l)^{1/k}$ et après simplifications, $e(l, p) \leq 4.(p-l)^{2/l} + \mathcal{O}(1)$.

Remarque

Comme indiqué plus haut, cette minoration est valable sous certaines hypothèses, qui correspondent à l'existence de signaux exponentiels appropriés. Dans le cas général p n'est pas de la forme $l + k^l$: il faut ajouter un terme correcteur $c, 0 \leq c < (k + 1)^l - k^l$. Nous ne savons pas encore s'il est possible d'étendre M_2 au cas général.

3.3 Cas des signaux intermédiaires

Ici, on suppose $l < p < l + k^l$ (pour un certain k, par exemple 2). Essayons d'étendre à ce cas les principes de construction précédents, à savoir la mise en place de signaux auxiliaires pour singulariser les sites p et $p + l$. Nous ne sommes pas dans un cas d'application du théorème des signaux fortement croissants (8.6), il pourrait donc exister un certain nombre d'états, suffisant pour réaliser tous les signaux de cette zone (situation similaire à celle des signaux larges). Notre intuition est qu'une telle constante n'existe pas, bien que certaines *familles* de signaux puissent être réalisées avec un petit nombre d'états. L'idée (qui n'est pas une preuve) est que, si les signaux auxiliaires mis en jeu ne sont pas arbitrairement croissants,

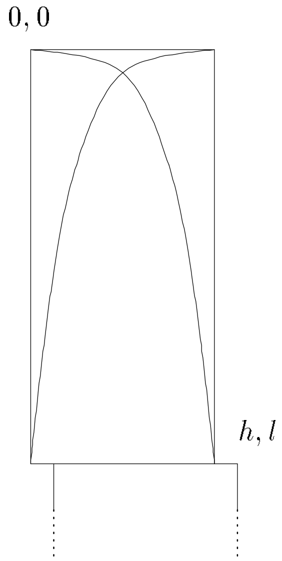


FIG. 8.8 – *Construction d'un signal fin. Les signaux auxiliaires sont des exponentielles.*

ils sont en revanche arbitrairement *complexes*.

Nous décrivons maintenant une famille de signaux élémentaires intermédiaires dont on connaît le comportement : supposons $p = r.k^l$ (pour certains k et r). Alors on peut réaliser le signal élémentaire en juxtaposant des signaux fins, tous construits à partir d'exponentielles (comme vu dans la sous-section précédente). Le coût de la juxtaposition est constant en nombre d'états. On obtient ainsi une majoration du nombre d'états nécessaires : $M_3(l, p) = M_2(l, p) + \mathcal{O}(1)$.

En revanche, rien n'indique qu'il n'existe pas une majoration plus fine et nous n'avons pas encore trouvé de minorant croissant (comme dans le cas des signaux fins).

4 Récapitulatif

Extrêmement puissants, les automates cellulaires restent néanmoins difficiles à programmer ; il serait agréable de disposer d'une « boîte à outils » regroupant divers éléments fréquement utilisés par les programmeurs (d'autres auteurs ont la même démarche, comme B. Martin avec le langage *brick* [Mar94]). À notre sens, les outils les plus simples sont les signaux élémentaires et il peut être utile de minimiser le nombre d'états nécessaires à leur réalisation puisqu'ils sont très fréquement utilisés. La figure 8.9 résume les résultats obtenus dans ce domaine.

Une autre motivation importante est la recherche de solutions synchronisantes de type Minsky utilisant très peu d'états (*cf.* [Min67]). Les solutions les plus rapides de cette famille fonctionnent en temps $3n$ ou $3n + log(n)$ et on sait les implémenter avec 7 états (*cf.* [Yun93, Yun94]). De plus, on sait synchroniser un nombre infini de lignes (mais pas toutes) en n'utilisant 6 états seulement.

Notre intuition est que le signal « carré » de pente 3, codé avec 3 états seulement (aller et retour) peut être utilisé pour construire des solutions économiques. En pratique, l'explosion

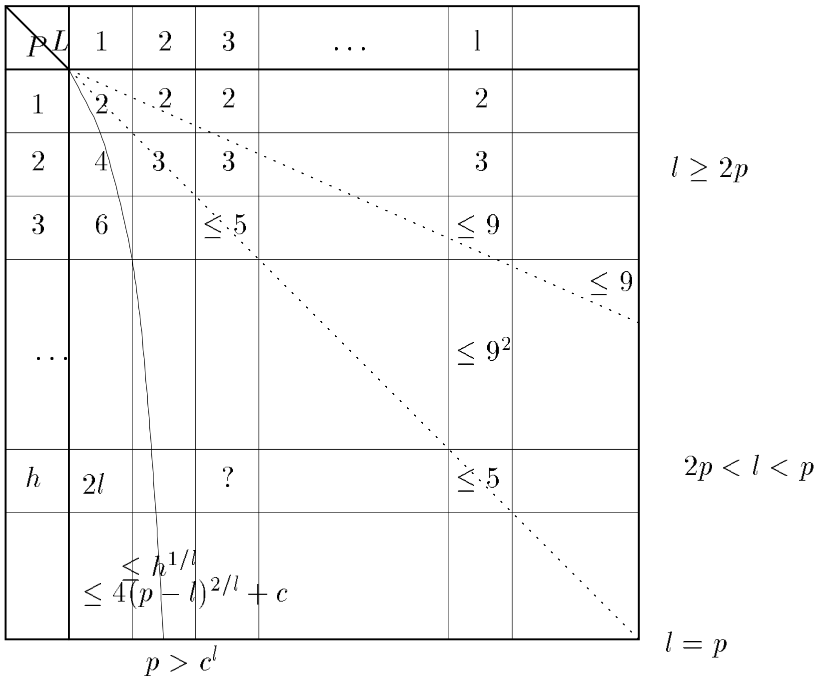


FIG. 8.9 – *Signaux élémentaires : nombre d'états selon l'épaisseur et la hauteur.*

combinatoire et la largeur des signaux (3 cellules) rendent le problème difficile et nous ne savons pas encore s'il est possible de construire une solution en 6 états synchronisant *toutes* les lignes. Un argument encourageant est l'existence de plusieurs solutions en 6 états pour une infinité de lignes, utilisant le signal carré à trois états. La figure 8.10 (page suivante) montre trois diagrammes espace-temps de solutions en temps $3n$:

- à gauche : synchronisation d'une ligne de longueur 13 (tiré de [Yun93]) avec « suffisamment » d'états ;
- au milieu : synchronisation d'une ligne de longueur $2^m + 1$ (ici $m = 4$) avec 6 états et en utilisant des signaux larges ;
- à droite : synchronisation avec deux généraux d'une ligne de longueur $2^m + 1$ (ici $m = 4$) avec 6 états et en utilisant des signaux larges.

```
.-------------
o\------------
o-\-----------
-4-\----------
-5--\---------
-6---\--------
--4---\-------
--5----\------
--6-----\-----
---4-----\----
---5------\---
---6-------\--
----4-------\-
----5--------\
----6--------/
-----4------/-
-----5-----/--
-----6----/---
------4--/----
------5-/-----
------6/------
------..------
-----/..\-----
----/-..-\----
---/-1..4-\---
--/--2--5--\--
-/---3--6---\-
/---1----4---\
\---2----5---/
-\--3----6--/-
--\1------4/--
---.------.---
--/.\----/.\--
-/-.-\--/-.-\-
/-1.4-\/-1.4-\
\-2-5-/\-2-5-/
-\3-6/--\3-6/-
-..-..--..-..-
/..o..\/..o..\
\..0../\..0../
**************

s=13 n=14 t=41
```

```
.----------------
.S---------------
.oS--------------
-xoS-------------
oo-oS------------
oxx-oS-----------
--x--oS----------
-oo---oS---------
-oxx---oS--------
---x----oS-------
--oo-----oS------
--oxx-----oS-----
----x------oS----
---oo-------oS---
---oxx-------oS--
-----x--------oS-
----oo---------oS
----oxx---------S
------x--------Sx
-----oo-------Sx-
-----oxx-----Sx--
-------x----Sx---
------oo---Sx----
------oxx-Sx-----
--------xSx------
-------o.x-------
-------S.S-------
------Sx.oS------
-----Sxo.xoS-----
----Sx-x.o-oS----
---Sx-oo.xx-oS---
--Sx--o-.-x--oS--
-Sx---xx-oo---oS-
Sx---oox-oxx---oS
S----o-----x----S
oS---xx---oo---Sx
-oS-oox---oxx-Sx-
--oSo-------xSx--
---o.x-----o.x---
---S.S-----S.S---
--Sx.oS---Sx.oS--
-Sxo.xoS-Sxo.xoS-
Sx-x.o-oSx-x.o-oS
S-oo.xx-S-oo.xx-S
oSo-.-xSSSo-.-xSx
-o.x-o.x-o.x-o.x-
-S.S-S.S-S.S-S.S-
Sx.oSx.oSx.oSx.oS
So.xSo.xSo.xSo.xS
.................
*****************

s=6 n=17 t=51
```

```
.---------------.
.S-------------S.
.oS-----------Sx.
-xoS---------Sxo-
oo-oS-------Sx-xx
oxx-oS-----Sx-oox
--x--oS---Sx--o--
-oo---oS-Sx---xx-
-oxx---oSx---oox-
---x----S----o---
--oo---SSS---xx--
--oxx-Sx-oS-oox--
----xSx---oSo----
---o.x-----o.x---
---S.S-----S.S---
--Sx.oS---Sx.oS--
-Sxo.xoS-Sxo.xoS-
Sx-x.o-oSx-x.o-oS
S-oo.xx-S-oo.xx-S
oSo-.-xSSSo-.-xSx
-o.x-o.x-o.x-o.x-
-S.S-S.S-S.S-S.S-
Sx.oSx.oSx.oSx.oS
So.xSo.xSo.xSo.xS
.................
*****************

s=6 n=17 t=26
```

FIG. 8.10 – *Vers une solution synchronisante de type Minsky avec 6 états ?*

Chapitre 9

perspectives

La méthode de regroupement a permis de donner une preuve *interne* de théorèmes d'accélération, parfois en association avec des techniques classiques telles que le produit et la mise en séquence (à l'aide d'une synchronisation) d'automates cellulaires.

On peut imaginer de recenser les opérations utiles à la résolution de tels problèmes, de rendre formelles celles qui ne le sont pas encore et s'en servir comme d'un « langage de programmation » des automates cellulaires. Clairement un tel langage ne permettrait pas d'obtenir des programmes minimaux (tels qu'une solution synchronisante du type de celle de Minsky avec 6 états) mais en revanche il autoriserait une écriture limpide de certains algorithmes.

Parmi les opérations à recenser on trouverait probablement :

- le produit usuel de deux automates cellulaires correspondant à l'exécution *en parallèle* et sans interférences des deux machines. Nous proposons la notation $A \parallel B$. Cette opération n'est probablement pas la plus utile en termes de langage de programmation mais elle permet d'en définir d'autres ;

- le produit mixte permettant d'exécuter deux machines simultanément en les faisant interagir de temps en temps, que l'on pourrait noter $A|B$. Cette opération est essentielle et c'est souvent elle que l'on désigne dans la littérature par le terme générique de « produit d'automates cellulaires » ;

- la mise en séquence de deux automates cellulaires à l'aide d'une synchronisation correspondant à l'exécution d'une machine *puis* de l'autre. On pourrait par exemple noter cette opération $A \bullet B$. L'opérateur $\bullet$ peut se définir en termes de produit usuel et de produit mixte ;

- le regroupement, noté $\otimes$ que nous avons défini et utilisé dans les chapitres précédents.

Un tel formalisme permettrait d'exprimer clairement la composition de groupeurs, du type $(M_1 \otimes M_2) \otimes A$, ou certaines simulations telles que celles proposées dans cette thèse.

Par exemple, la méthode d'accélération d'un facteur k des reconnaisseurs décrite par J. Mazoyer et N. Reimen s'exprimerait par :

$$(G \otimes A) \bullet (G' \otimes A)$$

où G est le regroupeur qui construit les configurations k-groupées, G' celui qui simule k pas en 1 et A la machine initiale.

De même, notre méthode d'accélération d'un facteur k des synchroniseurs (chapitre 7) aurait pour expression :

$$((G_0 \otimes A) \parallel (G_1 \otimes A) \parallel \cdots \parallel (G_k \otimes A)) \bullet ((G_0{}' \otimes A) \parallel (G_1{}' \otimes A) \parallel \cdots \parallel (G_k{}' \otimes A))$$

où les G_i sont les regroupeurs fournissant des configurations k-groupées en divers endroits, G' est le regroupeur simulant k pas en un et A est la machine initiale.

Par ailleurs, il doit être intéressant d'explorer les propriétés des opérateurs :

- l'opérateur $\parallel$ est commutatif ;
- l'opérteur $\bullet$ est associatif ;
- il y a des règles de distributivité du type : $(A \parallel B) \bullet C = (A \bullet C) \parallel (A \bullet C)$, $C \bullet (A \parallel B) = (C \bullet A) \parallel (C \bullet B)$.

Une telle approche permettrait probablement d'établir un lien entre la vision du parallélisme sur automates cellulaires et celle, plus classique, qui étudie les propriétés algébriques des systèmes parallèles (réseaux de Petri, algèbres de processus, traces).

Annexe A

Validation des solutions de regroupement

```
/* Utilitaires
 */
dans(X,[X|_]).
dans(X,[_|L]) :-
        dans(X,L).

hors(_,[]).
hors(X,[Y|L]) :-
        X\=Y,
        hors(X,L).

/* Consequences immediates (strictes) d'un ensemble de sites.
 * Exemple : o o o o o   o   (o = donnee, x = resultat)
 *             x x o o o o
 *                   x x
 */
csqi(Sites,Res) :-
        csqi1(Sites,Sites,Res).

csqi1([],_,[]).

csqi1([(Mil,Tmp1)|Sites],Cxt,[(Mil,Tmp2)|Res]) :-
        Ghe is Mil-1, dans((Ghe,Tmp1),Cxt),
        Dte is Mil+1, dans((Dte,Tmp1),Cxt),
        Tmp2 is Tmp1+1,
        hors((Mil,Tmp2),Cxt),
        csqi1(Sites,Cxt,Res).

csqi1([(Mil,Tmp1)|Sites],Cxt,Res) :-
        Ghe is Mil-1, dans((Ghe,Tmp1),Cxt),
        Dte is Mil+1, dans((Dte,Tmp1),Cxt),
```

```
        Tmp2 is Tmp1+1,
        dans((Mil,Tmp2),Cxt),
        csqi1(Sites,Cxt,Res).

csqi1([(Mil,Tmp)|Sites],Cxt,Res) :-
        Ghe is Mil-1, hors((Ghe,Tmp),Cxt),
        csqi1(Sites,Cxt,Res).

csqi1([(Mil,Tmp)|Sites],Cxt,Res) :-
        Dte is Mil+1, hors((Dte,Tmp),Cxt),
        csqi1(Sites,Cxt,Res).

/* Consequences globales (larges) d'un ensemble de sites.
 * Exemple : o o o o o   o   (o = donnee et resulat, xyz = resultat)
 *             x x o o o o
 *               y y x x
 *                 z z
 */
csqg(Sites,Sites) :-
        csqi(Sites,[]).

csqg(Sites,Res) :-
        csqi(Sites,Imm),
        union(Sites,Imm,Uni),
        csqg(Uni,Res).
```

Annexe B

Outils réalisés en Pascal

Recherche systématique de signaux

Lors de mon service national en 1993, j'ai eu l'opportunité d'utiliser un ordinateur Bull DPS 9000 à 4 microprocesseurs. Bien loin des super-calculateurs les plus récents, cette machine présente néanmoins une faculté de traitement importante. Elle reste à ce jour la plus puissante qu'il m'ait été donné de manipuler.

J'ai donc pu essayer une recherche systématique des signaux par un algorithme « brutal ». L'idée initiale était de prouver que trois états (quiescent compris) ne suffisaient pas pour réaliser de petits signaux, ce qui s'est avéré faux puisque le programme a exhibé un signal de pente trois et d'épaisseur 3 (aller et retour) réalisable avec trois états.

L'étude exhaustive montre que le codage est optimal et, fait plus étonnant, qu'il est unique modulo renommages et symétries.

Plus tard j'ai réalisé une version plus astucieuse du programme permettant d'aquérir des résultats en temps raisonnable sur un simple PC (du moins pour les petits signaux, la complexité globale restant désastreuse).

La syntaxe est la suivante :

```
Signal largeur-du-signal hauteur-du-signal nombre-d'etats
       [/Rapport] [/Progression] [/Discret]
```

Exemple : `signal 4 2 3 /Rapport` recherche tous les signaux aller/retour de largeur 4, de pente 2 et utilisant 3 états au plus (dont l'état quiescent) et affiche tous les résultats trouvées.

Résultat de l'exemple précédent :

```
__ooo+__    __o+o___          __o+o+__    __oo____
___o_o__    __++_+__          ___o_o__    __+++___
 ___o+o+_    _oo_____         ___ooo+_    _o+o____

 __o+o+__    __oo_+__         __ooo+__    __o+++__
 ___o_o__    __+++___         ___o_o__    __++_+__
 ___o+o+_    _oo_+___         ___ooo+_    _o+++___

 __o+o+__    __oo++__         __o+o+__    ___+++__
 ___o_o__    __++_+__         ___o_o__    __oo____
 ___o+o+_    _oo++___         ___o+o+_    __+++___
```

Table des figures

Bibliographie

[AB95] P. Aigrain and D. Beauquier. Polyomino tilings, cellular automata and codicity. *Theoretical Computer Science*, 147(1–2):165–180, 1995.

[AČ87] J. Albert and K. Čulik. A simple universal cellular automaton and its one-way and totatlistic version. *Complex Systems*, 1:1–16, 1987.

[Bal67a] R. Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, 10(1):22–42, 1967.

[Bal67b] R. Balzer. Search for a solution: A case study. Technical report, RAND Corporation, 1967.

[Bar75] A. G. Barto. *Cellular Automata as Models of Natural Systems*. PhD thesis, The University of Michigan, 1975.

[Boy95] T. Boykett. Combinatorial construction of one dimensional reversible cellular automata. *Contrubutions to General Algebra*, 9:81–90, 1995.

[But74] J. T. Butler. A note on cellular automata simulations. *Information and Control*, 26(3):286–295, 1974.

[CČ83] C. Choffrut and K. Čulik. On real-time cellular automata and treillis automata. *Information Processing Letters*, 17:149–153, 1983.

[CČ84] C. Choffrut and K. Čulik. Folding of the plane and the design of systolic arrays. *Acta Informatica*, 21:393–407, 1984.

[ČK95] K. Čulik and J. Kari. An aperiodic set of wang cubes. *Journal of Universal Computer Science*, 1(10):675–686, 1995.

[Col69] S. N. Cole. Real-time computation by n-dimensional arrays of finite-state machines. *IEEE Transactions on Computers*, C(18):388–394, 1969.

[Čul89] K. Čulik. Variation of the firing squad problem and applications. *Information Processing Letters*, 30:153–157, 1989.

[DM94] M. Delorme and J. Mazoyer. Reconnaissance de langages sur automates cellulaires. Rapport interne 94-46, LIP, ENS Lyon, 1994.

[Dub93] J. C. Dubacq. How to simulate non-reversible turing machines by one-dimensional cellular automata, 1993.

[Dur93] B. Durand. Undecidability of the surjectivity problem for 2d cellular automata: a simplified proof. In *Lecture Notes in Computer Science*, 1993.

[Dur96a] J. O. Durand-Lose. *Automates Cellulaires, Automates à Partitions et Tas de Sable.* Thèse de doctorat, Université Bordeaux I, 1996.

[Dur96b] J. O. Durand-Lose. Simulation espace-temps réversible d'un automate cellulaire non-réversible, 1996.

[Fis65] P. C. Fisher. Generation of primes by a one-dimensional real-time iterative array. *Journal of the ACM*, 12(3):388–394, 1965.

[Gef93] W. Geffert. A speed-up theorem without tape compression. *Theoretical Computer Science*, 118, 1993.

[GMM93] E. Goles, A. Maass, and S. Martinez. On the limit set of some universal cellular automata. *Theoretical Computer Science*, 110:53–78, 1993.

[Gra75] A. Grasselli. Synchronization of cellular arrays: The firing squad problem in two dimensions. *Information and Control*, 28:113–124, 1975.

[Gut] H. Gutowitz. Frequently asked questions about cellular automata. http://alife.santafe.edu/alife/topics/cas/ca-faq/ca-faq.html.

[Gut91] H. Gutowitz. *Cellular Automata : Theory and Experimentation.* MIT/North Holland, 1991.

[Hee92] O. Heen. Nombre de communications dans les solutions synchronisantes, rapport de DEA (non publié), 1992.

[Hee95a] O. Heen. Accélération d'une constante sur automates cellulaires et applications. Rapport LITP 49, Institut Blaise Pascal, 1995.

[Hee95b] O. Heen. Linear speed-up without synchronization. Rapport LITP 57, Institut Blaise Pascal, 1995.

[Hee96a] O. Heen. Counting with few states, *in progress*, 1996.

[Hee96b] O. Heen. Linear speed-up for cellular automata synchronizers and applications. *Theoretical Computer Science*, 1996. à paraître.

[HN75] M. Harao and S. Noguchi. Fault tolerant cellular automata. *Journal of Computers and System Sciences*, 11(2):171–185, 1975.

[IJ87] O. H. Ibarra and T. Jiang. On one-way cellular arrays. *SIAM Journal of Computation*, 16(6):1135–1154, 1987.

[IJ89] O. H. Ibarra and T. Jiang. Relating the power of cellular automata arrays to their closure properties. *Theoretical Computer Science*, 57:225–238, 1989.

[IKM85] O. H. Ibarra, S. M. Kim, and S. Moran. Sequential machine caracterization of treillis and cellular automata and application. *SIAM Journal of Computation*, 14:426–427, 1985.

[IPK85] O. H. Ibarra, M. A. Palis, and S. M. Kim. Fast parallel language recognition by cellular automata. *Theoretical Computer Science*, 41:231–246, 1985.

[Kar90] J. Kari. Reversibility of 2D CA is undecidable. *Physica D*, 45:379, 1990.

[Knu73] D. E. Knuth. *The Art of Computer Programming.* Addison-Wesley, 1973.

[KV95] M. Kutrib and R. Vollmar. The firing squad synchronization problem in defective cellular automata. *IEICE Transactions on Information and Systems*, E78-D(7):409–440, 1995.

[LN88] K. Lindgren and M. Nordahl. Complexity measures in cellular automata. *Complex Systems*, 2:409–440, 1988.

[LN90] K. Lindgren and M. Nordahl. Universal computation in simple one dimensional cellular automata. *Complex Systems*, 4:299–318, 1990.

[Mar84] N. Margolus. Physique like models of computation. *Physica D*, 10:81–95, 1984.

[Mar92] B. Martin. Embedding a finite bounded cellular torus into a ring of automata. Technical report, Laboratoire d'informatique, Signaux et Systèmes de Sophia-Antipolis, 1992.

[Mar94] B. Martin. A universal cellular automaton in quasi-linear time and its s-m-n form. *Theoretical Computer Science*, 123:199–237, 1994.

[Maz87] J. Mazoyer. A six states minimal time solution to the firing squad synchronisation problem. *Theoretical Computer Science*, 50:183–237, 1987.

[Maz90] J. Mazoyer. Solutions au problème de la synchronisation d'une ligne de fusiliers : étude de leur structure. Rapport d'habilitation, 1990.

[Maz92] J. Mazoyer. Entrées et sorties sur une ligne d'automates. In M. Cosnard M. Nivat Y. Robert, editor, *Algorithmique parallèle*, pages 47–61. Masson, 1992.

[Min67] M. Minsky. *Computation, Finite and Infinite Machines.* Prentice-Hall, Englewood, CA, 1967.

[ML68] F. R. Moore and G. G. Langdon. A generalized firing squad problem. *Information and Control*, 12(3):212–220, 1968.

[Moo64] E. F. Moore. *Sequential Machines, Selected Papers.* Addison-Wesley, Reading, Mass., 1964.

[Moo95] C. Moore. Non-abelian cellular automata. Technical Report Working Paper 95-09-080, Santa Fe Institute, 1995.

[Moo96] C. Moore. Quasi-linear cellular automata. Technical report, Santa Fe Institute, 1996.

[Mor95] K. Morita. Reversible simulation of one-dimensional irreversible cellular automata. *Theoretical Computer Science*, 148:157–163, 1995.

[MR92] J. Mazoyer and N. Reimen. A linear speed-up theorem for cellular automata. *Theoretical Computer Science*, 101:59–98, 1992.

[MT93] J. Mazoyer and V. Terrier. Signals in one dimensional cellular automata, 1993.

[Pap94] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[Ped82] J. Pederson. Cellular automata as algebraic systems. *Complex Systems*, 6:237–250, 1982.

[PU87] C. H. Papadimitriou and Jeffrey D. Ullman. A communication-time tradeoff. *SIAM Journal on Computing*, 16(4):639–646, 1987.

[Rei93] N. Reimen. *Contribution à l'étude des automates cellulaires.* Thèse de doctorat, Université Paris 7, 1993.

[Rob87] A. D. Robinson. Fast computation of additive cellular automata. *Complex Systems*, 1:211–216, 1987.

[Rom76] F. Romani. Cellular automata synchronization. *Information Sciences*, 10:299–318, 1976.

[Smi71a] A. R. Smith III. Cellular automata complexity trade-offs. *Information and Control*, 18(5):466–482, 1971.

[Smi71b] A. R. Smith III. Simple computation-universal cellular spaces. *Journal of the ACM*, 18(3):339–353, 1971.

[Smi72] A. R. Smith III. Real-time language recognition by one-dimensional cellular automata. *Journal of Computation and System Sciences*, 6(3):233–253, 1972.

[Sut95] K. Sutner. On the computational complexity of finite cellular automata. *Journal of Computers and System Sciences*, 50(1):87–97, 1995.

[Ter91] V. Terrier. *Temps réel sur automates cellulaires.* Thèse de doctorat, ENS Lyon, 1991.

[Ter95] V. Terrier. One real time one-way cellular array. *Theoretical Computer Science*, 141:331–335, 1995.

[Ter96] V. Terrier. Language not recognizable in real time by one-way cellular automata. *Theoretical Computer Science*, 156(1–2):281–287, 1996.

[TM90] T. Toffoli and N. Margolus. Invertible cellular automata: A review. *Physica D*, 45:229, 1990.

[Ume93] H. Umeo. A fault-tolerant scheme for optimum-time firing squad synchronization. In *ParCo 93*, 1993.

[Vis84] G. Vishniac. Simulating physics with cellular automata. *Physica D*, 10:96–115, 1984.

[Wak66] A. Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66–78, 1966.

[Wol84a] S. Wolfram. Computation theory of cellular automata. *Communication in Mathematical Physics*, 96:15–57, 1984.

[Wol84b] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[Yun93] J. B. Yunès. *Synchronisation et automates cellulaires: la ligne de fusiliers.* Thèse de doctorat, Université Paris 7, 1993.

[Yun94] J. B. Yunès. Seven-state solutions to the firing squad synchronization problem. *Theoretical Computer Science*, 127:313–332, 1994.

[Yun96] J. B. Yunès. Fault tolerant solutions to the firing squad synchronization problem. Rapport LITP 06, Institut Blaise Pascal, 1996.

Saving ressources on cellular automata

Olivier Heen

Abstract

A cellular automaton is an infinite collection of computing units : the cells. They are all similar and regularly interconnected. It is amazing to observe how such a simple device can modelize complex phenomena. And indeed cellular automata are of great interest in computer science, theoretical physics and chemistry, and even in biology.

In this work we focus on the computational aspects of one dimensional cellular automata. We restate important complexity results with special care to the amount of needed ressources. More precisely, we try to characterize the minimal size of cells in order to perform particular tasks, and give an answer to the following questions :

- What extra ressources are needed to save k steps of computation? Or to compute k times faster?
- What is the « price » for sending information from cells to cells?
- What are the inauthorized synchronization times?

The systematical study of such questions leads to a new method for simulating cellular automata. This method emphizes the intrinsic aspects of some properties : results can be obtained « from within » the theory of cellular automata thanks to a relatively simple product of machines (the so-called grouping product).

Key-words

One-Dimensional Cellular Automata, Complexity, Linear Speed-Up, Simulation, Computational Geometry, Massive Parallelism, Synchronization.

Économie de ressources sur automates cellulaires

Olivier Heen

Résumé

Un automate cellulaire est un assemblage régulier et potentiellement infini d'unités de calcul toutes semblables : les cellules. Il est étonnant de constater la complexité des phénomènes qu'une machine aussi simple permet de modéliser. De fait, les automates cellulaires trouvent des applications variées en informatique, mais aussi en physique et chimie théoriques et même en biologie.

Ce travail traite de l'aspect informatique des automates cellulaires unidimensionnels, qui forment un modèle intéressant du calcul parallèle. Nous reformulons des résultats de complexité dans une optique d'économie des ressources. Plus précisément, nous cherchons à caractériser la taille minimale des cellules pour réaliser certaines tâches et répondons à quelques questions :

- Dans quelles proportions faut-il accroître les ressources pour gagner k pas de calcul ? Pour calculer k fois plus vite ?
- Quel est le « prix » à payer pour envoyer de l'information de cellule en cellule ?
- Quels sont les temps de synchronisation autorisés ?

L'étude systématique de telles questions a conduit à la mise au point d'une nouvelle méthode de simulation d'automates cellulaires mettant en valeur le caractère intrinsèque de certaines propriétés : les résultats s'obtiennent « à l'intérieur » de la théorie des automates cellulaires en termes de produits relativement simples de machines (produit de regroupement).

Mots-clés

Automates Cellulaires Unidimensionnels, Complexité, Accélération Linéaire, Simulation, Géométrie Discrète, Parallélisme Massif, Synchronisation.