Simulations d'automates cellulaires

Bruno Martin

Année 2005

HABILITATION À DIRIGER DES RECHERCHES

PRÉSENTÉE À :

L'UNIVERSITÉ DE NICE – SOPHIA ANTIPOLIS

École Doctorale STIC

SPÉCIALITÉ : INFORMATIQUE

PAR

Bruno MARTIN

Simulations d'automates cellulaires

Soutenue le 8 avril 2005 devant la commission d'examen composée de :

Pr.	I. Bond	
Pr.	R. Cori	*Rapporteur*
Pr.	E. Formenti	
Pr.	I. Guessarian	*Rapporteur*
Pr.	J. Mazoyer	
Pr.	L. Vuillon	*Rapporteur*

Préface

Ce document reprend la plupart des travaux de recherche publiés après ma thèse de troisième cycle [**30**]. J'ai volontairement omis les publications [**7, 21**] écrites en collaboration car elles ne suivent pas le fil directeur de la simulation d'automates cellulaires. Comme en atteste le titre de cette thèse, je me concentre sur des résultats de simulation de deux types ; dans la première partie je simule le fonctionnement d'automates cellulaires par d'autres modèles de calcul et dans la seconde, je m'intéresse à la simulation d'automates cellulaires définis sur des graphes de Cayley en changeant le graphe de connexion des cellules.

Dans les chapitres 2 et 3, j'introduis les automates cellulaires comme un modèle classique de calcul parallèle. J'ai en effet montré dans [**33, 35**] que les automates cellulaires forment ce que l'on nomme un système de programmation acceptable en calculabilité. Je m'intéresse dans le chapitre 5 à la simulation des automates cellulaires par différents types de PRAM (CREW-PRAM et XPRAM). J'y regroupe les résultats publiés dans les actes de MFCS'1993 [**31**] et de SCI-ISAS'1998 [**37**]. Dans le chapitre 6, je simule le fonctionnement des automates cellulaires par des machines spatiales, un modèle de calcul introduit par Y. Feldman et E. Shapiro dans les années 1990. Les machines spatiales permettent de définir aussi bien un modèle de calcul séquentiel qu'un modèle de calcul parallèle. J'ai publié la simulation du fonctionnement d'un automate cellulaire par une machine spatiale séquentielle dans un article de TSI en 1995 [**34**] après l'avoir présenté à la conférence RenPar'1994 [**32**]. J'ai ensuite montré l'existence d'une famille de machines spatiales parallèles qui mime le fonctionnement d'un automate cellulaire à EuroPar'1999 [**38**]. Le chapitre 7 résume ma contribution à la complexité des modèles et résume quelques conséquences de ces résultats de simulation.

Dans la seconde partie, je définis dans le chapitre 8 les automates cellulaires sur des graphes de Cayley . J'illustre ces définitions en rappelant quelques résultats de Zs. Róka qui assurent la simulation d'un réseau hexagonal d'automates par des tores d'automates. Le chapitre 9 présente deux simulations d'un tore d'automates par un anneau d'automates. La première était ébauchée dans ma thèse de troisième cycle [**30**] et a fait l'objet d'une publication dans la revue IJFCS en 1997 [**36**]. La deuxième simulation, obtenue en collaboration avec C. Peyrat [**41**], améliore le résultat précédent dans certains cas et est actuellement soumise à publication. En combinant mes résultats avec ceux de Zs. Róka, j'ai également montré comment simuler un réseau hexagonal d'automates par un anneau d'automates dans un article de TCS en 2001 [**39**].

Enfin, en parallèle de mes travaux de recherche, j'ai publié en 2004 un ouvrage intitulé «Codage, cryptologie et applications» [**40**]. J'y rassemble le contenu de plusieurs des enseignements de troisième cycle que je dispense à l'Université de Nice Sophia-Antipolis.

Remerciements

Merci tout d'abord à mes rapporteurs : I. Guessarian pour ses remarques pertinentes, R. Cori pour son rapport chaleureux et L. Vuillon pour son enthousiasme communicatif. Je remercie I. Bond –qui m'incite à écrire cette habilitation depuis longtemps–, J. Mazoyer et E. Formenti, les deux spécialistes des automates cellulaires, d'avoir accepté de faire partie de ce jury. Merci à tous de vous être libérés le temps d'une soutenance.

Merci à C. Peyrat et P. Solé, mes co-auteurs du projet Recif, qui m'ont permis de mener à bien certains travaux. Merci également aux collègues de Sophia, de Nice, de France et de l'étranger. Grâce à E. Grandjean et M. Santha et aux différents GDR et PRC qu'ils ont animé, j'ai pu découvrir de nouveaux domaines de l'informatique. Je suis reconnaissant à A. Petit de son intérêt et de sa confiance qui ont été constants depuis dix ans. Enfin, merci à J. Gruska ; s'il n'a pas (encore) réussi à me convertir au calcul quantique, il m'a poussé à écrire un livre.

Je suis redevable aux étudiants qui me permettent d'étendre mes connaissances, tant du coté du système `unix` que de celui des courbes elliptiques.

Je dois beaucoup à mes amis pour toutes nos discussions. Merci également à ceux qui étaient là pour recevoir mes coups de shinaï et à ceux avec lesquels ramer est un plaisir.

Sandrine, un grand merci pour notre complicité…

Table des matières

Première partie

Complexité des modèles

CHAPITRE 1

Introduction

Dans cette première partie, je m'intéresse à ce que A. Borodin nomme la complexité des modèles. C'est ainsi qu'il a qualifié mon travail alors qu'il venait à Sophia-Antipolis pour le colloque de complexité algébrique à la mémoire de J. Morgenstern en 1995.

Dans un premier temps, je rappelle quelques définitions sur les automates cellulaires de dimension un. Il s'agit du modèle de calcul que je prendrai ensuite comme modèle de référence. Je rappelle brièvement l'existence de mon automate cellulaire intrinsèquement universel [**29, 30, 33**] et je rappelle quelques faits [**35**] permettant d'utiliser les automates cellulaires comme système de programmation acceptable, une notion introduite par H. Rogers [**50**] et reprise dans l'ouvrage de M. Machtey et P. Young [**27**].

Je présente ensuite mon résultat de simulation d'un automate cellulaire par une PRAM [**31**]. Je m'en suis inspiré pour concevoir la simulation d'un automate cellulaire par une XPRAM [**37**], un modèle du parallélisme plus réaliste introduit par Valiant [**57**].

Enfin, je m'intéresse à la simulation des automates cellulaires par des machines spatiales. Ce modèle de calcul proposé par Y. Feldman et E. Shapiro [**15**] permet de définir aussi bien des machines séquentielles que parallèles. Je montre qu'il est possible de simuler le fonctionnement d'un automate cellulaire arbitraire par une machine spatiale séquentielle [**32, 34**]. Je décris ensuite le fonctionnement d'une famille de machines spatiales parallèles pour simuler le fonctionnement d'un automate cellulaire [**38**].

Ces résultats de simulation d'automates cellulaires par d'autres modèles de calcul conduisent directement à la construction de machines universelles. Je propose plusieurs notions d'universalité publiées dans [**35**] ; elles sont toutes équivalentes du point de vue du calcul, mais prennent en compte les différentes manières de simuler le fonctionnement d'une autre machine.

CHAPITRE 2

Modèles de calcul classiques

Nous rappelons les définitions usuelles de théorie des langages et nous présentons deux modèles de calcul : les machines de Turing qui constituent un des principaux modèles de référence pour le calcul séquentiel et celui des automates cellulaires qui, pour nous, est un des principaux modèles du calcul parallèle. Nous définissons également la notion de système de programmation que nous utiliserons beaucoup dans la suite.

1. Alphabets, mots, langages

Soit Σ un alphabet, c'est à dire un ensemble fini non vide de symboles, comme par exemple l'alphabet binaire $\{0,1\}$.

Un mot fini est une suite finie de symboles sur cet alphabet, par exemple 01101 est un mot sur $\{0,1\}$.

$\Sigma^\star$ représente l'ensemble des mots finis sur l'alphabet Σ.

Muni d'une opération associative la concaténation notée $.$ et d'un élément neutre ε, $(\Sigma^\star,.)$ est un monoïde.

On définit un langage comme un sous-ensemble de $\Sigma^\star$. On a quelques langages particuliers : le langage vide, $L = \varnothing = \{\}$ différent du langage du mot vide $L = \{\varepsilon\}$. Un langage peut être soit fini comme $L = \{0110, 1001\}$, soit infini dénombrable comme le langage des mots binaires pairs.

2. Machines de Turing

Introduit par A. Turing en 1938, le modèle des machines de Turing est le plus répandu et le mieux connu. Nous en rappelons la définition [**20, 46**] :

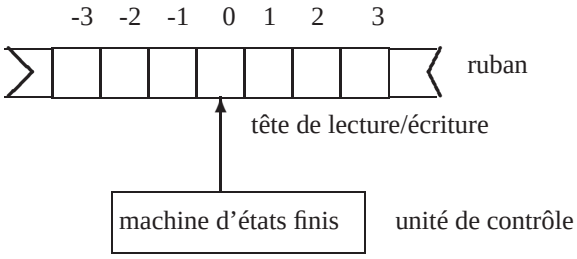

Fig. 1. Une machine de Turing.

Définition 2.1. Une machine de Turing est la donnée de $M = \langle \Sigma, Q, \delta, I, F \rangle$ où
- Σ est un alphabet fini muni d'un symbole particulier, le blanc, noté B ;
- Q est un ensemble fini, l'ensemble des états ;
- δ est la fonction de transition de la machine :

$$\delta : Q \times \Sigma \to Q \times \Sigma \times \{\text{gauche,0,droite}\}$$

- $I \subseteq Q$ est l'ensemble des états initiaux ;
- $F \subseteq Q$ est l'ensemble des états terminaux.

La machine de Turing est composée de deux éléments,
- le ruban sur lequel est écrit la configuration initiale de la machine, i.e. un mot de $\Sigma^\star$.
- une unité de contrôle qui rend compte de l'état de la machine qui comporte une tête de lecture/écriture qui lui permet de lire ou d'écrire un caractère du mot inscrit sur le ruban.

La fonction de transition s'interprète de la façon suivante : M étant dans l'état $q \in Q$ lit un caractère $\alpha \in \Sigma$ sur le ruban et, selon la fonction de transition δ, change l'état de la machine, remplace le caractère α par un caractère β et déplace sa tête de lecture d'un caractère vers la gauche ou vers la droite (ou ne la bouge pas).

On définit également la notion de description instantanée d'une machine qui recense à la fois la configuration courante de la machine de Turing, son état courant et la position de sa tête de lecture/écriture. Cette notion est formalisée par exemple dans [**27**].

3. Automates cellulaires

Le modèle des automates cellulaires a été introduit en 1948 par J. von Neumann pour des considérations qui relevaient plus de la biologie que de l'informatique. Il s'agissait de construire un modèle mathématique de la reproduction de cellules, d'où le nom de l'automate cellulaire conçu par J. von Neumann : le jeu de la vie. Selon son créateur, un automate cellulaire bidimensionnel est composé d'un ensemble de cellules identiques disposées sur une grille bi-infinie.

Nous nous restreignons au modèle unidimensionnel des automates cellulaires, pour lequel on considère un ensemble de cellules identiques indicées par $\mathbb{Z}$ ou par $\mathbb{N}$. Chaque cellule est une machine pourvue d'un nombre fini d'états. Les cellules évoluent parallèlement et de façon synchrone suivant des intervalles de temps discrets selon une fonction de transition. Cette fonction de transition attribue à chaque cellule un nouvel état en fonction de son propre état et des états des cellules voisines à l'instant précédent. La fonction de transition est identique pour toutes les cellules qui évoluent de manière uniforme. Plus formellement,

Définition 2.2. Un automate cellulaire est un tableau de cellules identiques indicées par $\mathbb{Z}$. Chaque cellule est une machine à états finis identique $C = (Q, \delta)$ où
- Q est l'ensemble fini des états ;
- δ est une application $Q \times Q \times Q \to Q$ appelée fonction de transition ;

La fonction de transition permet d'attribuer un nouvel état à chaque cellule (fig 2). Si on note $c(i,t)$ l'état de la cellule i au temps t,

$$c(i,t+1) = \delta(c(i-1,t), c(i,t), c(i+1,t))$$

On distingue un état particulier, appelé quiescent et noté généralement q qui vérifie la propriété : $\delta(q,q,q) = q$.

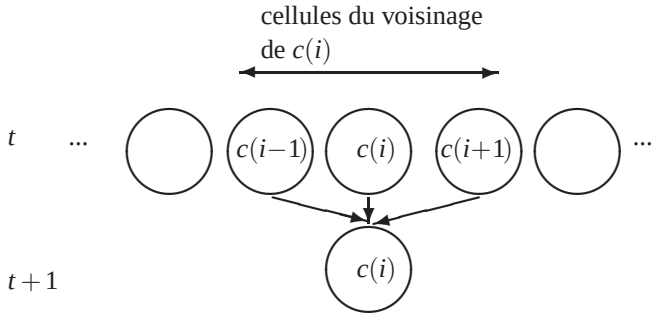

Fig. 2. Transition d'une cellule.

Pour un instant t fixé, la suite $c(i)$ des états des cellules pour $i \in \mathbb{Z}$ est appelée la configuration de l'automate cellulaire au temps t.

Définition 2.3. On distingue ici deux types de configurations :
– la configuration «générale», une application $c : \mathbb{Z} \to Q$ qui associe un état à toute cellule.
– la configuration finie, une application $c : \mathbb{Z} \to Q$ qui attribue un état à toute cellule de telle sorte que la partie non quiescente de l'automate cellulaire soit finie et connexe.

Si on note $\mathbb{C}$ l'ensemble de toutes les configurations possibles d'un automate cellulaire, on peut alors définir une fonction de transition globale :

Définition 2.4. Pour un automate cellulaire donné $C = (Q, \delta)$ d'ensemble de configurations $\mathbb{C}$, la fonction de transition globale Δ est définie au moyen de la fonction de transition locale δ par :

$$\forall c \in \mathbb{C}, \forall i \in \mathbb{Z}, \Delta(c)(i) = \delta\left(c(i-1), c(i), c(i+1)\right)$$

Remarquons que dans la définition 2.2 aucune hypothèse n'est faite sur l'ensemble des états. En particulier, cet ensemble ne possède aucune structure. Nous introduisons dans la définition 2.5 la notion d'automate cellulaire totalistique qui donne une structure simple à l'ensemble des états.

Définition 2.5 (Albert et Culik [**1**])**.** Un automate cellulaire $C = (Q, \delta)$ est dit totalistique si $Q \subset \mathbb{N}$ et s'il existe une fonction $\gamma : \mathbb{N} \to Q$ telle que $\forall a, b, c \in Q,\ \delta(a, b, c) = \gamma(a + b + c)$.

On a introduit une notion d'ordre sur les états qui permet de définir la fonction de transition comme faisant abstraction de la position relative des cellules entre elles. Contrairement aux apparences, un automate cellulaire peut reconnaitre l'ordre sur les cellules (sa droite et sa gauche) comme le prouve le lemme 2.1. (Observons que l'ensembles des états de l'automate cellulaire Q est une partie de $\mathbb{N}$ mais pas nécessairement un segment initial de $\mathbb{N}$).

Lemme 2.1 (Albert et Culik [**1**])**.** Pour tout automate cellulaire, il existe un automate cellulaire totalistique qui le simule sans perte de temps et qui n'a, au plus, que quatre fois plus d'états.

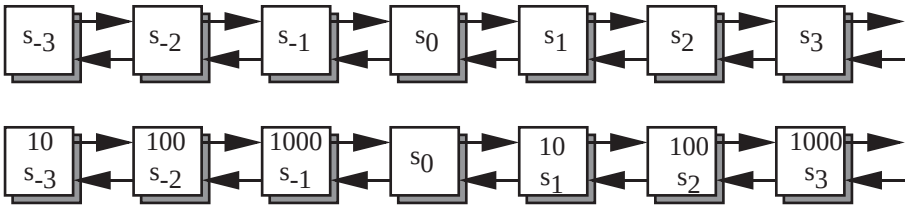

Fig. 3. Une configuration de C et sa version totalistique.

Preuve. On considère un automate celulaire $C = (Q, \delta)$ à n états. On construit $C' = (Q', \gamma)$ le totalistique qui simule C. On construit tout d'abord Q' à partir de Q comme : $Q' = \{s \cdot m : s \in Q, m \in \{1, 10, 100, 1000\}\}$. Les états de Q' peuvent être vus comme des entiers en base b où $b = n + 1$. Avec cette construction, $\sharp Q' = 4 \cdot \sharp Q$. On construit ensuite γ à partir de δ :

$$\begin{array}{lll}
\gamma(zyx) & = 10.\delta(x,y,z) & = \delta(x,y,z)0 \\
\gamma(zyx0) & = 100.\delta(x,y,z) & = \delta(x,y,z)00 \\
\gamma(yx0z) & = 1000.\delta(x,y,z) & = \delta(x,y,z)000 \\
\gamma(x0zy) & = \delta(x,y,z) & = \delta(x,y,z)
\end{array}$$

pour x, y et z des états de Q et xyz, $xyz0$, $yz0x$, $z0xy$, $1, 10, 100, 1000$, $\delta(x,y,z)$, $\delta(x,y,z)0$, $\delta(x,y,z)00$ et $\delta(x,y,z)000$ des entiers en base b. On observe alors que C' simule convenablement le fonctionnement de C sans perte de temps si on transforme la configuration initiale de C en celle décrite par la figure 3. □

Notons que la définition 2.5 va nous permettre d'établir une numérotation de Gödel des automates cellulaires dans le chapitre 3, section 3.1.

Une autre variante de la définition 2.2 revient à considérer l'ensemble des états Q comme le produit cartésien fini de plusieurs ensembles (finis) d'états : $Q = \prod_{i=1}^{n} Q_i$. Ainsi, il est alors possible de définir des automates cellulaires produits, comme dans le cas des automates finis. Remarquons que, dans ce cas, l'évolution de chaque automate cellulaire composant le produit est indépendante de toute autre évolution. Nous nous intéresserons plutôt à la définition suivante :

Définition 2.6. On appelle automate cellulaire uplet $D = (Q, \delta)$ tout automate cellulaire dont l'ensemble des états $Q = \prod_{i=1}^{n} Q_i$ est le produit cartésien fini d'ensembles finis et dont la fonction de transition $\delta : Q \times Q \times Q \to Q$ est telle que

$$\underline{\alpha} = \delta(\underline{x}, \underline{y}, \underline{z})$$

où $\underline{x}, \underline{y}, \underline{z}$ et $\underline{\alpha}$ sont des n-uplets.

Dans la définition 2.6, les différents ensembles d'états seront aussi appelés des calques.

Il est clair que les définitions 2.2 et 2.6 sont équivalentes. L'intérêt «pratique» de la définition 2.6 provient du fait que l'action de l'automate cellulaire sur chacun des calques peut être indépendante sauf en de rares exceptions. Donc, un automate cellulaire uplet pourra être décrit comme plusieurs automates cellulaires simples avec des interactions possibles entre les calques.

4. Systèmes de programmation

Nous rappelons la définition d'un système de programmation acceptable [**27**].

Définition 2.7. Un système de programmation est une liste $\varphi_0, \varphi_1, \ldots$ qui contient toutes les fonctions récursives partielles (d'un argument sur $\mathbb{N}$). Un système de programmation est appelé système de programmation universel si la fonction partielle φ_{univ} telle que $\varphi_{univ}(i,x) = \varphi_i(x)$ pour tout i et tout x est elle-même une fonction récursive partielle. En d'autres termes, le système doit avoir une fonction universelle récursive partielle. Un système de programmation universel est appelé système de programmation acceptable s'il existe une fonction récursive totale c pour la composition telle que $\varphi_{c(i,j)} = \varphi_i \circ \varphi_j$ pour tout i et j.

Les systèmes de programmation sont souvent appelés énumération ou numérotation de Gödel des fonctions partielles récursives. Il est connu [**27**] qu'il existe des systèmes de programmation acceptables et qu'ils sont tous isomorphes entre eux par l'isomorphisme de Rogers [**50**] (en d'autres termes, il existe une fonction f qui permet de traduire des programmes d'un système de programmation acceptable dans un autre).

Théorème 2.1 (Isomorphisme de Rogers)**.** Soit $\varphi_0, \varphi_1, \ldots$ et $\psi_0, \psi_1, \ldots$, deux systèmes de programmation acceptables. Alors il existe une fonction récursive totale injective f telle que $\varphi_x = \psi_{f(x)}$.

Le théorème 2.1 est très important ; il affirme qu'il est possible de traduire tout système de programmation en un autre. Mais il n'est pas possible de borner la complexité d'une telle fonction, ni d'exprimer une quelconque opinion sur l'efficacité de la fonction obtenue.

CHAPITRE 3

Quelques résultats sur les automates cellulaires

1. Introduction

Dans ce chapitre, nous rappelons quelques résultats sur les automates cellulaires. Nous commençons par quelques résultats de synchronisation qui sont fort utilisés dans la suite. Nous rappelons ensuite certains résultats permettant de démontrer que les automates cellulaires forment un système de programmation acceptable.

2. Synchronisation d'une ligne de fusiliers

Nous présentons ici quelques résultats sur la synchronisation d'une ligne d'automates. Ce problème, connu sous le nom de synchronisation d'une ligne de fusilliers, a été proposé par J. Myhill en 1957. En voici une formulation :

«Étant donnée une ligne initiale de fusilliers comprenant un général à une des extrémités de la ligne, comment les fusilliers peuvent-ils tirer au même instant en sachant que l'ordre de feu provient du général et met un certain temps pour se propager ?»

Chaque fusilier peut être représenté par une cellule d'un automate cellulaire. Le problème revient alors à construire la fonction de transition. La première solution à ce problème a été donnée en 1965 par M. Minsky et J. MacCarthy en découpant récursivement en deux la ligne d'automates jusqu'à ce qu'on arrive à une seule cellule. La ligne d'automates est alors synchronisée (cf. figure 1). Cette solution en temps $3 \cdot n - 1$ au problème de la synchronisation d'une ligne de n automates n'est pas optimale.

Lemme 3.1 (Lemme de synchronisation)**.** Il existe un automate cellulaire $Z = (Q, \delta)$ comprenant des états particuliers, g,$F \in Q$ et un état quiescent q tel que :

$$\Delta^t(q0^{n-1}gq) = qF^nq$$

pour un temps $t = 3 \cdot n - 1$ et de telle sorte que l'état F n'apparaisse pas avant le temps t, c'est à dire $\Delta(q0^{n-1}gq)(i,t) \neq F$ pour $0 \leq t < 3.n - 1$. Ci-dessus, Δ représente la fonction globale de transition correspondant à la fonction locale δ.

Les états du lemme 3.1 ont la signification suivante ; la cellule avec l'état particulier g représente le général qui donne l'ordre de feu. A la fin du processus de synchronisation, tous les fusilliers sont dans l'état de feu F.

La solution de Minsky a été maintes fois améliorée, tout d'abord sa complexité en temps puis son nombre d'états. Les premières solutions minimales sont dues à E. Goto, A. Waksman et R. Balzer [**3, 58**]. Quelques années plus tard, une solution en temps minimal à 6 états a été proposée par J. Mazoyer [**42**]. Le nombre de transitions nécessaires a été récemment diminué par K. Noguchi [**47**], au détriment du nombre d'états.

Lemme 3.2 (lemme de synchronisation rapide)**.** Il existe un automate cellulaire $Z = (Q, \delta)$ comprenant des états particuliers, g,$F \in Q$ et un état quiescent q tel que :

$$\Delta^t(q0^{n-1}gq) = qF^nq$$

pour un temps $t = 2 \cdot n - 2$ et de telle sorte que l'état F n'apparaisse pas avant le temps t, c'est à dire $\Delta(q0^{n-1}gq)(i,t) \neq F$ pour $0 \leq t < 2 \cdot n - 2$. Ci-dessus, Δ représente la fonction globale de transition correspondant à la fonction locale δ.

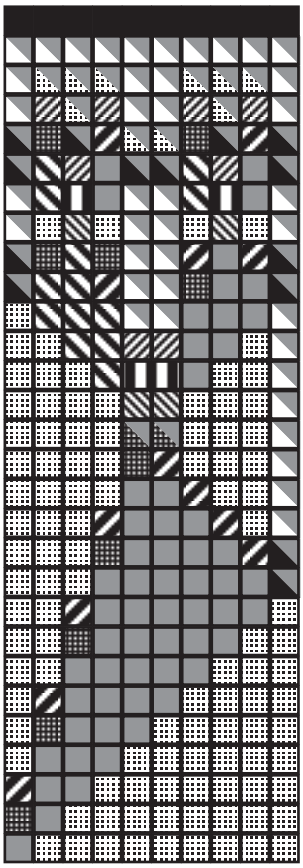

Fig. 1. Solution de Minsky.

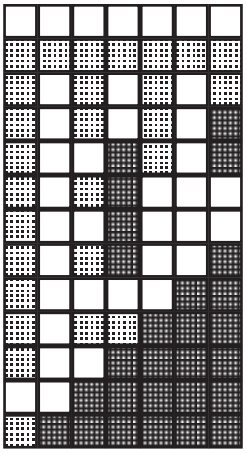

Fig. 2. Solution de Mazoyer à un général.

Les états ont la même signification que dans le lemme 3.1. La figure 2 donne un exemple de la solution de Mazoyer avec 6 états en temps $2 \cdot n - 2$.

Il est encore possible d'améliorer le temps de la synchronisation de la ligne si, au lieu d'avoir un seul général, nous en avons deux, un à chaque extrémité de la ligne (cf. figure 3). Cette solution a été développée par J. Mazoyer et N. Reimen [**44**] :

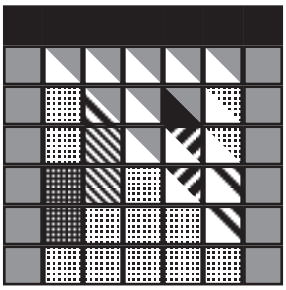

Fig. 3. Solution de Mazoyer à deux généraux.

Lemme 3.3 (Synchronisation à deux généraux)**.** Il existe un automate cellulaire $Z = (Q, \delta)$ comprenant des états particuliers, g,$F \in Q$ et un état quiescent q tel que :

$$\Delta^t(qg0^{n-2}gq) = qF^nq$$

pour un temps $t = n$ et de telle sorte que l'état F n'apparaisse pas avant le temps t, c'est à dire $\Delta(q0^{n-1}gq)(i,t) \neq F$ pour $0 \leq t < n-1$. Ci-dessus, Δ représente la fonction globale de transition correspondant à la fonction locale δ.

Il existe beaucoup d'autres généralisations du problème de la synchronisation, que ce soit sur la forme générale du graphe sous-jacent (voir par exemple [**54**]) ou sur le temps nécessaire à la synchronisation. Nous utiliserons en particulier la solution construite à partir de la remarque 3.1 qui peut être aisément obtenue à partir des résultats précédents, voire en utilisant [**19**].

Remarque 3.1. Etant donnée une solution au problème de la synchronisation en temps T, il est possible d'en construire une en temps $aT + b$ pour $a, b \in \mathbb{N}$, $a \neq 0$.

3. Automates cellulaires comme système de programmation acceptable

Dans cette section, nous rappelons la preuve que les automates cellulaires forment un système de programmation acceptable. Il faut pour cela établir une numérotation de Gödel des automates cellulaires, construire un automate cellulaire universel et une fonction récursive totale pour la composition. Nous rappelons ces trois propriétés pour le modèle des automates cellulaires.

3.1. Enumération de Gödel. Nous décrivons en détail la construction d'une énumération de Gödel des automates cellulaires. Celle-ci n'était en effet pas détaillée dans mes précédents travaux : [**30, 33**].

Théorème 3.1 (Martin [**35**])**.** Il existe une énumération effective des automates cellulaires.

Preuve. On considère $\mathcal{A} = (Q, \delta)$, un automate cellulaire à n états. On le transforme en un automate cellulaire totalistique $\mathcal{T} = (Q_T, \gamma)$ en suivant la construction décrite dans la preuve du lemme 2.1 où l'on considère l'écriture en base $b = n+1$ des images des différents éléments de l'ensemble $Q_T = \{0000, \cdots, nnnn\}$ (états du totalistique pris dans l'ordre lexicographique) par la fonction de transition totalistique γ. C'est ainsi que Wolfram [**59**] numérote ses automates cellulaires binaires, comme sa règle 150 décrite ci-dessous :

$(0,0,0)$	$(0,0,1)$	$(0,1,0)$	$(0,1,1)$	$(1,0,0)$	$(1,0,1)$	$(1,1,0)$	$(1,1,1)$
0	1	1	0	1	0	0	1

Observons que la plus grande valeur que peut prendre la somme des états de Q_T est $(nnn0)$ vu comme un entier en base b, soit nb^3+nb^2+nb en base 10. On pose $B=n.b^3+nb^2+nb+1$ comme base pour exprimer r, l'entier qui caractérise $\mathcal{T}$:

$$r=\gamma(0)B^0+\gamma(1)B^1+\ldots+\gamma(b^4-1)B^{b^4-1}$$

On code (B,r) par l'entier $R=\langle B,r\rangle$, où $\langle,\rangle$ est une bijection de $\mathbb{N}^2\to\mathbb{N}$.

On note $E(\mathcal{A})$ ce codage de $\mathcal{A}$. On ordonne ensuite l'écriture binaire de tous les codages des automates cellulaires par l'ordre hiérarchique. On associe le numéro de Gödel index($\mathcal{A}$) à l'automate cellulaire $\mathcal{A}$ en posant index($\mathcal{A}$)=i si $(\mathcal{A})$ est le i^{e} élément de l'ensemble ordonné précédent. On construit ainsi de manière effective une suite d'automates cellulaires $\mathcal{A}_0,\mathcal{A}_1,\mathcal{A}_2,\cdots$.

Réciproquement, étant donné un entier i, on cherche la valeur R qui lui correspond dans l'énumération des automates cellulaires. Soit ensuite $\Pi_1(R)=B$ la première projection de $R=\langle B,r\rangle$ et $\Pi_2(R)=r$ la seconde. Il suffit alors de décomposer $\Pi_2(R)$ en base $\Pi_1(R)$ pour retrouver r et donc γ la fonction de transition totalistique. L'ensemble des états est obtenu en cherchant la solution entière de l'équation d'inconnue n $n(n+1)^3+1=\Pi_1(R)$. Q est alors défini par $Q=\{s.m : s\in\{1,\ldots,n\},m\in\{1,10,100,1000\}\}$, des nombres exprimés en base $b=n+1$. □

Notre codage des automates cellulaires nous permet d'utiliser des entiers naturels en lieu et place des automates cellulaires. Ainsi, si l'entier p code un automate cellulaire i.e. si le prédicat PROG(p) est vrai, alors $\psi(p,x)$ est la valeur de la fonction récursive à un argument calculée par l'automate cellulaire de numéro p sur l'entrée x. Si p ne code pas un automate cellulaire i.e. si le prédicat PROG(p) est faux, on ne peut rien dire de $\psi(p,x)$. On définit une fonction partielle partiellement récursive universelle φ_{univ} comme dans [**27**] :

$$\varphi_{\text{univ}}(p,x)=\begin{cases}\psi(p,x) & \text{si PROG}(p)\\ \text{divergente} & \text{sinon}\end{cases}$$

Nous rappelons aussi la définition d'une machine universelle qui calcule une telle fonction.

Définition 3.1. Une machine est dite universelle au sens du calcul si elle peut calculer toute fonction Turing-calculable.

3.2. Un automate cellulaire universel. Il y a deux manières de concevoir l'universalité d'un modèle de calcul. La première est une universalité externe qui se sert d'une machine universelle d'un autre modèle de calcul par le biais de simulations. Nous lui préférons la notion d'universalité intrinsèque avec laquelle aucune référence extérieure au modèle n'est nécessaire.

Il est bien connu que les automates cellulaires sont capables d'une universalité externe [**55, 25**] en simulant une machine de Turing universelle ou un système de réécriture simple.

Le problème de l'universalité interne a été étudié par J. Albert et K. Culik en 1987 [**1**] avec la construction d'un automate cellulaire unidimensionnel intrinsèquement universel. Celui-ci est capable de simuler tout autre automate cellulaire unidimensionnel unidirectionnel[1] avec seulement quatorze états par cellule.

Théorème 3.2 (Albert, Culik [**1**])**.** Il existe un automate cellulaire universel U à quatorze états qui peut simuler tout automate cellulaire unidirectionnel donné A sur n'importe quelle configuration initiale.

[1]Un automate cellulaire est unidirectionnel s'il envoie son état interne dans une seule direction (c'est à dire soit vers la gauche soit vers la droite) au lieu de l'envoyer dans les deux directions à la fois. Autrement dit, la fonction locale de transition est soit de la forme $c(i,t+1)=\delta(c(i-1,t),c(i,t))$ soit de la forme $c(i,t+1)=\delta(c(i,t),c(i+1,t))$. Ce modèle d'automate cellulaire n'est pas restrictif car les deux types d'automates cellulaires ainsi définis ont été prouvés équivalents par K. Culik. Cependant, les automates cellulaires unidirectionnels sont deux fois plus lents que les bidirectionnels classiques.

Le théorème précédent ne précise pas la complexité de la simulation de A par U que nous rappelons ici. Nous commençons par considérer la complexité de la simulation d'une transition de A par l'automate cellulaire U sur l'entrée x. Supposons que le nombre d'états de A est k et que la longueur de x est m. Le calcul de la transition de A par U requiert $3(2k+2)^2+7(2k+1)$ unités de temps. Ainsi la machine universelle U est plus lente que la machine A d'un facteur quadratique en le nombre des états de A.

Complexité en espace : pour le codage de la fonction locale de transition de A en unaire, nous avons besoin de $(2k+2)^2$ cellules de la configuration initiale de U comme bloc de code. Ce bloc de code est dupliqué autant de fois que la longueur de l'entrée x, donc nous obtenons une complexité en espace de $O(mk^2)$.

Dans [**30, 33**], nous avons proposé un automate cellulaire universel qui a une meilleure complexité en temps et en espace mais au détriment du nombre des états.

Théorème 3.3 (Martin [**33**])**.** Il existe un automate cellulaire universel U avec des milliers d'états qui peut simuler en temps quasi-linéaire tout automate cellulaire A à k états donné sous forme totalistique sur n'importe quelle configuration initiale avec une perte de temps en $O(k\lceil \log k \rceil)$.

3.3. Une fonction pour la composition. Il est possible de définir une fonction c pour la composition en utilisant la définition du calcul pour un automate cellulaire. Nous ne détaillerons pas un tel programme cellulaire (voir [**33**]).

3.4. Système de programmation acceptable. Grâce aux résultats précédents, nous obtenons le théorème :

Théorème 3.4 (Martin [**33**])**.** Les automates cellulaires forment un système de programmation acceptable.

3.5. Conséquences. Pour illustrer l'importance du théorème 3.4, nous énonçons quelques résultats valides pour tout système de programmation aceptable.

La première conséquence que nous donnerons ici est le théorème *s–m–n*. Il montre que, pour tout système de programmation acceptable, la propriété de modifier des programmes en conservant certains paramètres d'entrée constants est toujours vérifiée :

Théorème 3.5. Pour tout système de programmation acceptable, il existe une fonction récursive totale s telle que pour tout i, tout $m \geq 1$ et tout $n \geq 1$ et pour tous $x_1, \ldots, x_m$ et $y_1, \ldots, y_n$,

$$\varphi_{s(i,m,x_1,\ldots,x_m)}(y_1,\ldots,y_n) = \varphi_i(x_1,\ldots,x_m,y_1,\ldots,y_n)$$

Autrement dit, cette fonction nous autorise à spécifier comme constants les m premiers arguments du i^{e} programme.

Comme seconde illustration de l'importance d'un système de programmation acceptable, nous pouvons aussi citer quelques résultats d'indécidabilité qui sont vrais quelque soit le système de programmation acceptable .

Soit ψ une fonction récursive. On dit que $\psi(x)$ converge si la valeur $\psi(x)$ existe ou, de manière équivalente, si x est dans le domaine de ψ ce que l'on note $x \in D\psi$. Respectivement, on dit que $\psi(x)$ diverge si la valeur $\psi(x)$ n'existe pas ou de manière équivalente, si $x \notin D\psi$. Cela signifie que si le calcul de ψ sur l'entrée x est convergent, alors le programme qui calcule ψ par rapport à une numérotation de Gödel s'arrête sur l'entrée x. Nous présentons ici le problème général de l'arrêt qui permet d'affirmer qu'il n'existe pas de programme qui décide si un programme donné s'arrête (ou non) sur une entrée donnée. Ici aussi, il s'agit d'un théorème vrai pour tout système de programmation acceptable [**27**].

Théorème 3.6. Soit $\varphi_0, \varphi_1, \ldots$ un système de programmation acceptable . La fonction f telle que pour tout x et y entiers,

$$f(x,y) = \begin{cases} 1 \text{ si } \varphi_x(y) \text{ est convergent} \\ 0 \text{ sinon} \end{cases}$$

n'est pas récursive

Ce théorème a un certain nombre de conséquences pratiques immédiates en ce qui concerne les automates cellulaires :

Corollaire 3.1. Il est indécidable de dire que,

(i) durant l'évolution d'un automate cellulaire sur une entrée donnée, il n'est pas possible de prédire l'occurrence d'un état particulier ;

(ii) durant l'évolution d'un automate cellulaire sur une entrée donnée, il n'est pas possible de prédire l'occurrence d'un cycle particulier ;

De même, tout autre résultat en se référrant à une quelconque définition de calcul licite pour les automates cellulaires sera indécidable

Avec tous ces résultats, on peut montrer comme conséquence le théorème de Rice qui signifie qu'il n'existe pas de propriété décidable non triviale sur le comportement des entrées et des sorties d'un programme. En d'autres termes, on ne peut pas trouver de propriétés sur les fonctions récursives partielles en regardant les programmes qui les exécutent. Pour ce faire, on définit $\mathcal{P}_C$, l'ensemble de fonctions récursives partielles satisfaisant une propriété donné : $\mathcal{P}_C = \{x : \varphi_x \in C\}$. Le théorème de Rice s'énonce :

Théorème 3.7. $\mathcal{P}_C$ est récursif si et seulement si $\mathcal{P}_C = \varnothing$ ou $\mathcal{P}_C = \mathbb{N}$.

On en déduit qu'il n'est pas possible de tester algorithmiquement si deux automates cellulaires peuvent accomplir le même calcul. Le théorème 3.7 est très important car il détruit tout espoir de tester algorithmiquement le comportement des entrées/sorties des automates cellulaires.

CHAPITRE 4

Simulation et universalité

Nous nous intéressons tout particulièrement à la simulation d'automates cellulaires, que ce soit par des automates cellulaires eux-mêmes (pour obtenir un automate cellulaire universel ou pour changer la topologie) ou par d'autre modèles de calcul pour construire différentes machines universelles.

Dans ce chapitre, nous rappelons une notion de simulation courante (voir par exemple [**12**] ou [**52**]). Nous présentons également différentes manières de concevoir des machines universelles que nous illustrerons dans les chapitres suivants. Ces différentes façons de construire des machines universelles ainsi que les conséquences ont été présentées au congrès FCT en 1997 [**35**].

1. Définition de la simulation

Dans cette section, nous proposons une définition de la simulation entre deux machines qui peuvent provenir de modèles de calcul différents. La définition 4.1 exprime que, si la machine A simule le fonctionnement d'une étape de la machine B en τ unités de temps, on doit pouvoir construire une correspondance entre les configurations équivalentes (à un codage près) de la machine A et de la machine B.

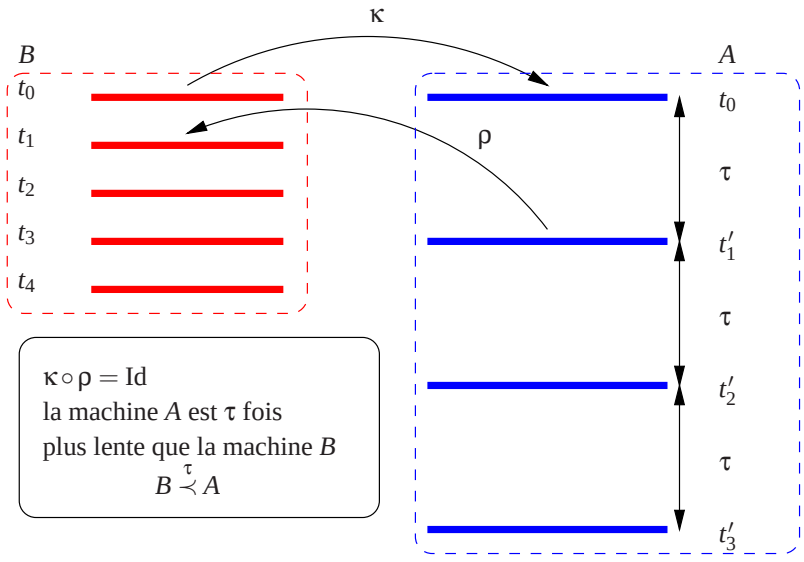

Fig. 1. Illustration de la définition de simulation où B est simulée par A.

Définition 4.1. Soit $\mathbb{C}_A$ et $\mathbb{C}_B$ les ensembles de configurations des machines A et B (respectivement). On dit que A simule chaque pas de calcul de B en temps τ (et on note $B \overset{\tau}{\prec} A$) s'il existe deux fonctions calculables $\kappa : \mathbb{C}_B \to \mathbb{C}_A$ et $\rho : \mathbb{C}_A \to \mathbb{C}_B$ telles que si $\kappa \circ \rho = \mathrm{Id}$ et pour tout $c, c' \in \mathbb{C}_B$, il existe une configuration $c'' \in \mathbb{C}_A$ telle que si $c \vdash_B c'$,

$\kappa(c) \vdash_A^{\tau} c''$ avec $\rho(c'') = c'$, où $\vdash_M$ représente une application de la fonction de transition globale de la machine M et $\vdash_M^t$ t itérations de cette même fonction de transition.

Observons également que la définition 4.1 est proche de la réduction utilisée pour les classes de complexité fonctionnelles, en relaxant toutefois la contrainte de complexité. En citant le livre de Papadimitriou [**48**], définition 10.1 :

Définition 4.2 (Papadimitriou [**48**])**.** On peut définir une réduction entre des problèmes de calcul. On dit que le problème de calcul B se réduit au problème de calcul A si les conditions suivantes sont vérifiées : il existe deux fonctions R et S, toutes deux calculables en espace logarithmique, telles que pour toute entrée x et z on a : si x est une donnée de B, alors $R(x)$ est une donnée de A. De plus, si z est une sortie correspondant à l'entrée $R(x)$, alors $S(z)$ est la sortie correspondant à l'entrée x.

A partir de la définition 4.2 on peut définir la complétude des classes de complexité fonctionnelles relatives à P et à NP.

La fonction R joue un rôle analogue à notre fonction κ et S un rôle analogue à ρ. Dans le cas de la définition 4.1, nous n'avons pas besoin de borner la complexité du calcul de ces deux fonctions puisque nous nous intéressons à des problèmes de calculabilité et non de complexité.

2. Construction de machines universelles

Dans son livre de 1967, M. Minsky [**46**] décrit une machine universelle comme :

The universal machine as an interpretive computer

The universal machine will be given just the necessary materials : a description, on its tape, of T and of s_x (string of symbols corresponding to the entry) ; some working space ; and the built-in capacity to interpret correctly the rules of operation as given in the description of T. Its behavior is very simple. U will simulate the behavior of T one step at a time...

Dans cette section, nous décrivons plusieurs façons de construire une machine universelle qui dépendent de la simulation. En effet, lorsqu'on veut construire une machine universelle, on commence par choisir un modèle de calcul M. Nous distinguons deux cas :

- on n'a, a priori, aucune connaissance sur le modèle M ; il nous faut :
  - énumérer les machines de M ;
  - construire une machine de M particulière qui simule le fonctionnement de toute autre machine de M.
- on connait déjà un modèle de calcul M' qui possède une machine universelle ; alors soit :
  - on construit une machine particulière de M capable de simuler le fonctionnement de toute machine du modèle M' ;
  - on construit une machine particulière de M qui simule une machine universelle de M'.

Selon le cas, on identifie une classe bien particulière de machine universelle que nous décrivons ci-dessous :

- l'universalité par simulation : on simule le fonctionnement
  - de n'importe quelle machine d'un autre modèle ;
  - de n'importe quelle machine du même modèle.
- l'universalité par hérédité : on simule le fonctionnement d'une machine universelle d'un autre modèle.
- l'universalité par construction : on construit explicitement une machine dont on a reçu le code dans un langage (codage) quelconque.

2.1. Universalité par simulation.

Définition 4.3 (Martin [**35**])**.** Une machine P_{univ} du modèle M est universelle par simulation si, pour toute machine P'_i du modèle M' et pour toute entrée x, $P'_i(x) \overset{\tau}{\prec} P_{\text{univ}}(i,x)$.

Dans ce cas, P_{univ} sert d'émulateur aux machines de M'. On distingue un cas particulier, celui de l'universalité intrinsèque, aussi appelée universalité forte par Albert et Culik [**1**] : dans ce cas, la machine P_{univ} du modèle M peut simuler toute machine de M. C'est notamment le cas de certaines machines de Turing universelles mais aussi celui de quelques automates cellulaires universels comme [**1, 29, 33**].

2.2. Universalité par hérédité.

Définition 4.4 (Martin [**35**])**.** Une machine P_h du modèle M est universelle par hérédité s'il existe une machine universelle P'_{univ} du modèle M' telle que pour tout i et x, $P'_{\text{univ}}(i,x) \overset{\tau}{\prec} P_h(i,x)$.

Dans ce cas, la machine P_h se comporte comme un interpréteur pour les machines du modèle M'. C'est en particulier le cas de certaines -petites- machines de Turing qui simulent des TAG-systems comme décrit par Margenstern dans [**28**] ou dans [**17, 25, 55**]. Le résultat classique qui stipule que les automates cellulaires sont aussi puissants que les machines de Turing repose sur la simulation d'une machine de Turing universelle (voir [**55**]).

2.3. Universalité par construction.

Définition 4.5 (Martin [**35**])**.** Une machine P_c du modèle M est universelle par construction si pour toute machine P'_i du modèle M', $P_c(\chi(P'_i)) = P'_i$ où χ dénote le code du programme de la machine P'_i.

L'exemple le plus classique qui illustre cette définition est celui des réseaux booléens. Pour ce modèle de calcul, une machine de Turing reçoit en entrée le codage standard (voir [**22**]) d'un circuit ainsi que sa taille et construit explicitement le réseau booléen correspondant. Pour le modèle des automates cellulaires, Arbib [**2**] énonce «Turing's result that there exists a universal computing machine suggested to von Neumann that there might be a universal construction machine P_c, which, when furnished with a suitable description P_n of any appropriate automaton A, will construct a copy of A». Il s'agit de la première machine universelle par construction qui mène à l'auto-reproduction.

2.4. Équivalence pour le calcul. Les trois définitions qui permettent la construction de machines universelles sont cependant équivalentes vis à vis de la définition de l'universalité au sens du calcul (définition 3.1, page 12). C'est ce qu'atteste le théorème suivant :

Théorème 4.1 (Martin [**35**])**.** Toute machine universelle P_{univ} du modèle M qu'elle soit universelle par simulation, par hérédité ou par construction l'est également au sens du calcul.

Preuve. Nous montrons que chacun de nos types d'universalité est également universel au sens du calcul

– Soit P_U une machine du modèle M universelle par simulation et soit $P_0, P_1, \cdots$ une énumération de ces machines. Pour tout i, soit $\varphi_i : \mathbb{N} \to \mathbb{N}$ la fonction calculée par P_i. Alors, $\varphi_0, \varphi_1, \cdots$ est une énumération des fonctions calculables par les machines du modèle M. Supposons en outre que M soit un système de programmation acceptable. Soit $\psi_0, \psi_1, \cdots$ une énumération des machines du modèle M'. Par la définition 4.3, on a que toute fonction calculable pour le modèle M' l'est également pour le modèle M. il existe donc f une fonction récursive telle que pour tout x, $\psi_x = \varphi_{f(x)}$. Ainsi,

$$\varphi_{\text{univ}}(f(i),x) = \varphi_{f(i)}(x) = \psi_i(x)$$

L'énumération $\psi_0, \psi_1, \cdots$ possède une fonction universelle et il existe donc une machine qui calcule cette fonction. Il suffit à une machine P_{univ} du modèle M de simuler cette machine universelle du modèle M' pour être universelle au sens du calcul.

– Soit P_{univ} une machine du modèle M universelle par hérédité. Comme $\psi_{\text{univ}}(i,x) = \psi_i(x)$, la définition 4.4 coïncide avec la définition 3.1, d'où le résultat.
– Soit P_{univ} une machine du modèle M universelle par construction. Par la définition 4.5, elle est aussi universelle au sens du calcul.

□

CHAPITRE 5

Simulation d'un CA par des PRAM

Le concept de Parallel Random-Access Machine (abrégé par PRAM) a été introduit comme une généralisation parallèle du modèle des RAM dans les années 1976 pour le traitement de grandes quantités de données. Dans ce modèle, le temps a été prouvé équivalent, à un facteur polynomial près, à l'espace d'une machine de Turing [**5**]. Ce modèle est capable de gérer un grand nombre de processeurs. Chacun d'eux utilise une mémoire locale et une mémoire globale partagée qui permet l'échange de messages entre les processeurs. De plus, les processeurs ne sont pas contraints d'exécuter la même opération au même instant comme c'est le cas avec les automates cellulaires.

Le modèle des XPRAM a été introduit dans les années 1990 par Valiant [**56, 57**] sous le nom de bulk synchronous parallel model (BSP). Il se démarque du modèle classique des PRAM en interdisant l'utilisation d'une mémoire globale partagée ; il ajoute un «réseau d'interconnexion» pour les échanges de données ainsi qu'un mécanisme de synchronisation global de la machine. Ce modèle de calcul parallèle à mémoire distribuée a été conçu pour être plus proche de la réalité que celui des PRAM avec un certain succès, comme en attestent des implémentations de BSP pour de vraies machines parallèles [**8, 9**].

Les Parallel Random-Access Machines sont souvent comparées aux réseaux booléens ou aux machines de Turing alternantes mais très rarement aux automates cellulaires. C'est une telle comparaison que nous allons illustrer dans un premier temps en montrant que les automates cellulaires peuvent servir de modèle de base aux Parallel Random-Access Machines. Ce premier résultat a été présenté à MFCS en 1993 [**31**]. Nous étudierons ensuite la simulation des automates cellulaires par des XPRAM. Selon le nombre de processeurs qu'on autorise, on obtient deux résultats présentés à la conférence SCI/ISAS en 1998 [**37**].

1. Définition des PRAM

Plus formellement [**22**], une Parallel Random-Access Machine consiste en un ensemble dénombrable de processeurs $p_0, p_1, \ldots$, une mémoire globale infinie $x_0, x_1, \ldots$ et un programme fini. Chaque processeur p_i a une mémoire locale infinie $y_0, y_1, \ldots$. Le registre y_0 est appelé l'accumulateur du processeur. Chaque processeur dispose d'un compteur de programme et d'un drapeau indiquant si le processeur est actif ou non. Un programme est une suite finie d'instructions parmi :

$y_i :=$ constant;	goto m if $y_i > 0$;
$y_i := y_i + y_k$;	$y_i := x_{y_i}$;
$y_i := \lceil y_i/2 \rceil$;	$x_{y_i} := y_i$;
$y_i := y_{y_i}$;	accept;
fork m;	

Le parallélisme est réalisé par l'instruction "fork m". Lorsqu'un processeur p_i exécute cette instruction, il sélectionne le premier processeur inactif p_j, remet à zéro la mémoire locale de p_j et charge l'accumulateur de p_i dans celui de p_j. Alors, p_j commence son calcul à l'étiquette m de son programme. Les communications entre les processeurs sont réalisées par des accès à la mémoire globale partagée. Dans le modèle PRAM, on autorise les lectures concurrentes simultanées de la mémoire globale par plusieurs processeurs. Si deux processeurs tentent d'écrire une valeur à la même case de la mémoire globale, la

machine s'arrête immédiatement et entre dans un état de rejet dans le cas du modèle le plus restrictif. Plusieurs processeurs peuvent lire une même case de la mémoire globale quand un processeur est en train de l'écrire. Dans ce cas, toutes les lectures sont réalisées avant que la valeur de la case mémoire soit changée.

Afin de traiter différents niveaux de concurrence et pour régler les conflits de lecture et d'écriture dans la mémoire globale, il a été défini des variantes de cette définition :

- le modèle EREW PRAM pour lequel ni les lectures concurrentes ni les écritures concurrentes ne sont autorisées.
- le modèle CREW PRAM qui autorise les lectures concurrentes mais pas les écritures.
- le modèle CRCW PRAM qui permet à la fois les lectures et les écritures concurrentes.

De plus, il est possible de rafiner le modèle des CRCW PRAM en détaillant la manière d'arbitrer les conflits à l'écriture.

2. Simulation d'un CA par une PRAM

Généralement, on définit l'universalité des PRAM à l'aide des circuits ou des machines de Turing alternantes comme modèle de référence. Comme chacun de ces deux modèles est capable d'universalité, il est évident que par une simulation adéquate les PRAM en sont aussi capables. Mais les deux modèles ci-dessus mentionnés ont des inconvénients. Dans le cas des circuits, les entrées sont obligatoirement finies et il faut enrichir le modèle de l'hypothèse d'uniformité pour qu'une famille de circuits travaille sur des entrées de taille quelconque. Nous proposons ci-dessous la simulation d'automates cellulaires par les PRAM. Nous obtenons ainsi le théorème 5.1 pour un automate cellulaire totalistique A à k états.

Théorème 5.1 (Martin [**31**])**.** Il existe une CREW PRAM capable de simuler le comportement de tout automate cellulaire totalistique A sur n'importe quelle entrée en temps constant.

Preuve. L'idée de la simulation est suggérée par la figure 1 qui décrit la manière dont la PRAM simule l'évolution d'un automate cellulaire. Le contenu d_i de chaque cellule c_i de A, l'automate cellulaire à simuler, est écrit dans un registre de la mémoire locale de chaque processeur. Le code (il s'agit en fait de la fonction de transition totalistique) de A est contenu dans la mémoire globale partagée de la PRAM. Le comportement de la PRAM est alors :

- chaque processeur p_i écrit d_i dans un registre (figure 1) ;
- chaque p_i somme le contenu des registres de p_{i-1} et p_{i+1} et inscrit le résultat r à la place de la valeur d_i. Le nombre r vérifie $0 \leq r \leq 3.\#\text{états}(A)$;
- chaque p_i recherche dans la table de transition l'image de r par la fonction de transition totalistique f_A et remplace la valeur de r par la valeur $f_A(r)$;
- les points ci dessus sont itérés jusqu'à obtention du résultat.

Nous donnons ci-dessous les détails de la preuve de la simulation d'un (quelconque) automate cellulaire A par la PRAM P. Supposons que P a un nombre infini de processeurs. La première chose à faire est d'associer un processeur à chaque cellule de A. Les registres contiennent alors les informations suivantes : chaque donnée de la demi-ligne d'automates A est contenue dans l'un des registres du processeur associé à la cellule i. Les registres suivants contiennent tous le nombre 0 qui représente l'état quiescent.

Le code de A (sans codage supplémentaire) est contenu dans la mémoire globale partagée sous la forme d'une table indicée par les valeurs du domaine de la fonction totalistique f_A. De cette manière, l'accès à l'image d'une valeur par la fonction de transition peut être

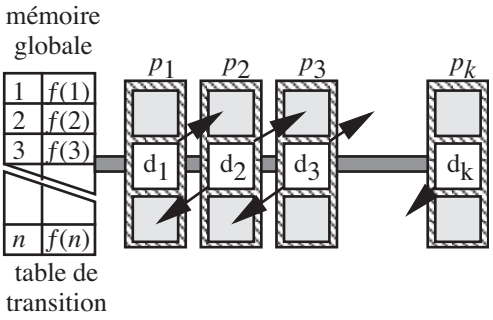

Fig. 1. Simulation d'un CA par une PRAM.

fait en temps constant. Alors le programme de chaque processeur p_i est le suivant :

$1\ y_3 := i + (3.k + 2);$
$2\ y_0 := x_{y_0};$
$3\ x_{y_3} := y_0;$
$4\ y_1 := x_{y_3-1};$
$5\ y_0 := y_0 + y_1;$
$6\ y_1 := x_{y_3+1};$
$7\ y_0 := y_0 + y_1;$
8 goto 2 if $y_0 > 0;$

Remarquons qu'une «nouvelle» cellule est créée quand son contenu passe de la valeur 0 à une valeur non nulle. De plus, une seule cellule à la fois peut être créée pendant une itération. Cela signifie que le nombre de cellules non-quiescentes au cours d'un calcul de l'automate cellulaire peut être borné. Cette borne correspond plus ou moins à l'idée qu'on a de l'espace d'une machine de Turing.

Observons qu'une CREW PRAM suffit pour la simulation puisqu'il n'y a pas d'écritures concurrentes mais seulement des lectures concurrentes de la table qui représente la fonction de transition de A. De surcroît, le processus défini ci-dessus simule une itération de A en un temps constant. □

3. Définition des XPRAM

Une XPRAM consiste en un ensemble dénombrable de processeurs auxquels on associe une mémoire locale ainsi qu'un mécanisme qui permet de synchroniser l'ensemble des processeurs de la machine, appelé barrier synchronisation. Chaque processeur p_i a une mémoire locale infinie $y_0, y_1, \ldots$. Le registre y_0 est appelé l'accumulateur du processeur. Chaque processeur p_i dispose d'un compteur de programme et d'un drapeau indiquant si le processeur est actif ou non. Un programme est une suite finie d'instructions choisies dans l'ensemble suivant :

$y_i :=$ constant;	goto m if $y_i > 0$;
$y_i := y_i + y_k;$	$[P_j.y]_{y_k} := y_l;$
$y_i := \lceil y_i/2 \rceil;$	$y_k := [P_j.y]_{y_l};$
$y_i := y_{y_i};$	accept;
synchro;	

En sus des instructions décrites dans la section 1, on définit des opérations de communication et de synchronisation. L'instruction $[P_j.y]_{y_k} := y_l$ correspond à l'envoi de la donnée locale dont l'adresse est contenue dans le registre y_l du processeur P_i vers le registre y_k du processeur j. L'instruction qui correspond à la réception de la donnée est $y_k := [P_j.y]_{y_l}$; elle mémorise dans le registre y_k du processeur courant P_i la donnée pointée par y_l reçue

du processeur P_j. Le calcul d'une XPRAM est découpé en une suite d'étapes générales – les supersteps –; chacune de ces étapes générales consiste en une suite d'instructions suivie d'une synchronisation globale (l'instruction synchro) qui permet l'échange global des données.

Les performances d'une XPRAM sont mesurées au moyen de trois paramètres :

- p, le nombre de processeurs ;
- ℓ, le nombre d'instructions entre chaque synchronisation globale ;
- g, le rapport entre le nombre total d'instructions effectuées par l'ensemble des processeurs en une unité de temps fixée et la quantité globale d'information échangée dans le réseau d'interconnexion pendant la même période de temps.

Le dernier paramètre, g, mesure en quelque sorte la fréquence à laquelle on peut accéder à des données distantes.

Le temps de calcul d'une étape générale S est défini d'une manière un peu compliquée. On définit tout d'abord le travail w comme le nombre total d'instructions exécutées par l'ensemble des processeurs entre deux synchronisations. Soit h_s et h_r deux variables qui comptent respectivement le nombre maximal de messages émis et le nombre maximal de messages reçus par chaque processeur au cours de l'étape générale S. Le temps de S est alors majoré par $w + g.\max\{h_s, h_r\} + \ell$.

Le temps de calcul d'une XPRAM est obtenu en sommant les temps de calcul de chaque étape générale. On obtient une expression de la forme $W + H.g + S.\ell$, où W, H et S dépendent de n, la taille de l'entrée et de p, le nombre de processeurs actifs.

Nous définissons l'efficacité η d'une simulation. On suppose qu'une XPRAM à p processeurs M de complexité en temps T simule le fonctionnement d'une XPRAM à p' processeurs M' de complexité en temps T'. L'efficacité de la simulation est $\frac{Tp}{T'p'} \geq \eta$.

4. Simulation d'une EREW PRAM par une XPRAM

Pour mieux comprendre le fonctionnement d'une XPRAM et introduire la notion de fonctions de hachage rapides nous reprenons la simulation d'une EREW PRAM par une XPRAM introduite dans le chapitre de L. Valiant [**57**]. Pour réaliser cette simulation, L. Valiant a besoin de bonnes fonctions de hachage qui calculent rapidement l'adresse en fonction du paramètre g.

On dispose d'une EREW PRAM M à p processeurs et on considère sa mémoire partagée comme un ensemble de m registres. On souhaite en simuler le fonctionnement par une XPRAM M' à p' processeurs avec également m registres qui seront distribués sur les mémoires locales des processeurs de M'. On suppose pour cela que $\forall i \in P$, la mémoire du processeur P_i de M' contient tous les registres $j \in \{1, \ldots, m\}$ tels que $j \mod p' = i$.

Pour chaque exécution d'un programme, on tire aléatoirement une fonction de hachage h d'un ensemble H de fonctions de hachage universelles. Chaque fonction h est une permutation de $\{1, \ldots, m\}$. L'idée est qu'en simulant le fonctionnement de M par M', M' va représenter le registre j de la mémoire de M par $h(j) \mod p'$. Le rôle de la fonction h est d'étaler le plus possible les accès mémoire dans les registres de M'. Pour cela, on requiert la propriété suivante sur H : pour tout ensemble S de p registres (le nombre de processeurs de la EREW PRAM), $h(S)$ devrait être uniformément distribué sur les p' processeurs de M' avec une forte probabilité.

Une classe H de telles fonctions de hachage h a été introduite dans [**45**] :

$$h(j) = \sum_{i \in \{1,\ldots,k\}} a_i j^i \mod m$$

Cette classe de fonctions vérifie bien la propriété, comme en atteste le lemme 5.1.

Lemme 5.1 (Melhorn and Vishkin [**45**])**.** Si m est premier, alors pour tout S de cardinal p,

$$\Pr\{R(p',S,h) \geq k | h \in H\} \leq \binom{p}{k} p'^{1-k} e^{kp'/m}$$

avec $R(p',S,h) = \max_{i\in\{1,\ldots,p'\}} \sharp\{ j \in S : h(j) \mod p' = i\}$.

On dit que $\mathcal{H}$, une famille de classes de fonctions de hachage, est bonne si, pour tout p' et pour tout $m \geq p' \log p'$, il existe $H \in \mathcal{H}$, une classe de fonctions $\{h : \{1,\ldots,m\} \to \{1,\ldots,m\}\}$ telle que $\hat{R}(p',H,p'\log p') = O(\log p')$, où $\hat{R}(p',H,v) = \max_{S,|S|=p} \left\{ \frac{1}{|H|} \sum R(p',S,h) \right\}$.

Corollaire 5.1 (Melhorn and Vishkin [**45**])**.** Il existe une bonne famille de classes de fonctions de hachage.

Preuve. Si m est premier, on remplace k par $4\log p'$ et p par $p'\log p'$ dans le lemme 5.1 et on obtient que, pour tout S de cardinal $p'\log p'$, $\Pr\{R(p',S,h) \geq 4\log p'\} = o(p'^{-1})$.

Si m n'est pas premier, on procède de même avec m' premier nombre premier plus grand que m. □

On dit qu'une famille de XPRAM paramétrée par p', le nombre de processeurs et par m, le nombre de registres adressés possède une fonction de hachage rapide si elle implémente une bonne fonction de hachage calculable en temps αg pour α une constante qui ne dépend ni de p' ni de m.

L. Valiant montre alors qu'une EREW-PRAM à p processeurs peut être simulée par une XPRAM à p' processeurs :

Théorème 5.2 (Valiant [**57**])**.** Une XPRAM à p' processeurs munie de fonctions de hachage rapides peut simuler une EREW-PRAM à p processeurs avec une efficacité moyenne optimale si $p \geq p'\log p'$.

Preuve. On souhaite simuler le fonctionnement de M, une EREW-PRAM à p processeurs. On choisit aléatoirement une fonction de hachage $h \in H \in \mathcal{H}$ en fonction des paramètres (p,m). Le contenu du registre j de M sera dans le registre $h(j) \mod p'$ de la XPRAM M'. On distribue les tâches des p processeurs de M arbitrairement de façon à ce que chaque processeur de M' en simule au plus $\lceil p/p' \rceil$. En simulant une étape de M, il va falloir effectuer $\lceil p/p' \log p' \rceil$ «supersteps» de M'. Dans chaque «superstep», pour chaque processeur, on va choisir aléatoirement un ensemble de $\log p'$ accès parmi les $\lceil p/p' \rceil$ qui sont à faire. Par le choix de $\mathcal{H}$, la moyenne du plus grand nombre d'accès effectués sur chaque registre pendant un «superstep» sera $O(\log p')$. Il s'ensuit que le temps moyen pour faire un «superstep» sera identique. En faisant la somme sur les «supersteps» de M' et les étapes de M, on obtient que le temps moyen de simulation de M par M' est $O(p/p')$.

On observe que, si le nombre total de registres utilisés pour l'entrée n'excède pas le nombre d'opérations de calcul de M', toutes les données peuvent être obtenues par h en temps parallèle $O(p/p')$. □

5. Simulation d'un CA par une XPRAM

Nous supposons dans la suite que nous simulons le fonctionnement d'un automate cellulaire à n cellules par une XPRAM à p processeurs.

Nous étudions dans un premier temps une borne théorique obtenue par une suite de simulations avant de décrire l'efficacité de nos simulations. Nous distinguons deux cas ; celui (plus facile) où nous avons plus de processeurs que le nombre de cellules de l'automate cellulaire et celui où le nombre de cellules de l'automate cellulaire excède le nombre de processeurs dont dispose notre XPRAM.

5.1. Une première borne. Dans ce paragraphe, nous citons quelques résultats de simulation connus qui, une fois combinés, vont nous donner une borne sur l'efficacité de la simulation d'un automate cellulaire par une XPRAM.

Nous rappelons tout d'abord le résultat qui mesure l'efficacité de la simulation d'une CREW-PRAM par une EREW-PRAM :

Lemme 5.2 (Karp and Ramachandran [**23**])**.** Une EREW-PRAM à p processeurs peut simuler le fonctionnement d'une CREW-PRAM à p processeurs avec une efficacité $O(\log p)$.

En combinant le lemme 5.2, le théorème 5.2 et notre simulation d'un automate cellulaire par une EREW-PRAM (théorème 5.1), on obtient la proposition suivante :

Proposition 5.1 (Martin [**37**])**.** Une XPRAM à p processeurs munie de fonctions de hachage rapides peut simuler n'importe quel automate cellulaire totalistique à n cellules avec une efficacité moyenne $O(\frac{p}{n}\log n)$ si $n \geq p\log p$.

Nous pouvons nous demander si cette borne obtenue au moyen d'une chaîne de simulations peut être atteinte par une simulation directe. Les simulations directes que nous proposons par la suite ne permettent pas de répondre affirmativement à cette interrogation.

5.2. Cas où $p \geq n$. C'est le cas le plus facile. On s'inspire de la simulation d'un automate cellulaire par une PRAM décrit dans la section 2 pour obtenir le résultat :

Théorème 5.3 (Martin [**37**])**.** Une XPRAM à p processeurs peut simuler n'importe quel automate cellulaire totalistique à n cellules avec une efficacité optimale lorsque $p \geq n$.

Preuve. On se donne l'automate cellulaire à simuler par sa fonction de transition totalistique f :

0	1	$\ldots$	$m-1$	m
$f(0)$	$f(1)$	$\ldots$	$f(m-1)$	$f(m)$

On réserve les $m+1$ premières adresses de la mémoire locale du processeur P_i pour mémoriser f et accéder directement à l'image d'une valeur. Le registre y_{m+2} va mémoriser l'état de la cellule $i-1$, y_{m+3} celui de la cellule simulée par P_i et y_{m+4} celui de la cellule $i+1$. Le programme de chaque processeur P_i est alors :

0. $y_{m+5} := 1$;
1. $y_{m+2} := [P_{i-1}.y]_{m+3}$;
2. $y_{m+4} := [P_{i+1}.y]_{m+3}$;
3. synchro;
4. $y_{m+3} := y_{m+3} + y_{m+2}$;
5. $y_{m+3} := y_{m+3} + y_{m+4}$;
6. $y_{m+3} := y_{y_{m+3}}$;
7. goto 1 if $y_{m+5} > 0$;

Les lignes 1 et 2 du programme permettent à P_i de récupérer le contenu des cellules voisines simulées ; les lignes 4 et 5 calculent la somme des valeurs de cellules et la ligne 6 remplace le contenu de l'état de la cellule simulée par P_i par son image par f. Le registre y_{m+5} est utilisé pour définir une condition d'arrêt compatible avec la définition d'un calcul sur automate cellulaire.

Le temps de calcul d'une étape générale est $6 + 2.g + \ell$. Observons qu'il ne dépend ni de p, ni de n. la simulation se fait en temps constant, ce qui nous donne une efficacité optimale. □

5.3. Cas où $p < n$. On considère maintenant que le nombre de processeurs de la XPRAM est inférieur au nombre de cellules de l'automate cellulaire. Dans ce cas, nous avons montré le théorème 5.4.

Théorème 5.4 (Martin [**37**])**.** Une XPRAM à p processeurs peut simuler un automate cellulaire à $n > p$ cellules en temps $O\left(\left\lceil \frac{T(n)(T(n)+n)}{p} \right\rceil\right) + g.O(pT(n)) + \ell.T(n)$ pour $T(n)$ la complexité en temps de l'automate cellulaire.

Preuve. On supposera que le nombre de cellules de l'automate cellulaire croît d'une unité à chaque itération. En d'autres termes, on s'intéresse au modèle d'automate cellulaire qui ne s'étend que vers la droite, plutôt qu'à celui qui s'étend des deux cotés. Il ne s'agit pas d'une restriction trop forte car les deux modèles ont été montrés équivalents dans [**43**].

Le programme du processeur P_i pour $1 < i < p$ est comme suit :

Algorithme 5.1.

while not fini **do**
  read(P_{i-1},T[0],newstate) ;
  read(P_{i+1},nbmsg,Rmsgin[0]) ;
  read (P_{i-1}, Lcellnb, Lmsgin) ;
  synchronize ;
  for $i \leftarrow 1$ **to** nbmsg **do** $T[\text{cellnb}+i] \leftarrow \text{Rmsgin}[i]$;
  if nbmsg> 1 **then** cellnb$\leftarrow$cellnb+1 ;
  for $i \leftarrow 1$ **to** cellnb+1 **do**
    $T'[i] \leftarrow f(T[i-1]+T[i]+T[i+1])$;
  for $i \leftarrow 1$ **to** cellnb **do** $T[i] \leftarrow T'[i]$;
  if cellnb-Lcellnb > 1 **then** nbmsgout$\leftarrow 2$;
    else nbmsgout$\leftarrow 1$;
  for $i \leftarrow 1$ **to** nbmsg **do** $\text{Lmsgout}[i] \leftarrow T[i]$;
  Lmsgout[0] $\leftarrow$ nbmsgout ;
  write (P_{i-1}, nbmsgout, Lmsgout) ;
  write (P_{i+1},T[cellnb-nbmsg-1],newstate) ;
  write (P_{i+1},cellnb,Lmsgin) ;
endwhile

On suppose que chaque processeur a mémorisé une copie de la fonction de transition et possède un accès direct à ses valeurs.

Dans l'algorithme précédent, nous avons quelque peu enrichi le jeu d'instruction des XPRAM de base. Ici, l'instruction de lecture `read` est de la forme `read(P,localvar,fromvar)` qui lit la variable `fromvar` du processeur `P` et qui l'inscrit dans la variable `localvar`. L'instruction d'écriture s'interprète de manière analogue. La variable `cellnb` contient le nombre de cellules simulées par le processeur P_i, la table `Rmsgin[]` mémorise les nouvelles valeurs des cellules simulées par le processeur P_{i+1} après la synchronisation globale ; la table `Lmsgout[]` contient les nouvelles valeurs des cellules simulées par le processeur P_i qui vont être envoyées au processeur P_{i-1} à la fin de l'étape générale ; la variable `nbmsg` contient le nombre de messages reçus du processeur P_{i+1}.

La variable `fini` doit être mise à jour pour définir une condition d'arrêt compatible avec la définition d'un calcul sur un automate cellulaire. La figure 2 illustre la simulation d'un automate cellulaire à 8 cellules par une XPRAM à 3 processeurs. A chaque étape générale, une cellule de plus est active sur l'automate cellulaire. Les flèches représentent les opérations de communication entre les processeurs.

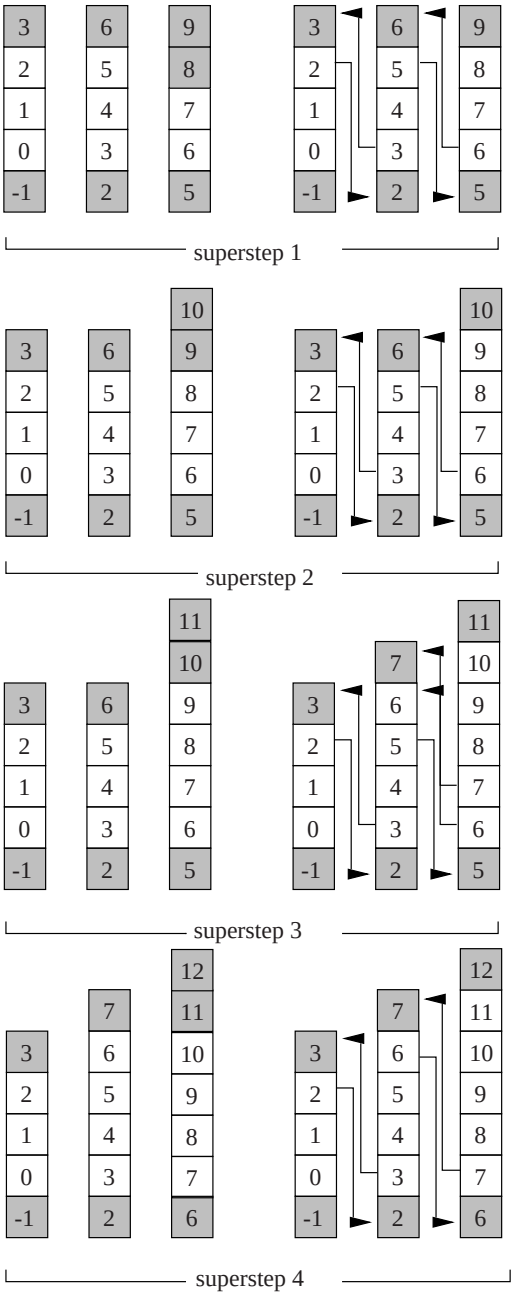

Fig. 2. Simulation d'un CA par une XPRAM.

Le comportement du processeur P_1 est analogue, sauf qu'il ne communique pas les nouveaux états des cellules qu'il simule au processeur P_0 ; le processeur P_p ne fait ni lecture ni écriture vers le processeur P_{p+1} ; il initialise `T[cellnb+1]` et `T[cellnb+2]` à la valeur 0 (qui représente l'état quiescent) ; il calcule aussi la nouvelle valeur de la cellule simulée par `T[cellnb+1]` et incrémente `cellnb` si la valeur de `T[cellnb+1]` passe de 0 à une autre valeur.

Le travail maximal par processeur est $\sum_{i=1}^{T(n)} \left\lceil \frac{n+i}{p} \right\rceil \approx O\left(\left\lceil \frac{T(n)(T(n)+n)}{p} \right\rceil \right)$, le nombre maximal de messages correspond au nombre maximal de messages reçus qui vaut $2p \cdot T(n)$

et il faut accomplir $T(n)$ synchronisations. On obtient donc l'expression suivante pour le temps de calcul de la XPRAM :

$$O\left(\left\lceil \frac{T(n)(T(n)+n)}{p} \right\rceil\right) + g.O(pT(n)) + \ell.T(n)$$

□

L'idée sous-jacente de cet algorithme de simultion est la suivante : si la charge d'un processeur devient plus grande que celle du processeur précédent il transfère la simulation d'une cellule à son prédécesseur.

6. Conséquences et conclusion

En corollaire du théorème 5.1, nous pouvons dire que :

Corollaire 5.2. Il existe une PRAM universelle au sens du calcul.

Pour cela, il suffit de simuler le fonctionnement d'un automate cellulaire arbitraire sur une entrée quelconque. Le modèle des PRAM comporte donc une machine universelle par simulation au sens de la définition 4.3. En effet, notre PRAM reçoit en entrée f_A, la fonction de transition totalistique d'un automate celulaire A, son entrée x et simule le fonctionnement de A sur l'entrée x. Ainsi, le modèle des PRAM peut être vu comme un système de programmation universel avec, comme modèle de référence les automates cellulaires.

En corollaire des théorèmes 5.3 et 5.4, nous pouvons dire que :

Corollaire 5.3 (Martin [**37**])**.** Il existe une XPRAM universelle au sens du calcul.

En effet, notre XPRAM est capable de simuler le fonctionnement de n'importe quel automate cellulaire sur n'importe quelle entrée. Pour construire une XPRAM universelle au sens du calcul, il suffit de simuler le fonctionnement d'un automate cellulaire arbitraire. Le modèle des XPRAM comporte donc une machine universelle par simulation au sens de la définition 4.3.

CHAPITRE 6

Simulation d'un CA par des machines spatiales

La motivation de Y. Feldman et de E. Shapiro quant à la création des machines spatiales repose sur les observations suivantes : une information binaire n'occupe qu'une quantité finie et minimale de l'espace et le transfert des informations est borné par la vitesse de la lumière. De plus, le modèle des PRAM est beaucoup trop puissant pour modéliser de façon fidèle le comportement des machines parallèles réelles. Dans le rapport de recherche [**14**] et dans l'article [**15**], les auteurs définissent le modèle des machines spatiales et discutent de son aptitude à modéliser les machines parallèles réelles.

Ils montrent aussi que les machines spatiales sont capables de simuler n'importe quelle machine de Turing et qu'elles sont donc capables d'universalité. C'est en nous inspirant de leur simulation que nous avons mimé le fonctionnement des automates cellulaires de deux manières différentes. Dans un premier temps, lors de la conférence Renpar de 1994 [**32**] et dans un article de TSI de 1995 [**34**], nous nous sommes limités au cas de machines spatiales à deux processeurs (appelées séquentielles) puis nous nous sommes interessés au cas des machines spatiales à plusieurs processeurs, dites parallèles, simulation que nous avons présentée lors de la conférence EuroPar en 1999 [**38**].

1. Définition des machines spatiales

Nous rappelons ici la définition «informelle» des machines spatiales [**15**] comme elles sont présentées dans un article des CACM. Pour plus de précisions, on pourra se reporter au rapport de recherches correspondant [**14**].

Le modèle des machines spatiales est défini usuellement dans un espace tridimensionnel discret. Le modèle calcule selon des intervalles de temps discrets et chaque composant effectue son calcul de manière synchrone. Un nombre fini de cases de l'espace est occupé par des processeurs qui peuvent échanger des messages selon le principe suivant :

- chacune des cases de la grille -qui est soit vide soit occupée- comporte six bits de communication, un par direction et à valeurs dans l'ensemble $\{0, 1, B\}$;
- au cours de chaque étape, chaque bit de communication est recopié dans la direction désignée sur le bit de communication correspondant de la case adjacente.

Chaque processeur représente une unité de contrôle similaire à celle d'une machine de Turing. Un processeur est donc constitué d'un nombre fini d'états et d'une fonction de transition. A chaque pas de calcul des machines spatiales, en fonction de l'état interne du processeur et de la valeur des bits de communication, l'unité de contrôle effectue une des opérations suivantes :

- changer d'état interne ;
- changer la valeur des bits de communication avant qu'ils ne soient recopiés ;
- déplacer le processeur vers une des cases adjacentes de l'espace.

Il est interdit aux processeurs d'occuper une même case de l'espace ou d'occuper une case qui l'était au début du pas de calcul. Dans ces cas, le processeur reste dans la case qu'il occupait et effectue les autres opérations normalement. Un nombre fini de bits de communication sont désignés comme bits d'entrée. Au début du calcul, tous les processeurs sont dans le même état initial et tous les bits de communication, à l'exception des bits d'entrée, sont dans l'état blanc *B*. La valeur des bits d'entrée peut changer arbitrairement au cours du

calcul. Une case de l'espace est désignée comme cellule de terminaison. On dit qu'un calcul est terminé quand un processeur entre dans un état final dans une case de terminaison. Le résultat du calcul est alors défini sur un ensemble de cellules distinguées, les cellules de sortie.

1.1. Propriétés des machines spatiales. Nous rappelons ici quelques propriétés des machines spatiales qui sont énoncées et démontrées dans [**15**].

On dit qu'une machine spatiale est séquentielle si

– un seul processeur est "actif" i.e. qui traite effectivement les données et ne sert pas seulement de mémoire ou de réflecteur[1] ;
– pour toute description instantanée -sauf peut-être pour la première- il ne peut y avoir qu'un seul message (ou suite non vide de bits de communication) qui circule ;

Si, de plus, le processeur actif d'une machine spatiale séquentielle est immobile, on dit que la machine spatiale séquentielle est stationnaire. Il a été montré dans [**15**] que si la puissance de calcul des machines spatiales séquentielles est équivalente à celle des machines de Turing, les machines spatiales séquentielles stationnaires ne sont pas plus puissantes que les automates finis. La puissance de calcul des machines spatiales stationnaires n'est pas accrue par l'augmentation du nombre de processeurs actifs s'ils sont également stationnaires. De manière analogue, il a été montré que les machines spatiales parallèles ont la même puissance de calcul que les machines spatiales séquentielles. Cependant, l'augmentation du nombre de processeurs d'une machine spatiale séquentielle stationnaire permet d'accélérer les calculs :

Théorème 6.1 (Feldman, Shapiro [**15**])**.** Une machine spatiale stationnaire d-dimensionnelle ($d = 1,2,3$) à p processeurs a un facteur d'accélération en $\Omega(p^{1+(1/d)})$ par rapport à une machine spatiale séquentielle stationnaire.

Dans le cas des machines spatiales non stationnaires, le problème du facteur d'accélération demeure ouvert.

2. Addition par une machine spatiale

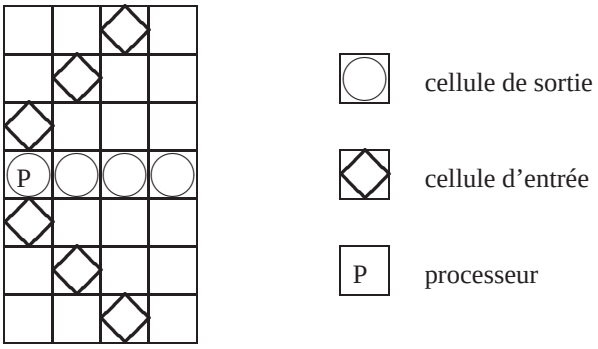

Fig. 1. État initial de la machine spatiale.

La machine spatiale qui permet d'additionner deux entiers écrits en binaire nous fournit un bon exemple de fonctionnement d'une machine spatiale. Dans ce cas, nous n'avons besoin que d'un unique processeur que nous représenterons soit par un P soit par son état, comme indiqué dans les figures 1 et 2.

La configuration initiale de la machine spatiale est décrite figure 1 où nous avons représenté les cellules d'entrées, les cellules de sorties et la position initiale du processeur P qui est dans son état initial. Observons que les cellules d'entrée peuvent être choisies

[1]Le processeur sert de réflecteur lorsqu'il ne fait que rediriger les bits de communication qu'il reçoit.

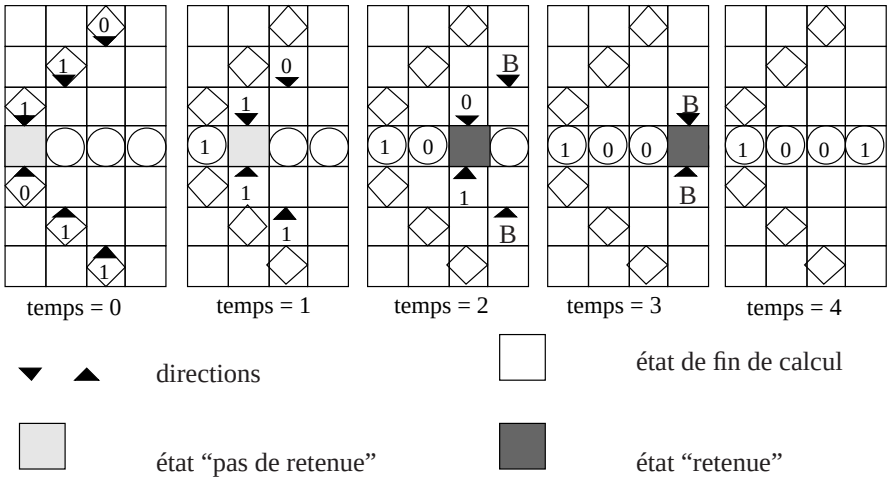

Fig. 2. Configurations successives de la machine spatiale.

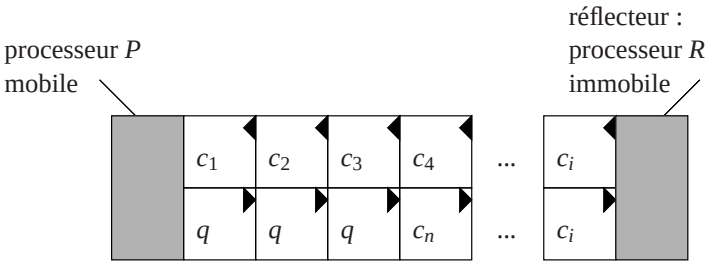

Fig. 3. Configuration initiale de la machine spatiale.

n'importe où dans le plan. Les trois autres états du processeur sont portés sur la figure 2 où nous avons décrit les étapes successives de l'addition binaire de $3+6$, représentés avec le bit de poids faible à gauche. Nous avons donc l'opération suivante, $110+011=1001$ où la machine spatiale calcule la somme bit à bit en reportant la retenue dans la case voisine de droite. Les directions imposées aux bits de communication sont représentées par les petits triangles noirs.

Nous obtenons la proposition 6.1 dont la démonstration est immédiate.

Proposition 6.1 (Martin [**34**])**.** L'addition de deux entiers x et y est réalisée par une machine spatiale à un seul processeur en temps $T(n) = n+1$ pour $n = \max\{\lfloor \log_2 x \rfloor, \lfloor \log_2 y \rfloor\} + 1$

3. Simulation d'un CA par une machine spatiale séquentielle

Nous décrivons dans cette section la simulation d'un automate cellulaire par une machine spatiale séquentielle. Par souci de simplicité, nous avons pris quelques libertés vis-à-vis du modèle des machines spatiales en accroissant l'ensemble des états des bits de communication. Leur nouvel ensemble d'états est alors le produit cartésien de $\{0,1,B\}$ par Q, ensemble des états de l'automate cellulaire à simuler. Nous appelons machine spatiale généralisée une telle machine. Nous avons le résultat suivant :

Théorème 6.2 (Martin [**34**])**.** Il existe une machine spatiale séquentielle généralisée capable de simuler un automate cellulaire unidimensionnel. De plus, si le temps de calcul de l'automate cellulaire est $T(n)$, sa simulation par une machine spatiale généralisée sera de complexité en temps $T^2(n)+(n-1)T(n)$ pour une entrée de taille n.

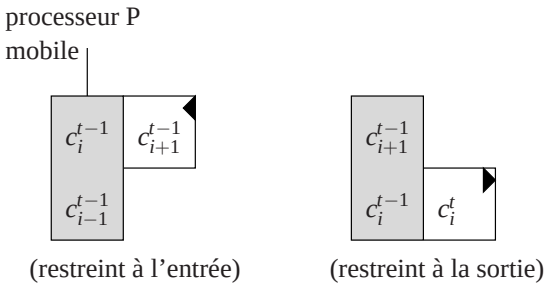

Fig. 4. Machine spatiale simulant un automate cellulaire.

Preuve. On suppose que la configuration initiale de l'automate cellulaire à simuler est donnée par la séquence $qq\{c_i^0\}_{i=1,\ldots,n}qq$ où l'indice supérieur représente le numéro de l'itération de l'automate cellulaire. La lettre q qui borde la séquence de configuration sert à délimiter la configuration d'entrée et symbolise des états dits quiescents. La configuration initiale de la machine spatiale séquentielle qui simule l'automate cellulaire est conforme à la description de la figure 3 où la configuration initiale de l'automate cellulaire a été repliée sur elle-même.

Nous utilisons une technique similaire à celle du mime d'une machine de Turing par une machine spatiale séquentielle présentée dans [**15**]. Nous utiliserons deux processeurs, P et R. Le premier, actif, comporte une unité de contrôle dont le rôle est de simuler le fonctionnement de la fonction de transition de l'automate cellulaire. Le second, R est inactif et ne sert qu'à réfléchir les bits de communication (cf. fig. 3).

Le processeur P mémorise alors les valeurs nécessaires pour effectuer la transition d'une cellule, par exemple c_i. Alors, pour envoyer la nouvelle valeur c_i^t, il a besoin d'avoir en mémoire les valeurs c_{i-1}^{t-1}, c_i^{t-1} et de recevoir la valeur c_{i+1}^{t-1}. Après avoir appliqué la fonction de transition à ces valeurs et renvoyé la valeur c_i^t, il mémorise les états c_i^{t-1}, c_{i+1}^{t-1} et se prépare à recevoir la valeur c_{i+2}^{t-1}. La figure 4 illustre cette simulation dont le dessin est restreint à la valeur reçue à l'instant $t-1$ et à la valeur émise à l'instant t.

La taille des configurations d'un automate cellulaire peut croître de deux cellules par unité de temps, c'est-à-dire que les deux états quiescents qui bordent la configuration de part et d'autre peuvent passer dans un état non quiescent. Pour simuler cette croissance de la configuration de l'automate cellulaire, il faut déplacer le processeur P d'une case vers la gauche à chaque fois qu'on rencontre une transition de la forme $\delta(q,q,\beta) = \alpha \in Q$ ou $\delta(\beta,q,q) = \alpha \in Q$ pour un état $\beta \neq q$. Dans le premier cas, il y aura émission d'un q puis de la valeur α et dans le second cas, il y aura émission de la valeur α puis de q.

La complexité de cette simulation est $n = |\{c_i\}|$ fois plus lente. Elle correspond en effet à une séquentialisation des transitions de l'automate cellulaire. On passe en effet de n unités de contrôle à une seule. De plus, la taille des configurations étant croissante, le temps de la simulation est borné supérieurement par $n.T(n) + \sum_{i=1}^{T(n)-1} 2.i = T^2(n) + (n-1)T(n)$. □

3.1. Deux lemmes d'équivalence. Dans cette partie, nous justifions l'équivalence entre les machines spatiales séquentielles généralisées et celles définies par Y. Feldman et E. Shapiro dans [**15**].

Lemme 6.1 (Martin [**34**])**.** Toute machine spatiale séquentielle dont l'ensemble des états des bits de communication a été étendu à n symboles peut être simulée par une machine spatiale séquentielle dont l'ensemble des états est restreint à l'ensemble $\{0,1,B\}$ avec une perte de temps de l'ordre du logarithme de n .

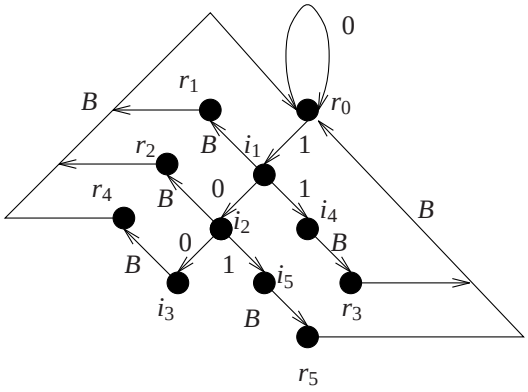

Fig. 5. Fonctionnement de la machine spatiale binaire en réception.

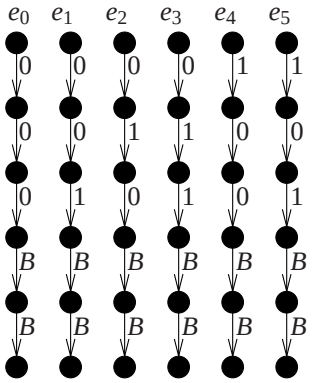

Fig. 6. Fonctionnement de la machine spatiale binaire en émission.

Preuve. Soit Σ_n l'ensemble étendu des états des bits de communication. Sans perte de généralité, on peut identifier Σ_n à $\{1,2,\ldots,n\}$. On simule le fonctionnement de la machine spatiale étendue par une machine spatiale binaire de la façon suivante : les états étendus seront codés en binaire et séparés par deux symboles B. La machine spatiale binaire va simuler exactement le fonctionnement de la machine spatiale étendue en recevant et en émettant les états codés bit à bit. Puisque les machines spatiales reçoivent et envoient simultanément des valeurs, nous avons besoin d'états produits. Les états de la machine spatiale binaire seront donc de la forme (état, bit reçu, bit émis). Nous allons décrire le fonctionnement de la machine spatiale binaire pour $n = 5$:

Réception : La machine spatiale binaire fonctionne à la manière d'un automate fini à la réception. Elle cumule les bits reçus jusqu'à recevoir le symbole séparateur BB. La figure 5 illustre la réception. Les états de type r_j indiquent que la valeur binaire j a été reçue avant le symbole B. Les états de type i_k sont des états internes au fonctionnement. Nous illustrons le fonctionnement de cet automate par l'exemple suivant : supposons que la machine spatiale reçoive successivement les bits $011BB$. La suite de transitions effectuée sera la suivante : $(r_0,0) \vdash (r_0,1) \vdash (i_1,1) \vdash (i_4,B) \vdash (r_3,B) \vdash (r_0,\ldots)$. La machine spatiale a donc reçu la valeur 3 et s'est repositionnée sur l'état initial r_0.

Émission : Le fonctionnement de la machine spatiale à l'émission est analogue à son fonctionnement à la réception. Nous pouvons construire un automate fini qui va émettre la valeur voulue comme décrit par la figure 6. Les états de type e_j indiquent que la valeur

j doit être émise en binaire suivie des deux blancs de séparation. Nous n'avons pas mentionné les états internes par souci de clarté. Nous les numérotons ensuite e_j^k avec $1 \leq k \leq 5$ qui donne le nombre de valeurs émises. Nous illustrons le fonctionnement de cet automate par l'exemple suivant : supposons que la machine spatiale étendue ait à émettre la valeur décimale 5. La machine spatiale binaire qui la simule va passer dans l'état e_5 et enverra successivement les symboles de la suite $101BB$ qui code la valeur 5. Avec la convention ci-dessus pour la numérotation des états, nous aurons la suite de transitions suivante : $(e_5,1) \vdash (e_5^1,0) \vdash (e_5^2,1) \vdash (e_5^3,B) \vdash (e_5^4,B)$.

Fonctionnement général : Maintenant que nous avons décrit le fonctionnement de la machine spatiale binaire en émission et en réception, nous pouvons indiquer son fonctionnement général. Pour ce faire, nous donnons la suite de transitions de la machine spatiale binaire qui simule les deux transitions de la machine spatiale généralisée indiquées ci-dessous :

$$\begin{array}{ll} ? & \vdash (q_4,2,\text{arrêt}) \\ (q_4,3) & \vdash (q_5,5,\text{arrêt}) \end{array}$$

qui correspond au cas où la machine spatiale généralisée émet la valeur 2, reçoit la valeur 3 et va émettre ensuite la valeur 5. La suite de transitions de la machine spatiale binaire sera alors la suivante. On rappelle que, dans ce cas, les états de la machine spatiale binaire sont des triplets d'états :

$$\begin{array}{ll} ((q_4,r_0,e_2),0) & \vdash ((q_4,r_0,e_2^1),0,\text{arrêt}) \\ ((q_4,r_0,e_2^1),1) & \vdash ((q_4,r_0,e_2^2),1,\text{arrêt}) \\ ((q_4,i_1,e_2^2),1) & \vdash ((q_4,r_0,e_2^3),0,\text{arrêt}) \\ ((q_4,i_4,e_2^3),B) & \vdash ((q_4,r_0,e_2^4),B,\text{arrêt}) \\ ((q_4,r_3,e_2^4),B) & \vdash ((q_5,r_0,e_5),B,\text{arrêt}) \end{array}$$

Il nous reste maintenant à décrire la simulation de la machine spatiale généralisée lorsque le processeur mobile se déplace. Nous allons donner la simulation de la suite de transitions dans le cas où le processeur mobile est à gauche et se déplace d'une case vers la gauche.

$$\begin{array}{ll} ? & \vdash (q_4,2,\text{gauche}) \\ (q_4,3) & \vdash (q_5,5,\text{arrêt}) \\ (q_5,2) & \vdash ? \end{array}$$

La simulation du déplacement d'un processeur est illustrée par la figure 7 où nous avons reporté à chaque unité de temps l'état du processeur mobile à gauche. Observons que la valeur 3 en binaire est reçue avant le déplacement et que, lors de ce mouvement, le processeur entre dans l'état particulier g_i^k qui signifie que le processeur va passer dans l'état i à l'issue du déplacement vers la gauche et qu'il s'est déjà décalé de k cases vers la gauche ($1 \leq k \leq 5$). Notons aussi que lors du mouvement deux "cases" vides ont été créées, ce qui correspond aussi à ce qui se passe dans le cas de la machine spatiale généralisée.

Nous ne décrirons pas le cas où le processeur se déplace d'une case vers la droite qui peut être déduit aisément du cas précédent. □

Dans la preuve du lemme 6.1, nous avons introduit la notion d'ensemble d'états produit. Le lemme suivant nous permet d'affirmer qu'il y a équivalence entre les machines spatiales dont l'ensemble des états est le produit cartésien de quelques ensembles finis et les machines spatiales habituelles. Ainsi,

Lemme 6.2 (Martin [**34**])**.** L'ensemble des états des machines spatiales peut être égal au produit cartésien de quelques ensembles finis.

Preuve. Nous utilisons un argument classique pour montrer l'équivalence entre les deux ensembles d'états. Lorsque l'ensemble des états est égal au produit cartésien de k ensembles finis, il suffit de les renuméroter au moyen d'une bijection de $\mathbb{N}^k \to \mathbb{N}$. □

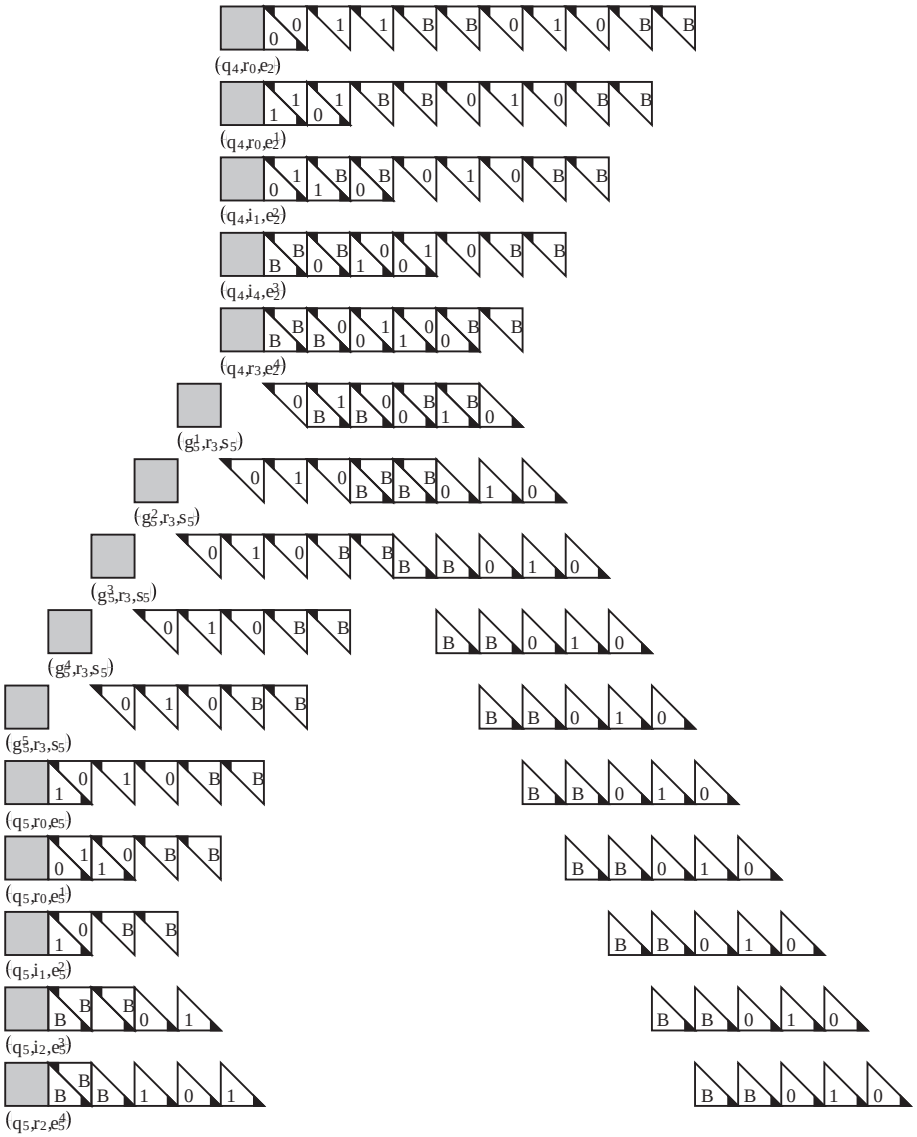

Fig. 7. Simulation du déplacement d'un processeur.

3.2. Le résultat principal. Nous pouvons maintenant énoncer notre résultat principal qui corrobore l'énoncé du théorème 6.2 en autorisant la simulation d'un automate cellulaire par une machine spatiale séquentielle selon la définition de [**15**].

Théorème 6.3 (Martin [**32, 34**])**.** Une machine spatiale séquentielle est capable de simuler un automate cellulaire unidimensionnel à k états. De plus, si le temps de calcul de l'automate cellulaire est $T(n)$, sa simulation par une telle machine spatiale sera de complexité en temps $(T^2(n)+(n-1)T(n))(\lfloor \log_2 k \rfloor + o(1))$.

La preuve du théorème 6.3 peut être aisément obtenue à l'aide du théorème 6.2 et des lemmes 6.1 et 6.2. On en déduit :

Corollaire 6.1. Il existe une machine spatiale séquentielle qui est universelle au sens du calcul.

4. Simulation d'un CA par une machine spatiale parallèle

Dans cette section, nous proposons une famille de machines spatiales parallèles pour simuler le fonctionnement d'un automate cellulaire.

Théorème 6.4 (Martin [**38**])**.** Il existe une machine spatiale qui peut simuler le fonctionnement de tout automate cellulaire totalistique à k états ; la simulation d'une transition de l'automate cellulaire se fait en temps $O(k\lceil \log_2 k\rceil)$.

Preuve. Soit A un automate cellulaire totalistique à k états et S la machine spatiale parallèle qui en simule le fontionnement. On utilisera la mesure de complexité $\mathrm{CLARGE}_A(x)$ qui compte le nombre maximal de cellules non quiescentes utilisées par A au cours de son calcul. Cette mesure de complexité, introduite dans [**4**] se définit au moyen de $\mathbb{C} = Q^{\mathbb{Z}}$, l'ensemble des configurations de A et de $F : \mathbb{C} \to \mathbb{C}$, la fonction de transition globale induite par la fonction locale.

$$\mathrm{CLARGE}_A(x) = \max\left\{|x|, \max_{i,j}\{|i-j| : \exists t,t' : F^t(c_0)_i \neq q \text{ et } F^{t'}(c_0)_j \neq q\}\right\}$$

$\mathrm{CLARGE}_A(x)$ nous donne le nombre de processeurs dont S va avoir besoin pour simuler le fonctionnement de A.

Chaque processeur de S va avoir un comportement similaire. On note P_i, le processeur qui simule le fonctionnement de la cellule i. On associe deux réflecteurs à chaque P_i (fig. 8) ; celui du haut mémorise l'état de la cellule tandis que celui du bas mémorise la table de transition.

Les états de l'automate cellulaire vont être codés en binaire sur η bits comme :

$$\eta = \begin{cases} \lceil \log_2 k\rceil & \text{si } k \equiv 0 \mod 2 \\ \lceil \log_2 k\rceil + 1 & \text{sinon} \end{cases}$$

La fonction de transition totalistique sera aussi codée en binaire comme la suite $f(0), f(1), f(2), \ldots, f(l)$. La longueur totale du code de la fonction de transition de A sera $l\eta$. Pour des raisons de synchronisation, on ajoute deux blancs supplémentaires dans le code de la fonction de transition.

La distance entre P_i et le réflecteur de l'état est $\eta/2$ et celle entre P_i et le réflecteur de la fonction de transition, $(l+2)/\eta$ et celle entre deux processeurs, η. Ce sont les distances qui vont permettre de calculer la complexité en temps de la simulation.

Chaque processeur P_i va avoir un comportement analogue :

- il reçoit la suite de bits en provenance de P_{i-1}, P_{i+1} et du réflecteur d'état et calcule leur somme s ;
- il attend l'image de s qui lui est renvoyée par le réflecteur de la fonction de transition ;
- il remplace la suite de bits qui code l'état par la nouvelle valeur ;
- sur réception du premier blanc du code de la fonction de transition, il transmet aux processeurs adjacents le contenu de la suite de bits qui code son propre état.

Il faut au préalable avoir mémorisé la configuration initiale de l'automate cellulaire, comme dans la figure 8.

Le temps nécessaire pour que S simule une transition de A est donc $(l+2)\eta$ où $l = O(k)$. □

Observons que, dans le résultat précédent, on démontre qu'une machine spatiale peut simuler un automate cellulaire à n cellules et l transitions. Ainsi, on peut construire une famille de machines spatiales capable de simuler le fonctionnement de tout automate cellulaire, indépendamment de n et de l. La description d'une machine spatiale de la famille pour n et l fixés, se calcule aisément de manière parallèle ; on obtient ainsi une famille uniforme de machines spatiales.

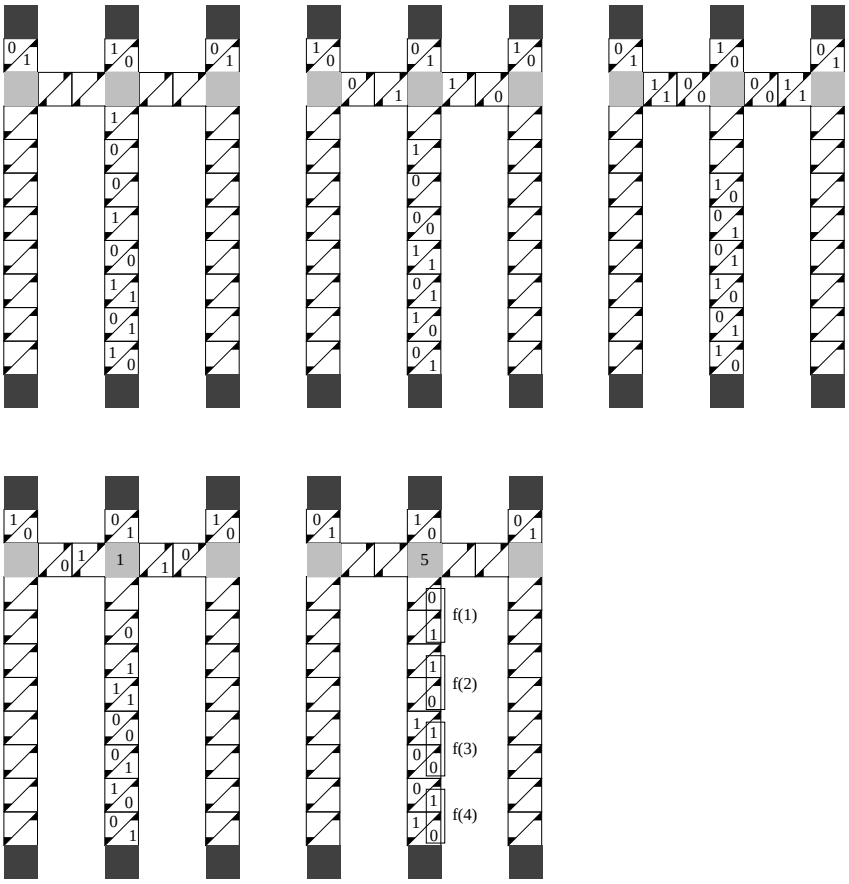

Fig. 8. Comportement du processeur P_i.

5. Conséquences et conclusions

Dans ce chapitre, nous avons montré que les machines spatiales sont capables de simuler les automates cellulaires de différentes manières.

Dans le cas de la simulation séquentielle, nous avons simulé le fonctionnement d'un automate cellulaire arbitraire au moyen d'une machine spatiale, construisant ainsi une machine spatiale séquentielle universelle par simulation au sens de la définition 4.3. Dans le cas de notre simulation parallèle, il nous faut connaître le plus grand nombre de cellules utilisées par l'automate cellulaire au cours du calcul ; la machine spatiale reçoit donc en entrée le code de l'automate cellulaire, sa configuration initiale et le nombre de cellules utilisées pour le calcul. Il faut ensuite construire explicitement la machine spatiale qui va simuler le fonctionnement de cet automate cellulaire et nous illustrons ainsi la construction d'une machine spatiale universelle par construction au sens de la définition 4.5.

En tout état de cause, nous obtenons le fait que dans l'ensemble des machines spatiales -tant séquentielles que parallèles- il en existe qui sont universelles au sens des automates cellulaires.

CHAPITRE 7

Conclusion de la première partie

1. Résumé des différentes simulations

Dans cette section, nous résumons les différentes simulations d'automates cellulaires que nous avons proposées et nous les mettons en perspective avec d'autres résultats de même nature. Nous avons reporté dans la figure 1 des résultats de simulation entre différents modèles de calcul soit séquentiels soit parallèles.

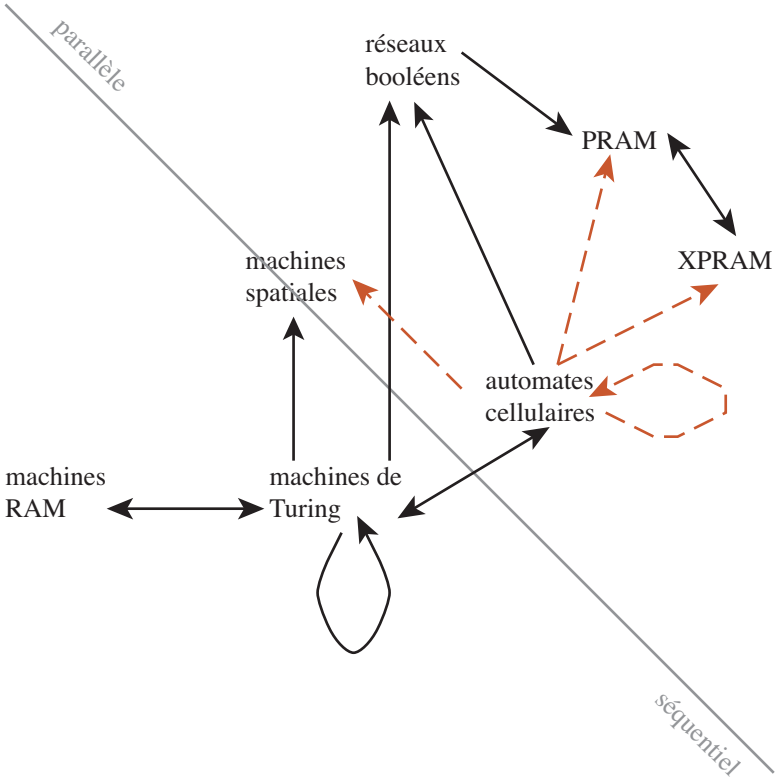

Fig. 1. Simulations entre les modèles de calcul.

Dans la figure 1, le graphe orienté dont les sommets sont des modèles de calcul et les arcs des relations de simulation résume quelques résultats classiques de simulation comme, par exemple, celle d'une machine RAM par une machine de Turing [**27**]. La convention est la suivante : le modèle à l'extrémité d'un arc est celui qui réalise la simulation. Par exemple, l'arc d'origine les machines de Turing et d'extrémité les machines spatiales signifie qu'il existe une simulation effective des machines de Turing par les machines spatiales [**15**]. Les résultats que nous avons présentés dans les chapitres précédents sont symbolisés par des traits interrompus courts. Observons que deux modèles comportent une boucle –au sens des graphes orientés–. Il s'agit dans ce cas d'une simulation au sein du même modèle

qui symbolise l'existence d'une machine intrinsèquement universelle au sens de la définition 4.3. L'équivalence entre deux modèles de calcul symbolisée par une double flèche est obtenue soit par deux simulations soit au moyen de l'isomorphisme de Rogers que nous avons rappelé dans le théorème 2.1.

Sur le graphe de la figure 1, on retrouve tous les résultats que nous avons proposés dans les chapitres précédents : l'existence d'un automate cellulaire intrinsèquement universel ; le fait que toute fonction calculable par automate cellulaire l'est aussi par des PRAM, par des XPRAM ou par des machines spatiales. De plus, par le biais des la composition de la relation de simulation, on obtient aussi que toute fonction Turing-calculable l'est aussi par automate cellulaire, par les PRAM ou les XPRAM ainsi que par les machines spatiales.

2. Machines parallèles CA-universelles

modèle	nb. proc.	complexité asymptotique	type univ.	réf.
CA	$nk\log k$	$k\log k$	intrinsèque	[**29, 33**]
CREW-PRAM	$p=n$	1	simulation	[**31**]
XPRAM	$p\geq n$	1	simulation	[**37**]
XPRAM	$p<n$	$T(n)(\frac{(T(n)+n)}{p}+pg+\ell)$	simulation	[**37**]
SM	2	$(T^2(n)+n\cdot T(n))\log k$	simulation	[**32, 34**]
SM	$n+2\cdot T(n)$	$k\log k$	construction	[**38**]

Tab. 1. Résumé des différentes simulations.

Le tableau 1 résume et référence l'ensemble des simulations d'automates cellulaires totalistiques à n cellules et à k états qui sont rappelées dans ce mémoire. Nous décrivons ce tableau colonne par colonne. La première colonne décrit le modèle qui simule un automate cellulaire totalistique. Nous avons abrégé automate cellulaire en CA et machines spatiales en SM. Dans la seconde colonne, nous indiquons le nombre d'unités de contrôle qui est requis pour réaliser la simulation. Dans le cas d'automates cellulaires, il s'agit du nombre de cellules et dans le cas des (X)PRAM ou des machines spatiales, du nombre de processeurs (noté p). La troisième colonne donne la complexité asymptotique de la simulation. Nous indiquons également dans la quatrième colonne par quel type de simulation il est possible d'obtenir une machine universelle pour le modèle considéré. Enfin, dans la dernière colonne, nous indiquons l'article –ou les articles– dans lequel on peut trouver une preuve de la simulation.

Considérons par exemple la troisième ligne du tableau 1. Celle-ci résume le résultat de simulation d'un automate cellulaire totalistique à n cellules par une XPRAM qui dispose de $p=n$ processeurs. On lit dans la colonne 3 que cette simulation se fait en temps constant et dans la colonne 4 qu'on peut déduire de cette simulation l'existence d'une XPRAM universelle par simulation i.e. que la XPRAM est capable de simuler le fonctionnement de n'importe quel automate cellulaire sous forme totalistique.

Observons que dans le tableau 1, nous ne faisons pas référence à l'universalité par hérédité. Celle-ci est plus restrictive que l'universalité par simulation. En effet, pour ce type de simulation, on ne peut pas simuler directement un automate cellulaire arbitraire. On ne devrait simuler que le fonctionnement d'un seul automate cellulaire universel. C'est ce type de simulation qui est employé pour construire certaines petites machines de Turing universelles. La simulation d'un automate cellulaire au moyen d'une machine spatiale parallèle requiert la connaissance du nombre maximum de cellules actives au cours du calcul de l'automate cellulaire. Le modèle des machines spatiales ne permet pas de «réveiller» des processeurs au cours du calcul. C'est la raison pour laquelle notre simulation ne donne lieu qu'à une machine spatiale parallèle universelle par construction. Mais, avec la donnée du nombre de cellules actives de l'automate cellulaire, les machines spatiales parallèles sont en mesure de simuler le fonctionnement de n'importe quel automate cellulaire.

3. Conséquences et conclusion

Nous citons dans cette section quelques conséquences relativement directes de nos résultats d'universalité obtenus par la simulation d'automates cellulaires. Nous commençons par un résultat d'indécidabilité obtenu par réduction du problème de l'arrêt. En bornant le temps de calcul, on obtient ensuite un résultat de P-complétude. Enfin, nous montrons qu'il est possible de définir la complexité de Kolmogorov au moyen des automates cellulaires.

3.1. Indécidabilité de l'occurrence d'un état.

Théorème 7.1. Soit $C = (Q, \delta)$ un automate cellulaire, $\hat{q} \in Q$ un état distingué et $c \in \mathbb{C}$ une configuration initiale. Il n'est pas possible de décider si $\hat{q}$ apparaît au cours du calcul de C sur l'entrée c.

On procède par réduction du problème de l'arrêt. L'automate cellulaire simule le fonctionnement d'une machine de Turing. D'après notre classification des modes de simulation, l'automate cellulaire est universel par hérédité. Le théorème est également valide pour tout modèle de calcul qui possède une machine universelle par hérédité ou par simulation.

3.2. P-complétude de l'occurrence d'un état au temps t. Pour le calcul parallèle, la P-complétude joue un rôle analogue à la NP-complétude pour les calculs nondéterministes. Elle correspond aux problèmes qui ne peuvent pas être résolus efficacement par un algorithme parallèle autrement appelés intrinsèquement séquentiels [**18**].

Deux problèmes jouent pour la P-complétude un rôle comparable à celui que la satisfiabilité joue pour la NP-complétude :

Le problème générique de la simulation de machine
Donnée : Une chaîne x, la description $\overline{M}$ d'une machine de Turing M, et un entier t en unaire.

Question : M accepte-elle x en moins de t pas de calcul ?

Le problème de la valuation des circuits
Donnée : $\overline{\alpha}$ codage standard du circuit α, d'entrées $x_1, \ldots, x_n$ et de sortie y.

Question : La sortie y de α est-elle vraie pour les entrées $x_1, \ldots, x_n$?

Le problème suivant est P-complet et la démonstration est immédiate en s'inspirant du livre de R. Greenlaw, H. Hoover et W. Ruzzo [**18**] :

Donnée : c la configuration initiale d'un automate cellulaire, T une borne sur le temps en unaire, q un état, i une cellule.

Question : La cellule i est-elle dans l'état q au temps T ?

La démonstration repose aussi sur la simulation d'une machine de Turing quelconque par l'intermédiaire de la simulation d'une machine de Turing universelle. Comme pour le résultat d'indécidabilité, celui-ci est vrai dès lors que notre ordinateur possède un programme universel par hérédité ou par simulation.

3.3. Complexité de Kolmogorov. Dans cette section, nous illustrons la construction d'une complexité de Kolmogorov au moyen des automates cellulaires –ou de tout autre modèle de calcul convenable–. Rappelons que la complexité de Kolmogorov mesure la complexité d'un objet combinatoire x. De manière intuitive, la complexité de Kolmogorov correspond à la taille du plus petit programme qui permet de décrire x. Nous nous inspirons ici de la définition de la complexité de Kolmogorov donnée dans le livre de M. Li et P. Vitányi [**24**].

On identifie l'objet fini x avec son indice dans une énumération adéquate des objets et on considère la classe des fonctions récursives partielles comme des méthodes de description. Le problème de la description d'un objet consistant en un entier naturel sera

ramené à celui de trouver un programme donné sous un codage en binaire. Pour cela, on se donne une énumération effective des automates cellulaires $P_0, P_1, P_2, \ldots$ comme dans la section 3.1. Nous utilisons le fait qu'une énumération effective des automates cellulaires induit une énumération effective des fonctions récursives partielles $\varphi_0, \varphi_1, \varphi_2, \ldots$ telle que $\forall i$, P_i calcule φ_i.

Définition 7.1 (Li et Vitanyi [**24**])**.** Soit x, y et p des entiers naturels. Toute fonction récursive partielle φ avec la donnée de p et y telle que $\varphi(\langle p,y\rangle) = x$ est une description de x. La complexité $C_\varphi(x)$ relative à y est :

$$C_\varphi(x|y) = \begin{cases} \min\{l(p) : \varphi(\langle p,y\rangle) = x\} \\ \infty \text{ si } p \text{ n'existe pas} \end{cases}$$

où $l(p)$ représente la longueur de p. On dit que p est un programme pour calculer x par φ étant donnée y.

Le théorème 7.2 est la clé de voûte du développement de toute la théorie de la complexité de Kolmogorov.

Théorème 7.2 (Li et Vitanyi [**24**])**.** Il existe une fonction récursive partielle φ_0 telle que, pour toute fonction récursive partielle φ, il existe une constante c_φ telle que $C_{\varphi_0}(x|y) \leq C_\varphi(x|y) + c_\varphi$ pour tout $x, y \in \mathbb{N}$.

Preuve. Soit φ_{univ} la fonction récursive partielle calculée par notre automate cellulaire intrinsèquement universel P_U. Rappelons que, par définition, l'automate cellulaire P_U sur l'entrée $(n, \langle y, p\rangle)$ simule le fonctionnement de P_n sur l'entrée $\langle y, p\rangle$. On a $\varphi_{\text{univ}}(n, \langle y, p\rangle) = \varphi_n(\langle y, p\rangle)$. D'où, $C_{\varphi_{\text{univ}}}(x|y) \leq C_\varphi(x|y) + c_\varphi$ où $c_\varphi = 2 \cdot l(n) + 1$. □

Nous sommes alors en mesure de définir la complexité de Kolmogorov avec les automates cellulaires comme modèle de référence.

Définition 7.2 (Li et Vitanyi [**24**])**.** On choisit une fonction universelle de référence φ_{univ} dont on omettra l'indice. On définit alors la complexité de Kolmogorov $C(x|y) = C_{\varphi_{\text{univ}}}(x|y)$. Cette fonction universelle particulière φ_{univ} est la fonction de référence pour C. On se donne également un automate cellulaire particulier P_U qui calcule φ_{univ} et on l'appelle automate cellulaire de référence.

Par la définition de [**24**], on sait que la complexité de Kolmogorov ne dépend que de l'objet. Mais, avec notre définition, il semble qu'elle dépende également de l'énumération. La proposition suivante nous indique que, étant donnés deux énumérations reliées par un va-et-vient effectif la complexité de Kolmogorov d'un objet est la même, à une constante additive près. Il s'agit d'une conséquence de l'isomorphisme de Rogers.

Proposition 7.1 (Li et Vitanyi [**24**])**.** Soit $\varphi_0, \varphi_1, \ldots$ et ψ_0, ψ_1 deux énumérations distinctes des fonctions récursives partielles telles qu'il existe f et g, deux fonctions récursives partielles telles que $\psi_i = \varphi_{f(i)}$ et $\varphi_i = \psi_{g(i)}$. Soit $C(x|y)$ la complexité de Kolmogorov relative à φ_{univ} et $C'(x|y)$ celle relative à ψ_{univ}. Alors, pour tout objet x, $|C(x|y) - C'(x|y)| \leq k$.

En conséquence de quoi, on déduit :

Corollaire 7.1 (Martin [**35**])**.** La complexité de Kolmogorov définie au moyen des machines de Turing est, à une constante additive près, la même que celle définie au moyen des automates cellulaires.

Et il en serait de même pour les autres modèles de calcul que nous avons étudié dans les chapitres précédents.

Deuxième partie

Automates cellulaires sur graphes de Cayley

CHAPITRE 8

Définitions, notations et premières simulations

Les automates cellulaires sur des graphes de Cayley ont été introduits pour démontrer une généralisation du théorème de Moore et Myhill par [**26, 6**] puis étudiés en tant que tels par Zs. Róka [**51, 54**]. Le but est de décrire le réseau de connexion des automates au moyen du graphe d'un groupe finiment présenté. Il faut pour cela que le réseau de connexions soit régulier. On peut également définir des automates sur des réseaux hyperboliques au moyen des graphes de Cayley, comme dans [**49**].

Dans ce chapitre, nous rappelons les définitions des groupes et des graphes de Cayley. Nous montrons ensuite comment on peut définir un automate cellulaire sur un graphe de Cayley. Enfin, nous rappelons deux résultats de Zs. Róka [**51, 52, 53**] qui permettent de simuler le fonctionnement d'un réseau hexagonal d'automates sur des tores d'automates. Ces résultats nous servent pour illustrer la notion d'automate cellulaire défini sur un graphe de Cayley et nous seront utiles dans le chapitre 9.

1. Groupes et graphes de Cayley

On se donne $\Gamma = \langle \mathcal{G} \,|\mathcal{R}\, \rangle$ un groupe finiment présenté où $\mathcal{G}$ est l'ensemble des générateurs du groupe et $\mathcal{R}$ celui de ses relations [**10**]. Soit également $\mathcal{S} \subset \Gamma$ un sous-ensemble ne contenant pas 1_Γ, l'identité du groupe. Lorsque cela sera nécessaire, la loi interne sera notée par un point.

Exemple. Soit le groupe $\Gamma = \langle \alpha, \beta | \alpha\beta = \beta\alpha \rangle$ de générateurs $\mathcal{G} = \{\alpha, \beta\}$ et d'ensemble de relations $\mathcal{R} = \{\alpha\beta = \beta\alpha\}$. Il définit $(\mathbb{Z}^2, \cdot)$.

On définit alors $G = (V, E)$, le graphe de Cayley du groupe Γ en identifiant V, les sommets de G, aux éléments de Γ et en ajoutant une arête entre x et y si et seulement si $y = s_i \cdot x$ pour $x \in \Gamma$ et $s_i \in \mathcal{S}$.

Exemple. Soit le groupe $\Gamma_R^l = \langle w | w^l = 1_\Gamma \rangle$. Son graphe de Cayley définit un anneau de l sommets.

Exemple. Soit le groupe $\Gamma_T^{n,m} = \langle \alpha, \beta | \alpha^n = 1_\Gamma, \beta^m = 1_\Gamma, \alpha\beta = \beta\alpha \rangle$. Son graphe de Cayley définit un tore à $n \times m$ sommets.

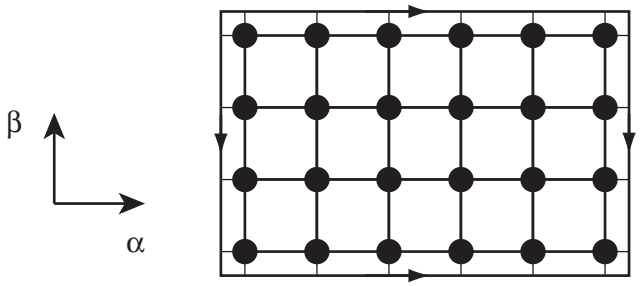

Fig. 1. Un tore à 6×4 sommets.

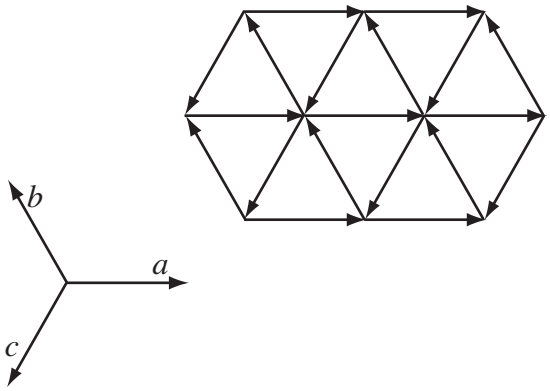

Fig. 2. Voisinage dans un réseau hexagonal.

Exemple. Soit le groupe $\Gamma_H^{r,p,q} = \langle a,b,c : ab = ba, ac = ca, bc = cb, abc = 1, a^r c^{1-q} = 1, b^{p-1}c^{-q} = 1\rangle$. Son graphe de Cayley définit un réseau hexagonal à $r \times p \times q$ sommets.

On définit un chemin de G comme une suite des sommets adjacents ou, de manière équivalente comme la donnée d'un sommet de départ et d'un mot sur l'alphabet $\mathcal{S}$. Un cycle est un chemin ayant au moins trois sommets tel que le dernier sommet soit adjacent au premier ou, alternativement, la donnée d'un sommet de départ et d'un mot sur $\mathcal{S}$ qui se réduit à 1_Γ. Un cycle est hamiltonien s'il passe par tous les sommets de G une et une seule fois. Un cycle hamiltonien du tore est dit alterné si le mot sur $\mathcal{S}$ qui l'étiquette est de la forme $(\alpha\beta)^\star(\alpha+\varepsilon)$ ou $(\beta\alpha)^\star(\beta+\varepsilon)$ ou bien $(\alpha^{-1}\beta^{-1})^\star(\alpha^{-1}+\varepsilon)$ ou encore $(\beta^{-1}\alpha^{-1})^\star(\beta^{-1}+\varepsilon)$ avec ε qui dénote le mot vide. La figure 3 décrit un cycle hamiltonien du tore à 6×4 sommets. Si on part du sommet en bas à gauche, le mot qui étiquette ce cycle est de la forme $(\alpha\beta)^{12}$.

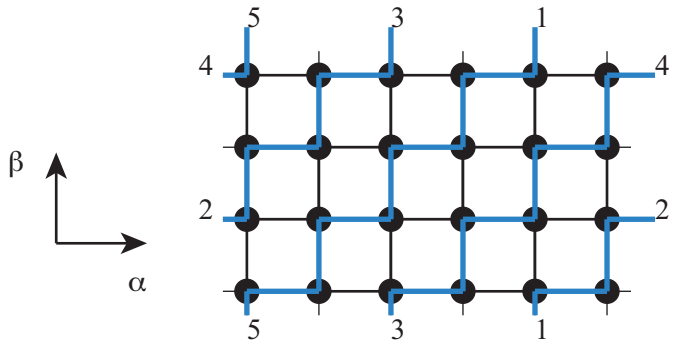

Fig. 3. Un cycle hamiltonien du tore à 6×4 sommets.

2. Automates cellulaires sur des graphes de Cayley

On définit un automate cellulaire sur un graphe de Cayley $G = (\Gamma, \mathcal{S})$ comme un quadruplet $\mathcal{A} = (Q, G, N, \delta)$ où Q est l'ensemble fini des états de la cellule ; $G = (\Gamma, \mathcal{S})$ est le graphe du groupe Γ, N le voisinage tel que $N = \mathcal{S} \cup \{1_\Gamma\}$ et $\delta : Q^{|N|} \to Q$ la fonction locale de transition qui met à jour l'état des cellules en accord avec l'état des cellules du voisinage défini par N à l'instant précédent. On associe une cellule de l'automate à chacun des sommets du graphe. Nous ne considérerons ici que le voisinage dit de von Neumann qui est défini comme l'ensemble des sommets à distance au plus un pour la distance de Hamming.

Rappelons que la distance de Hamming compte le nombre de positions sur lesquelles deux mots finis de même longueur diffèrent.

Une configuration de l'automate cellulaire est une application qui associe un état à chacune des cellules de l'automate. L'ensemble de toutes les configurations de l'automate cellulaire est noté $\mathbb{C} = Q^{\Gamma}$. Il est alors possible de définir la fonction globale de transition qui est définie au moyen de la fonction locale δ par :

$$\forall c \in \mathbb{C}, \forall x \in \Gamma, \Delta(c)(x) = \delta(c(x), c(s_1.x), \ldots, c(s_{|\mathcal{S}|}.x))$$

On peut alors définir des automates cellulaires sur des anneaux au moyen du groupe Γ_R^l, sur des tores au moyen du groupe $\Gamma_T^{n,m}$ et sur des réseaux hexagonaux grâce au groupe $\Gamma_H^{r,p,q}$. Il est possible de définir des automates cellulaires sur d'autres groupes comme dans [**16, 52**]. Pour d'autres exemples de graphes de Cayley qui permettent de définir des automates cellulaires, on peut se réferer à [**11**].

3. Premières simulations

Nous rappelons rapidement les notations ; on considère un automate cellulaire défini sur le groupe $\Gamma_H^{r,p,q} = \langle a,b,c : ab = ba, ac = ca, bc = cb, abc = 1_\Gamma, a^r c^{1-q} = 1_\Gamma, b^{p-1}c^{-q} = 1_\Gamma \rangle$. Son graphe de Cayley définit un réseau hexagonal à $r \times p \times q$ sommets. Zs. Róka en simule le fonctionnement par un automate cellulaire défini tout d'abord sur le groupe $\Gamma_T^{n,m,k} = \langle a,b,c | a^n = 1_\Gamma, b^m = 1_\Gamma, c^k = 1_\Gamma, ab = ba, ab = ba, bc = cb \rangle$ dont le graphe de Cayley définit un tore à $n \times m \times k$ sommets puis par un automate cellulaire sur le groupe $\Gamma_T^{n,m} = \langle a,b | a^n = 1_\Gamma, b^m = 1_\Gamma, ab = ba \rangle$ dont le graphe de Cayley définit un tore à $n \times m$ sommets.

3.1. Simulation d'un réseau hexagonal par un réseau torique de dimension 3. Cette section présente le théorème de Róka [**52**] qui simule le fonctionnement d'un automate cellulaire défini sur un réseau hexagonal avec un automate cellulaire défini par un réseau torique de dimension 3.

Théorème 8.1 (Róka [**52**])**.** Tout automate cellulaire sur $\Gamma_H^{r,p,q}$ peut être simulé par un automate cellulaire sur $\Gamma_T^{n,m,k}$ avec $n = |\alpha_1(p+r-1) - \beta_1 p|$, $m = |\alpha_2(p-1) - \beta_2(p+q-1)|$, $k = |\alpha_3(1-p-r) + \beta_3 p|$ et $(p-1)\alpha_1 = \text{ppcm}\ (p-1, p+q-1)$, $(p+r-1)\alpha_2 = \text{ppcm}\ (p+r-1, p)$, $(-r)\alpha_3 = \text{ppcm}\ (r, q-1)$, $(p+q-1)\beta_1 = \text{ppcm}\ (p-1, p+q-1)$ and $p\beta_2 = \text{ppcm}\ (p+r-1, p)$, $(q-1)\beta_3 = \text{ppcm}\ (r, q-1)$.

Preuve. Soit τ_H un automate cellulaire sur $\Gamma_H^{r,p,q}$ de configuration initiale $c^0_{\tau_H}$. Montrer l'existence de τ_{3D} un automate cellulaire sur $\Gamma_T^{n,m,k}$ qui simule le fonctionnement de τ_H revient à démontrer que $c^0_{\tau_H}$ est périodique dans toutes les directions données par les générateurs. En d'autres termes, qu'il existe trois entiers n, m et k tels que $a^n = 1_\Gamma, b^m = 1_\Gamma$ et $c^k = 1_\Gamma$.

Soit $v_h \in \Gamma_H^{r,p,q}$ et $v_{3D} \in \Gamma_T^{n,m,k}$. La configuration initiale de τ_{3D} est : $c^0_{\tau_{3D}}(v_{3D}) = c^0_{\tau_H}(v_h), c^0_{\tau_{3D}}(v_{3D}a^i) = c^0_{\tau_H}(v_h a^i)$ pour $1 \leq i < n, c^0_{\tau_{3D}}(v_{3D}b^j) = c^0_{\tau_H}(v_h b^j)$ pour $1 \leq j < m, c^0_{\tau_{3D}}(v_{3D}c^l) = c^0_{\tau_H}(v_h c^l)$ pour $1 \leq l < k$.

Le voisinage de $c^0_{\tau_H}$ est préservé dans $c^0_{\tau_{3D}}$. On définit alors τ_{3D} comme $Q_{3D} = Q_H$ et $\delta_{3D} = \delta_H$ qui simule τ_H sans perte de temps.

Montrons que c_{τ_H} est périodique dans toutes les directions données par les générateurs. Nous nous limiterons à la preuve pour le générateur a. La construction est analogue pour les deux autres générateurs. En décrivant les relations, on obtient $R = \{c = a^{-1}b^{-1}; a^r c^{1-p} = 1; b^{q-1}c^{-p} = 1\}$ qui définit un système de deux équations d'inconnue a :

$$\begin{cases} a^{p+r-1}b^{p-1} = 1_\Gamma \\ a^p b^{p+q-1} = 1_\Gamma \end{cases}$$

Soient $\alpha_1 = \frac{\text{ppcm}\,(p-1,q+p-1)}{p-1}$ et $\beta_1 = \frac{\text{ppcm}\,(p-1,q+p-1)}{q+p-1}$; d'où la valeur de a : $a^{\alpha_1(p+r-1)-\beta_1 p} = 1_\Gamma$.

Les autres valeurs du théorème sont obtenues en résolvant les systèmes obtenus par la description des relations sur le réseau pour les variables b et c et en définissant $\alpha_2, \alpha_3, \beta_2$ et β_3. □

3.2. Simulation d'un réseau hexagonal par un réseau torique de dimension 2. Cette section présente un corollaire du théorème 8.1. En observant que la troisième direction donnée par les générateurs dépend des deux autres, il est possible de simuler le fonctionnement d'un automate cellulaire sur un réseau torique par un automate cellulaire sur un tore de dimension deux :

Corollaire 8.1 (Róka [**52**])**.** Tout automate cellulaire sur $\Gamma_H^{r,p,q}$ peut être simulé sans perte de temps par un automate cellulaire sur $\Gamma_T^{n,m}$ avec $n = |\alpha_1(p+r-1) - \beta_1 p|$, $m = |\alpha_2(p-1) - \beta_2(p+q-1)|$, $(p-1)\alpha_1 = \text{ppcm}\,(p-1, p+q-1)$, $(p+r-1)\alpha_2 = \text{ppcm}\,(p+r-1, p)$, $(p+q-1)\beta_1 = \text{ppcm}\,(p-1, p+q-1)$ et $p\beta_2 = \text{ppcm}\,(p+r-1, p)$.

C'est grâce au corollaire 8.1 que nous allons pouvoir simuler le fonctionnement d'un automate cellulaire sur un réseau hexagonal par un automate cellulaire sur un anneau dans le chapitre 9.

CHAPITRE 9

Simulation d'un tore d'automates par un anneau d'automates

1. Introduction

Nous présentons dans ce chapitre différents résultats de simulation d'un tore de $n \times m$ automates par un anneau d'automates. La difficulté revient à conserver une copie des cellules du voisinage du tore sur l'anneau. Pour cela, nous avons recours à la technique des calques que nous avons introduite pour la construction d'un automate cellulaire universel [**33**].

Le premier résultat, publié en 1997 dans [**36**], propose une première solution au moyen de quatre calques sans requérir de condition particulière sur les valeurs respectives de n et de m.

Le second résultat, soumis pour publication [**41**] et obtenu en collaboration avec C. Peyrat, permet de restreindre le nombre de calques à un minimum de trois pour des valeurs particulières de n et de m. Dans ce dernier résultat, nous montrons également que notre simulation est optimale en temps.

Le premier résultat est de nature essentiellement algorithmique tandis que le second demande plus de travail combinatoire.

2. Simulation d'un tore d'automates par un anneau à 4 calques

Le théorème 9.1 présente la simulation d'un tore de $n \times m$ automates qu'on nommera par commodité des nœuds dans la suite. Remarquons qu'il n'est pas nécessaire de faire des hypothèses sur les valeurs relatives de m et de n et que la perte de temps relative à la simulation est sous-linéaire. En particulier, si $n = m$ et pour un anneau à N automates de complexité en temps $T(N)$, la complexité en temps de la simulation sur un anneau de N automates sera $\sqrt{N}.T(N)$. Nous énonçons ci-dessous le théorème 9.1 :

Fig. 1. Une cellule de l'anneau.

Théorème 9.1 (Martin [**36**])**.** Tout automate cellulaire sur $\Gamma_T^{n,m}$ (un tore de $m \times n$ automates) de complexité en temps $T(mn)$ peut être simulé par un automate cellulaire à quatre calques sur Γ_R^{nm} en temps $\min\{m,n\}T(mn)$.

Preuve. Soit le tore d'automates $\mathcal{A}_T = (Q_T, G_T, N_T, \delta_T)$ où G_T représente le graphe du groupe $\Gamma_T^{n,m}$ et N_T le voisinage de von Neumann de dimension deux qu'il s'agit de simuler.

On construit l'anneau d'automates $\mathcal{A}_R = (Q_R, G_R, N_R, \delta_R)$ où G_R représente le graphe du groupe Γ_R^{nm} et N_R le voisinage de von Neumann de dimension un. L'ensemble Q_R des états de $\mathcal{A}_R$ est obtenu comme $Q_R = (Q_T)^3 \times Q_{\text{Sync}}$. On définit ainsi quatre calques comme

il est décrit dans la figure 1 où (C) représente le calque central (où on réalisera effectivement la simulation), (U) une copie du nœud «nord» du tore, (D) une copie du nœud «sud» du tore et (Sync) un calque de synchronisation.

Il s'agit tout d'abord de décrire le «plongement» d'une configuration du tore dans une configuration de l'anneau (et inversement, comment il est possible de «lire» la configuration du tore sur celle de l'anneau).

On fixe donc $\nu = \alpha^x\beta^y$ un nœud arbitraire du tore de coordonnées (x,y) et on décrit $\mu = a^k$, la k^{e} cellule de l'anneau d'automates. On définit T une application qui attribue un état à un nœud du tore et trois applications U,C et D qui fournissent respectivement les états inscrits sur les calques $(U),(C)$ et (D) des cellules de l'anneau.

$T(\nu)$ est copié sur trois cellules différentes (et sur les trois calques) de l'anneau :

$$\begin{array}{lcl} U(a^{m(x+1)+y}) & = & T(\nu) \\ C(a^{mx+y}) & = & T(\nu) \\ D(a^{m(x-1)+y}) & = & T(\nu) \end{array}$$

Inversement, pour retrouver le nœud du tore correspondant à la cellule k de l'anneau, on utilisera :

$$T\left(\alpha^{(k \bmod n)}\beta^{(k|n)}\right) \quad = \quad C(\mu)$$

(ci-dessus, $j|k$ dénote la division entière de j par k).

Le calque de synchronisation est construit de la manière suivante : on place un général sur le calque Sync des cellules $0,(m-1),m,(2m-1),2m,\ldots,(nm-1)$. Le fonctionnement de l'anneau sur ce calque est régi par la fonction de transition solution du problème des fusiliers à deux généraux que nous avons rappelé dans le lemme 3.3.

Le scénario de la simulation est alors le suivant :

(i) le contenu du calque (C) est recopié vers la droite de proche en proche sur le calque (U) tant qu'il n'apparaît pas de «top» de synchronisation (i.e. que l'état contenu dans le calque (Sync) n'est pas l'état F de feu) ;

(ii) le contenu du calque (C) est recopié vers la gauche de proche en proche sur le calque (D) tant qu'il n'apparaît pas de «top» de synchronisation (i.e. que l'état contenu dans le calque (Sync) n'est pas l'état F de feu) ;

(iii) à l'apparition de l'état F sur le calque de synchronisation (Sync), l'information recopiée est à sa place et les cellules de l'anneau d'automate peuvent simuler le fonctionnement des nœuds du tore d'automates. Une fois cette opération réalisée, on procède à nouveau aux étapes (i) et (ii) précédentes.

Nous ne décrirons pas plus avant la fonction de transition de l'automate. Les opérations élémentaires de recopie ont été décrites dans ma thèse de doctorat [**30**].

□

La figure 2 décrit le plongement d'un tore à 4×6 automates (à droite) dans un anneau de 24 automates (à gauche). Le tore est découpé suivant ses deux courbes de Jordan. Les cellules de l'automate sont à l'intersection de la grille toroïdale qui «pave» le tore. Nous avons choisi un nœud particulier (numéroté 31) et son voisinage dans la croix en pointillés. L'anneau d'automates est représenté de manière analogue et nous retrouvons la cellule qui simule le nœud 31 et son voisinage dans l'anneau, comme décrit dans la preuve du théorème 9.1.

La dynamique de la simulation d'un tore à 4×3 automates par un anneau de 12 cellules est décrit dans la figure 3. Chaque cellule est conforme à la description de la figure 1. Nous avons collé les cellules adjacentes pour simplifier la représentation. Nous avons également omis les informations de simulation ainsi que le contenu du calque principal dans un souci de lisibilité. Seuls les contenus des deux calques auxiliaires sont représentés pour

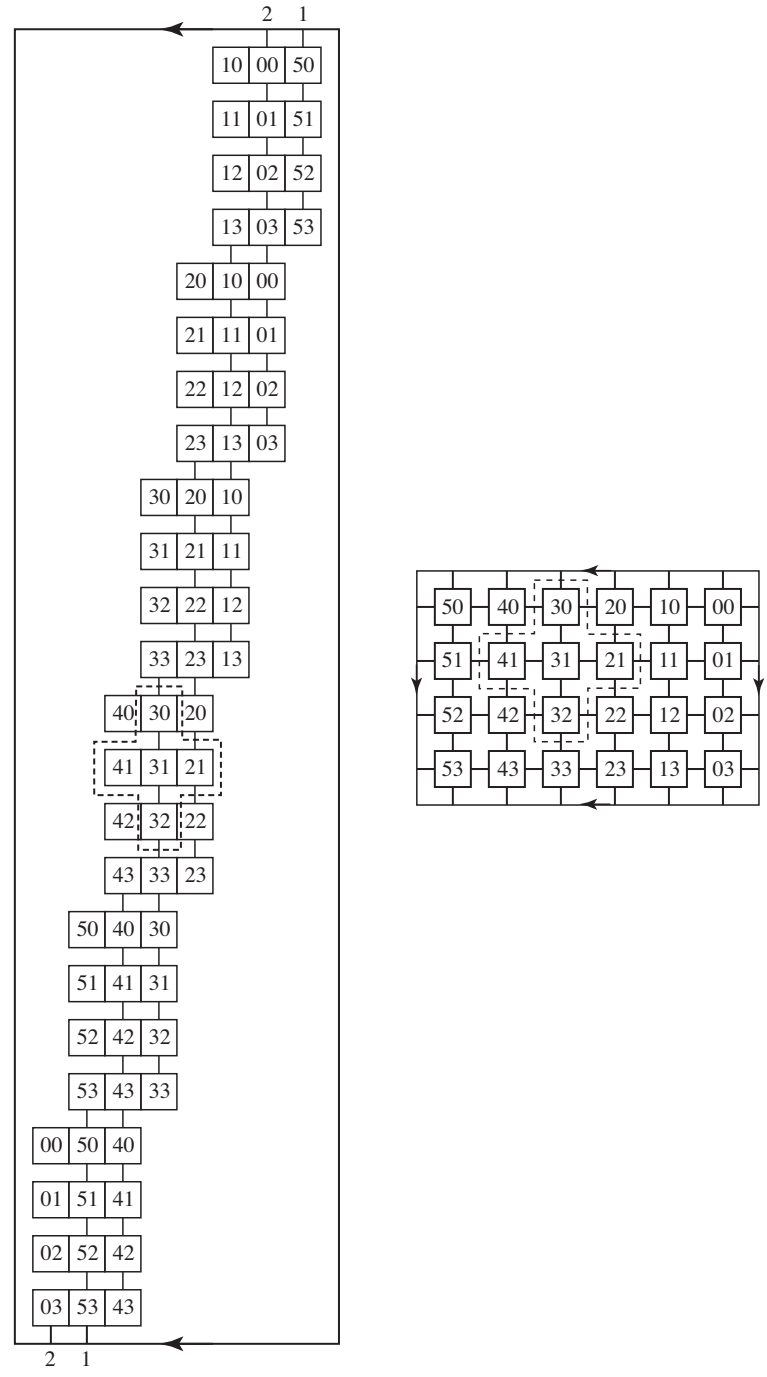

Fig. 2. Plongement d'un tore de 4×6 automates dans un anneau de 24 automates.

mettre en valeur le déplacement des informations. Observons que la simulation d'une transition des nœuds du tore requiert 3 unités de temps, ce qui est conforme avec l'énoncé du théorème 9.1.

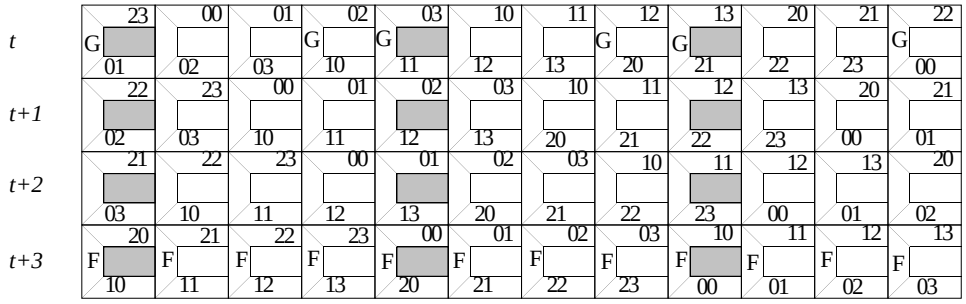

Fig. 3. Evolution d'un anneau de 12 automates.

3. Simulation d'un tore d'automates par un anneau à 3 calques

Nous nous sommes ensuite demandé s'il était possible d'améliorer les choses en jouant sur différents paramètres : la complexité de la simulation et le nombre des états. En fait, c'est en s'interrogeant sur la complexité de la simulation qu'il nous est venu l'idée de minimiser le nombre des calques et donc le nombre des états.

Dans cette section, nous proposons de démontrer le théorème 9.2 :

Théorème 9.2 (Martin, Peyrat [**41**])**.** Tout automate cellulaire sur $\Gamma_T^{n,m}$ (un tore de $m \times n$ automates) de complexité en temps $T(mn)$ peut être simulé par un automate cellulaire sur Γ_R^{nm} si et seulement si m et n sont tels que pgcd $(m,n) = 2$. Si $n \equiv 2 \mod m$, le nombre de calques est alors minimal et égal à trois et le temps de simulation est $\Theta((n+m).T(mn))$.

Cette démonstration est plus combinatoire que la précédente ; elle passe par la décomposition du tore en cycle hamiltonien alterné puis par un arrangement plus astucieux des cellules. Pour arriver à la décomposition en cycle hamiltonien, il faut fixer des conditions sur les valeurs relatives de m et de n. Enfin, pour calculer la complexité de la simulation, il faut mesurer la distance entre la cellule qui simule un nœud et sa copie sur l'anneau.

On introduit quelques notations. On dénote le tore de $n \times m$ automates par $\mathcal{A}_T = (Q_\mathrm{T}, G_T^{n,m}, N, \delta_\mathrm{T})$ où $G_T^{n,m}$ est le graphe du groupe $\Gamma_T^{n,m}$ et $N = \{1_\Gamma, \alpha, \alpha^{-1}, \beta, \beta^{-1}\}$ représente le voisinage de von Neumann de dimension deux. L'anneau de nm cellules sera noté $\mathcal{A}_R = (Q_\mathrm{R}, G_R^{nm}, N, \delta_\mathrm{R})$ où G_R^{nm} est le graphe du groupe Γ_R^{nm}, $N = \{1_\Gamma, w, w^{-1}\}$ représente le voisinage de von Neumann et $\delta_\mathrm{R} : (Q_\mathrm{R})^3 \to Q_\mathrm{R}$ la fonction (locale) de transition.

L'ensemble des états de l'anneau Q_R est formé de trois calques : $Q_\mathrm{R} = (Q_\mathrm{T})^2 \times Q_\mathrm{Sync}$. On représentera donc les états de l'anneau par un triplet (M, C, Sync) où (M) est le calque principal (ou sera faite la simulation), (C) le calque contenant la copie d'un nœud et (Sync) le calque de synchronisation.

On note $N(\nu)$ les voisins du nœud $\nu = \alpha^i\beta^j$ de coordonnées i et j sur le tore et $C(\nu)$ le contenu du calque C de la cellule qui représente le nœud ν.

3.1. De la nécessité d'un cycle hamiltonien alterné.

La proposition suivante exprime une condition nécessaire pour le théorème 9.2.

Proposition 9.1 (Martin, Peyrat [**41**])**.** Si un automate cellulaire sur $\Gamma_T^{n,m}$ (un tore de $m \times n$ automates) peut être simulé par un automate cellulaire sur Γ_R^{nm} avec un nombre minimum de calques, alors il existe un cycle hamiltonien alterné dans $G_T^{n,m}$, le graphe du groupe $\Gamma_T^{n,m}$.

Preuve. Soit ν un nœud arbitraire du tore et soient V_1 et V_2 ses voisins sur l'anneau. On peut exprimer la contrainte de voisinage sur le tore comme $N(\nu) = \{\nu\alpha, \nu\alpha^{-1}, \nu\beta, \nu\beta^{-1}\} \subset \{V_1, C(V_1), V_2, C(V_2), C(\nu)\}$.

Fait 9.1. V_1 et V_2 appartiennent à $N(\nu)$.

Par l'absurde, supposons que $V_1 \notin N(\nu)$ –mais V_1 est un voisin de ν sur l'anneau–. Alors, $C(\nu)$ et $C(V_1)$ doivent être dans l'intersection $C(\nu) \cap C(V_1)$. Sans perte de généralité, on peut supposer que $C(\nu) = \nu\alpha$, $V_1 = \nu\alpha\beta$, $C(V_1) = \nu\beta$. Il s'ensuit que $\{V_2, C(V_2)\} = \{\nu\beta^{-1}, \nu\alpha^{-1}\}$. Ainsi ni $C(V_2)$ ni $C(\nu)$ n'appartiennent à $N(V_2)$ sur le tore, une contradiction.

On supposera sans perte de généralité que $V_1 = \nu\alpha^{-1}$.

Fait 9.2. Si $V_1 = \nu\alpha^{-1}$, alors $V_2 \neq \nu\alpha$.

Par l'absurde, supposons $V_2 = \nu\alpha$. Sans perte de généralité, on peut supposer que $C(V_1) = \nu\beta$. Cela implique que $C(\nu)$ doit être dans $N(V_1)$ mais ne peut être égal à $\nu\beta^{-1}$. En conséquence, $C(V_2) = \nu\beta^{-1}$ et $C(\nu)$ doit également être dans $N(V_2)$ tout en étant différent de ν, ce qui est impossible.

On supposera donc que $V_2 = \nu\beta$ et on s'intéresse à V_3, l'autre voisin de V_2 sur l'anneau.

Fait 9.3. Si $V_1 = \nu\alpha^{-1}$ et $V_2 = \nu\beta$, alors $V_3 = \nu\alpha\beta$.

Par les deux faits précédents, V_3 ne peut prendre que l'une des deux valeurs : $V_3 = \nu\alpha\beta$ ou $V_3 = \nu\alpha^{-1}\beta$. Supposons que $V_3 = \nu\alpha^{-1}\beta$ et considérons $N(V_2) = \{\nu, V_3, \nu\beta^2, \nu\alpha\beta\}$. $C(\nu)$ ne peut être égal ni à $\nu\beta^2$ ni à $\nu\alpha\beta$ car ces deux nœuds ne sont ni dans $N(V_1)$ ni dans $N(\nu)$ et que V_1 et ν n'ont pas de voisin commun. Ainsi, $\{\nu\beta^2, \nu\alpha\beta\} = \{C(V_2), C(V_3)\}$. Mais alors $N(V_3) \cap \{C(V_2), C(V_3)\} = \varnothing$, une contradiction. □

On numérote les nœuds $(\nu_i)_{1 \leq i \leq nm}$ en fonction de leur position sur le tore. Sans perte de généralité et par les faits précédents, on peut choisir $\nu_1 = 1_\Gamma$, $\nu_2 = \alpha$, $\nu_3 = \alpha\beta$ et $\nu_4 = \alpha^2\beta$. On montre facilement par récurrence que $\nu_i = \alpha^{\lceil \frac{i-1}{2} \rceil}\beta^{\lfloor \frac{i-1}{2} \rfloor}$.

3.2. Valeurs convenables pour m et n. Nous décrivons ici une condition nécessaire et suffisante pour décomposer un tore d'automates sur un cycle hamiltonien alterné.

Proposition 9.2 (Martin, Peyrat [**41**])**.** Il existe un cycle hamiltonien alterné si et seulement si pgcd $(n, m) = 2$.

Cette proposition se démontre au moyen des faits suivants :

Fait 9.4. Lorsque le pgcd $(m, n) \neq 1$, un cycle hamiltonien alterné n'existe que si et seulement si pgcd $(m, n) = 2$.

Preuve. Notons d la valeur du pgcd (m, n). On a $d \geq 2$. En décrivant le cycle hamiltonien alterné, on a $\alpha^{\frac{nm}{d}}\beta^{\frac{nm}{d}} = 1_\Gamma$. Cela implique que $\frac{2nm}{d} = nm$, d'où l'on déduit $d = 2$. Réciproquement, si $d = 2$, il existe un cycle hamiltonien alterné défini par $\alpha^{\frac{nm}{2}}\beta^{\frac{nm}{2}} = 1_\Gamma$, $\alpha^i\beta^i$ et $\alpha^{i+1}\beta^i \neq 1_\Gamma$ si $i < \frac{nm}{2}$. □

Fait 9.5. Si le pgcd $(n, m) = 1$ avec nm pair, alors il n'existe pas de cycle hamiltonien alterné.

Preuve. Par contradiction, supposons que le pgcd $(n, m) = 1$ avec nm pair tel qu'il existe un cycle hamiltonien alterné. Alors $\alpha^{\frac{nm}{2}}\beta^{\frac{nm}{2}} = 1_\Gamma$; il s'ensuit que $\alpha^{\frac{nm}{2}} = 1_\Gamma$ et $\beta^{\frac{nm}{2}} = 1_\Gamma$. On en déduit que n et m doivent tous deux être pairs ; une contradiction. □

Fait 9.6. Si le pgcd $(n, m) = 1$ avec nm impair, alors il n'existe pas de cycle hamiltonien alterné.

Preuve. Par contradiction, supposons que le pgcd $(n, m) = 1$ avec nm impair tel qu'il existe un cycle hamiltonien alterné. Alors, $\alpha^{\frac{nm}{2}+1}\beta^{\frac{nm}{2}} = 1_\Gamma$; il s'ensuit que $\beta^{\frac{nm}{2}} = 1_\Gamma$. On en déduit que n doit être pair ; une contradiction. □

En combinant les faits précédents, on démontre la proposition 9.2.

3.3. Unicité. Maintenant que nous connaissons les valeurs adéquates de m et de n pour qu'il existe un cycle hamiltonien alterné, il faut trouver un placement adéquat des copies des nœuds pour reconstruire le voisinage du tore sur l'anneau. Ce placement est décrit dans la proposition suivante :

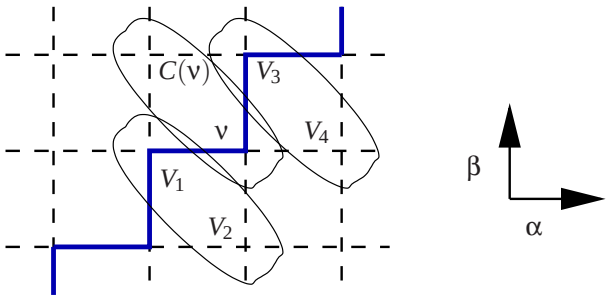

Fig. 4. Placement des cellules.

Proposition 9.3 (Martin, Peyrat [**41**])**.** Le placement des cellules dépend de l'existence d'un cycle hamiltonien alterné ; il est décrit par le mot

$$\xrightarrow{\beta}\begin{pmatrix}\nu(\alpha^{-1})\\ \downarrow \alpha\beta^{-1}\\ \nu(\beta)\end{pmatrix}\xrightarrow{\alpha}\begin{pmatrix}\nu\\ \downarrow \alpha^{-1}\beta\\ \nu(\alpha^{-1}\beta)\end{pmatrix}\xrightarrow{\beta}\begin{pmatrix}\nu\beta\\ \downarrow \alpha\beta^{-1}\\ \nu(\alpha)\end{pmatrix}\xrightarrow{\alpha}\begin{pmatrix}\nu(\beta\alpha)\\ \downarrow \alpha^{-1}\beta\\ \nu(\beta^2)\end{pmatrix}\xrightarrow{\beta}$$

pour $\nu = \alpha^i\beta^j$ la cellule qui simule le fonctionnement du nœud (i, j).

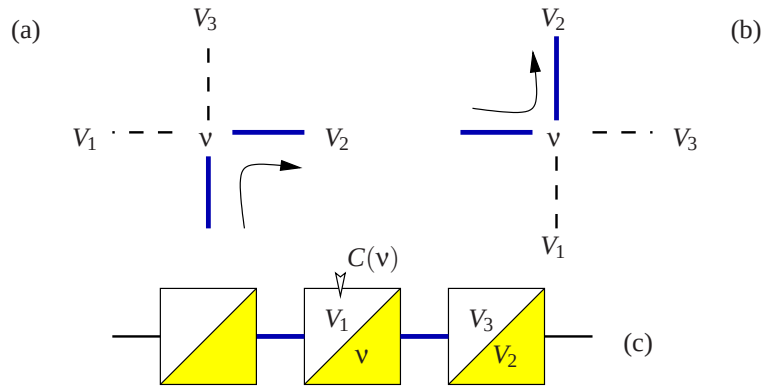

Fig. 5. Mauvais placement des copies.

Fait 9.7. Si $C(\nu) \in N(\nu)$, certains voisins du tore ne sont voisins ni sur l'anneau ni sur le calque des copies de l'anneau.

Preuve. Le placement des voisins du tore est décrit dans la figure 5 en (a) ou (b) selon le début du cycle hamiltonien alterné. Nous nous restreignons aux seuls voisins du nœud ν étiquetés V_1, V_2 et V_3. Si $C(\nu) \in N(\nu)$, alors ni V_1 ni V_3 ne peuvent être voisins de V_2 sur l'anneau. Ainsi, la cellule qui simule le fonctionnement de V_2 ne dispose pas des informations suffisantes pour effectuer la simulation d'une transition. □

Le placement adéquat des copies est décrit par la figure 4. Par le fait 9.7, comme V_2 n'est pas un voisin de V_1 et que V_4 n'est pas un voisin de V_3, cela implique que $C(\nu)$ doit

être dans le voisinage à la fois de V_1 et de V_3, comme décrit dans la figure 4. On peut donc supposer que le placement des cellules est bien de la forme donnée par la proposition 9.3.

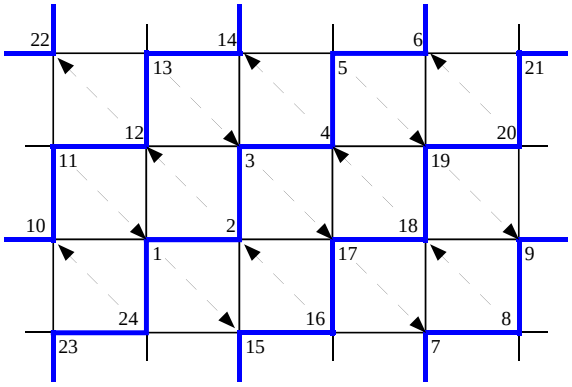

Fig. 6. Placement des nœuds d'un tore à 4×6 automates.

3.4. Minimiser la distance entre un nœud et sa copie. Afin de trouver le temps de simulation minimal, il nous reste à calculer la distance entre un nœud et sa copie sur l'anneau d'automates.

Proposition 9.4 (Martin, Peyrat [**41**])**.** Quand n et m vérifient pgcd $(n,m) = 2$, la distance entre un nœud et sa copie sur le cycle hamiltonien alterné est comprise entre $\min\{m+n, \frac{mn}{2}\}$ et $\max\{m+n, \frac{mn}{2}\}$. La valeur de cette distance est minimale lorsque $n \equiv 2 \mod m$ et vaut $2 \cdot n - 2$.

Preuve. Sans perte de généralité, on peut supposer que $n \geq m$. Nous voulons calculer la distance entre un nœud et sa copie sur l'anneau d'automates. En utilisant la description du cycle hamiltonien alterné et le placement des nœuds, on veut calculer la distance entre $\alpha^i\beta^j$ et $\alpha^i\beta^j\alpha^{-1}\beta$ ou $\alpha^i\beta^j\alpha\beta^{-1}$. En décrivant les conditions du cycle hamiltonien alterné pour une cellule à l'extremité d'un arc étiqueté par α (resp. β), on obtient : $\alpha^i\beta^j\beta^d\alpha^d = \alpha^i\beta^j\alpha^{-1}\beta$ (resp. $\alpha^i\beta^j\alpha^d\beta^d = \alpha^i\beta^j\alpha\beta^{-1}$). Remarquons que la distance $2d$ entre un nœud et sa copie ne dépend ni de i ni de j ; elle doit être paire puisque à la fois m et n sont pairs. Il s'ensuit que le graphe de Cayley est biparti. Pour une cellule à l'extrémité d'un arc étiqueté par α on obtient la relation :

$$\begin{cases} i+d \equiv i-1 \mod n \\ j+d \equiv j+1 \mod m \end{cases} \Leftrightarrow \begin{cases} d+1 = \lambda n \\ d-1 = \mu m \end{cases}$$

Pour une cellule à l'extrémité d'un arc étiqueté par β on obtient la relation :

$$\begin{cases} i+d \equiv i+1 \mod n \\ j+d \equiv j-1 \mod m \end{cases} \Leftrightarrow \begin{cases} d-1 = \lambda n \\ d+1 = \mu m \end{cases}$$

Pour une cellule à l'extrémité d'un arc étiqueté par α on a $2 = \lambda n - \mu m$. Puisqu'il existe un cycle hamiltonien alterné ssi pgcd $(n,m) = 2$, par Bezout, il existe a, b tels que $an - bm = 2$. La distance $2d = 2an - 2$ est minimale lorsque $a = 1$ i.e. $n \equiv 2 \mod m$. On en déduit que la distance minimale entre un nœud et sa copie sur l'anneau d'automates dans le cas d'un cycle hamiltonien alterné est $2n-2$.

La preuve pour une cellule à l'extrémité d'un arc étiqueté par β serait analogue. □

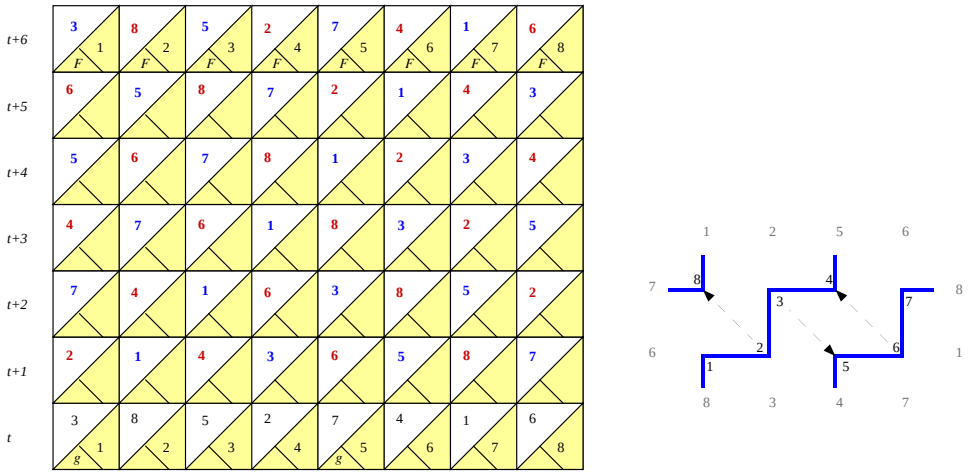

Fig. 7. Fonctionnement de la simulation d'un tore à 4×2 automates.

3.5. Fonctionnement de la simulation. Le calque de synchronisation est construit de la manière suivante : on place un général sur le calque Sync des cellules $1, n+1, \ldots, (m-1)n+1$. Le fonctionnement de l'anneau sur ce calque est régi par la fonction de transition solution du problème des fusiliers à un général (Lemme 3.2) qui fonctionne en temps $2an-1$.

Le scenario de la simulation est alors le suivant :

a) le contenu du calque principal des cellules d'indices pairs est recopié vers la droite de proche en proche tant que le «top» de synchronisation n'apparaît pas ;

b) le contenu du calque principal des cellules d'indices impairs est recopié vers la gauche de proche en proche tant que le «top» de synchronisation n'apparaît pas ;

c) à l'apparition de l'état F sur le calque de synchronisation, l'information est bien placée et les cellules peuvent simuler une transition du tore. Une fois cette opération réalisée, on effectue à nouveau les étapes a) et b).

Un algorithme pour déplacer des informations de manière synchrone est décrit dans [**33**].

Remarquons que cette simulation est optimale en temps. Cela provient du placement des copies des nœuds du tore sur les cellules de l'anneau. En effet, le tore est découpé en tranches de n cellules et la première cellule de chaque tranche devient un général pour synchroniser la tranche de n cellules. De plus la distance entre une nœud et sa copie est un entier modulo m.

4. Optimalité de la simulation par un anneau à 3 calques

Il est aisé de vérifier le théorème suivant qui énonce qu'on ne peut réaliser ce type de simulation avec moins de trois calques en synchronisant les recopies :

Théorème 9.3 (Martin, Peyrat [**41**])**.** Il n'existe pas d'automate cellulaire sur Γ_R^{nm} à deux calques qui puisse simuler le fonctionnement d'un automate cellulaire sur $\Gamma_T^{n,m}$.

5. Conséquences et conclusion

Nous présentons ci-dessous deux conséquences immédiates de la simulation d'un tore d'automates par un anneau. La première utilise itérativement le résultat de simulation pour simuler le fonctionnement de tout tore d'automates de dimension finie par un anneau. La seconde reprend la simulation de Róka d'un réseau hexagonal d'automates par un tore de dimension deux et la combine à la simulation d'un tore par un anneau.

Enfin, nous discutons notre simulation à trois calques d'un tore d'automates par un anneau d'automates.

5.1. Simulation de tores de dimension finie par un anneau d'automates. En utilisant itérativement le théorème 9.1, il est possible de simuler le fonctionnement d'un tore de dimension arbitraire mais finie d, par un anneau d'automates. C'est ce qu'exprime le théorème suivant :

Théorème 9.4 (Martin [**36**])**.** Tout automate cellulaire sur un tore de dimension d à $(k_1 \times \ldots \times k_d)$ nœuds et de complexité en temps $T(\prod_{i=1}^{d} k_i)$ peut être simulé par un anneau de $(\prod_{i=1}^{d} k_i)$ cellules en temps $\prod_{i=1}^{d-1} k_i T(\prod_{i=1}^{d} k_i)$.

5.2. Simulation d'un réseau hexagonal par un anneau d'automates. Dans cette section, nous combinons notre simulation d'un tore d'automates de la section 2 avec la simulation d'un automate cellulaire défini sur un réseau hexagonal de Róka présentée dans la section 3.2 du chapitre 8. Nous obtenons ainsi le corollaire :

Corollaire 9.1 (Martin [**39**])**.** Tout automate cellulaire sur $\Gamma_H^{r,p,q}$ peut être simulé par un automate cellulaire sur Γ_R^{nm} pour n et m qui vérifient $n = |\alpha_1(p+r-1) - \beta_1 p|$, $m = |\alpha_2(p-1) - \beta_2(p+q-1)|$, $(p-1)\alpha_1 = \text{ppcm}\,(p-1, p+q-1)$, $(p+r-1)\alpha_2 = \text{ppcm}\,(p+r-1, p)$, $(p+q-1)\beta_1 = \text{ppcm}\,(p-1, p+q-1)$ et $p\beta_2 = \text{ppcm}\,(p+r-1, p)$.

5.3. Remarques sur la simulation à trois calques. Notre première remarque sur cette simulation optimale porte sur la synchronisation. Synchroniser ou ne pas synchroniser est une question assez générale. Une croyance répandue est que tout algorithme d'automate cellulaire qui est réalisé au moyen d'une solution au problème de la synchronisation peut être réalisé sans synchronisation. A notre connaissance, personne ne sait si cette croyance est formulée comme théorème, ni même si une quelconque preuve existe. La simulation que nous proposons dans la preuve du théorème 9.2 utilise fortement une solution au problème de la synchronisation à un général. En fait, l'utilisation de la synchronisation s'est imposé naturellement à partir du moment où la distance entre un nœud et sa copie dans le cas optimal valait précisément le temps nécessaire à la synchronisation. Cependant, il est possible de se passer de celle-ci en utilisant des signaux (voir par exemple [**13**] à ce sujet) et en déplacant les copies dans le sens opposé. Une rapide description de cette simulation est la suivante : il faut partager l'anneau d'automates en deux segments de même taille, initier deux signaux qui partent du milieu et qui zig-zaguent entre le milieu et les «bords» pour déclencher l'émission des copies. Cependant, pour continuer ce scenario, il faut également avoir comme donnée la distance entre un nœud et sa copie. Cette simulation sans synchronisation, bien que possible, est beaucoup plus difficile à expliquer et à mettre en œuvre que celle que nous décrivons ici.

Bibliographie

[1] J. Albert and K. Culik II. A simple universal cellular automaton and its one-way and totalistic version. Complex Systems, 1 :1–16, 1987.

[2] M.A. Arbib. The universal Turing machine, chapter From universal Turing machines to self-reproduction, pages 177–189. Oxford Science Publications, 1988.

[3] R. Balzer. An 8 state minimal time solution to the firing squad synchronization problem. Information and Control, 10 :22–42, 1967.

[4] H. Ben-Azza. Automates cellulaires et pavages vus comme des réseaux booléens. PhD thesis, Université Claude Bernard (Lyon I), 1995.

[5] A. Borodin. On relating time and space to size and depth. SIAM journal on computing, 6 :733–734, 1977.

[6] T.G. Ceccherini-Silberstein, A. Machi, and F. Scarabotti. Amenable groups and cellular automata. Ann. Institut Fourier, 49(2) :673–685, 1999.

[7] C. Charnes, B. Martin, and P. Solé. A lattice-based McEliece scheme for encryption and signature. Electronic Notes in Discrete Mathematics, 6, 2001.

[8] T. Cheatham. Models, languages and compiler technology for high performance computers. In MFCS'94, LNCS, pages 3–26. Springer Verlag, 1994.

[9] T. Cheatham, A. Fahmy, D. Stefanescu, and L. Valiant. Bulk synchronous parallel computing — a paradigm for transportable software. In Proceedings of the 28th Annual Hawaii Conference on System Sciences, volume II. IEEE Computer Society Press, January 1995.

[10] P.M. Cohn. Algebra. John Wiley and Sons, 1989.

[11] H.M.S. Coxeter and W.O. Moser. Generators and relations for discrete groups. Springer Verlag, 1972.

[12] P. Dehornoy. Complexité et décidabilité. Mathématiques et Applications. Springer Verlag, 1993. ISBN 3-540-56899-9.

[13] M. Delorme. An introduction to cellular automata. Research Report 98-37, LIP ENS-Lyon, july 1998.

[14] Y. Feldman and E. Shapiro. Spatial machines : Towards a more realistic approach to parallel computation. Technical Report CS88-05, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel 76100, May 1988.

[15] Y. Feldman and E. Shapiro. Spatial machines, a more realistic approach to parallel computation. Communications of the ACM, 35(10) :61–73, 1992.

[16] M. Garzon. Models of massive parallelism : Analysis of cellular automata and neural networks. EATCS monographs. Springer Verlag, 1995.

[17] D. Gordon. On the computational power of totalistic cellular automata. Math. System Theory, 20 :43–52, 1987.

[18] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. Limits to parallel computation. Oxford University Press, 1995.

[19] O. Heen. Linear speed-up for cellular automata synchronizers and applications. Theoretical Computer Science, (188) :45–57, 1997.

[20] J.E. Hopcroft and J.D. Ullman. Introduction to automata theory, languages, and computation. Addison-Wesley, 1979.

[21] F. Hoza, B. Martin, and C. Monor. An approach of non-uniform automata arrays. In Hamza M.H., editor, IASTED International Conference on Modelling and Simulation. Acta Press, 2000.

[22] J. Gabarrô J.L. Balcâzar, J. Dîaz. Structural Complexity II. Spinger Verlag, 1990.

[23] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In Handbook of Theoretical Computer Science, volume A, chapter 17. Elsevier, 1990.

[24] M. Li and P. Vitányi. An introduction to Kolmogorov complexity and its applications. Texts and monographs in computer science. Springer Verlag, 1993.

[25] K. Lindgren and M. G. Nordhal. Universal computation in simple one-dimensional cellular automata. Complex Systems, 4 :299–318, 1990.

[26] A. Machí and F. Mignosi. Garden of eden configurations for cellular automata on cayley graphs of groups. SIAM Journal on Discrete Mathematics, 6(1) :44–56, 1993.

[27] M. Machtey and P. Young. An introduction to the general theory of algorithms. Theory of computation series, North Holland, 1978.

[28] M. Margenstern. Étude de la frontière entre décidabilité et indécidabilité du problème de l'arrêt pour les machines de turing, petites et/ou contraintes. Thèse d'habilitation LITP TH 94.01, Laboratoire d'Informatique Théorique et Programmation, 1994.

[29] B. Martin. Efficient unidimensional universal cellular automaton. In Proc. MFCS'92, number 629 in LNCS, pages 374–382. Springer Verlag, 1992.

[30] B. Martin. Construction modulaire d'automates cellulaires. Thèse de doctorat, Ecole Normale Supérieure de Lyon et Université Claude Bernard, LYON I, février 1993.

[31] B. Martin. A uniform universal CREW PRAM. In Proceedings of the Mathematical Foundations of Computer Science, number 711 in Lecture Notes in Computer Science, pages 557–565. Springer Verlag, august 1993.

[32] B. Martin. Machines spatiales universelles. In L. Bougé, M. Cosnard, and P. Fraigniaud, editors, Actes des 6èmes Rencontres Francophones du Parallélisme, pages 55–58. Ecole Normale Supérieure de Lyon, june 1994.

[33] B. Martin. A universal cellular automaton in quasi-linear time and its S-m-n form. Theoretical Computer Science, 123 :199–237, 1994.

[34] B. Martin. Machines spatiales universelles. Technique et Science Informatiques, 14(5) :551–566, 1995.

[35] B. Martin. Cellular automata universality revisited. In FCT'97, number 1279 in LNCS, pages 329–339. Springer Verlag, 1997.

[36] B. Martin. Embedding torus automata into a ring of automata. Int. Journal of Found. of Comput. Sc., 8(4) :425–431, 1997.

[37] B. Martin. BSP simulation of cellular automata. In N. Callaos, T. Yang, and J. Aguilar, editors, SCI'98 / ISAS'98, volume 2, pages pp 701–705. IIIS Press, 1998.

[38] B. Martin. A parallel simulation of cellular automata by spatial machines. In EuroPar'99, number 1685 in LNCS, pages 557–560. Springer Verlag, 1999.

[39] B. Martin. A simulation of cellular automata on hexagons by cellular automata on rings. Theoretical Computer Science, 265 :231–234, 2001.

[40] B. Martin. Codage, cryptologie et applications. Collection technique et scientifique des télécommunications. Presses Polytechniques et Universitaires Romandes, 2004.

[41] B. Martin and C. Peyrat. A single copy minimal-time simulation of a torus of automata by a ring of automata, soumis à. Discrete Applied Mathematics, 2004.

[42] J. Mazoyer. A six-states minimal time solution to the firing squad synchronization problem. Theoretical Computer Science, 50 :183–238, 1987.

[43] J. Mazoyer. Entrées et sorties sur une ligne d'automates. In Algorithmique parallèle. Masson, 1992.

[44] J. Mazoyer and N. Reimen. A linear speed-up theorem for cellular automata. Theoretical Computer Science, 101 :59–98, 1992.

[45] K. Melhorn and U. Vishkin. Randomized and deterministic simulations of prams by parallel machines with restricted granularity of parallel memories. Acta Informatica, 21 :339–374, 1984.

[46] M. Minsky. Finite and infinite machines. Prentice Hall, 1967.

[47] K. Noguchi. Simple 8-state minimal time solution to the firing squad synchronization problem. Theoretical Computer Science, 314 :303–334, 2004.

[48] C.H. Papadimitriou. Computational complexity. Addison Wesley, 1994.

[49] C. Papazian. Graphes d'automates finis, reconnaissance et calcul. PhD thesis, Ecole Normale Supérieure de Lyon, 2002.

[50] H. Rogers. Theory of recursive functions and effective computability. Mc Graw-Hill, 1967.

[51] Zs. Róka. One–way cellular automata on Cayley graphs. In Proc. FCT'93, number 710 in LNCS, pages 406–417. Springer Verlag, 1993.

[52] Zs. Róka. Automates cellulaires sur graphes de Cayley. PhD thesis, École Normale Supérieure de Lyon, 1994.

[53] Zs. Róka. Simulations between cellular automata on cayley graphs. Research Report 94-40, LIP-ENS-Lyon, 1994.

[54] Zs. Róka. The Firing Squad Synchronization Problem on Cayley graphs. In Proc. MFCS'95, number 969 in LNCS, pages 402–411. Springer Verlag, 1995.

[55] A.R. Smith III. Simple computation-universal cellular spaces. Journal ACM, 18 :339–353, 1971.

[56] L.G. Valiant. A bridging model for parallel computation. Communications of the ACM, 33(8) :103–111, 1990.

[57] L.G. Valiant. General purpose parallel architecture. In Handbook of Theoretical Computer Science, volume A, chapter 18. Elsevier, 1990.

[58] A. Waksman. An optimum solution to the firing squad synchronization problem. Information and Control, 9 :66–78, 1966.

[59] S. Wolfram. Theory and applications of cellular automata. Advanced series on complex systems. World Scientific, Singapore, 1986.

Résumé

Ce mémoire est composé de deux grandes parties. Dans la première, nous simulons le fonctionnement d'automates cellulaires par différents modèles de calcul parallèle comme les PRAM, les XPRAM et les machines spatiales. Nous obtenons ainsi différentes preuves de l'universalité de ces modèles. Nous tirons quelques conséquences de ces résultats du point de vue de la calculabilité et de la complexité. Dans la seconde partie, nous considérons les automates cellulaires définis sur des graphes de Cayley finis. Nous rappelons la simulation de Róka qui permet de mimer le fonctionnement d'un tore hexagonal d'automates par un tore d'automates de dimension deux. Nous décrivons ensuite différentes manières de plonger un tore d'automates de dimension deux dans un anneau d'automates. Nous déduisons de ces résultats la simulation de tores de dimension finie par un anneau d'automates et celle d'un tore hexagonal d'automates par un anneau d'automates.

Abstract

This report consists in two main parts. The first deals with the simulation of cellular automata by different models of parallel computation : PRAM, XPRAM and spatial machines. We thus obtain new proofs of the universality of these models based on cellular automata. We draw some conclusions from these results from the point of view of computability and complexity. In the second part, we consider cellular automata defined on finite Cayley graphs. We point out the simulation of Róka which allows to mimic the operation of a hexagonal torus of automata by a torus of automata of dimension two. We describe then various manners of embedding a torus of automata of dimension two in a ring of automata. We deduce from these results the simulation of torus of finite dimension by a ring of automata and that of a hexagonal torus of automata by a ring of automata.