Reconnaissance de langage en temps réel sur automates cellulaires 2D

Anaël Grandjean

Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de Docteur

Délivré par l'**Université de Montpellier**

Préparée au sein de l'école doctorale **I2S**
Et de l'unité de recherche **LIRMM**

Spécialité: **Informatique**

Présentée par **Anaël Grandjean**

Reconnaissance de langages en temps réel sur automates cellulaires 2D

Soutenance prévue le 6 décembre 2016 devant le jury composé de :

Directeurs de thèse		
Bruno Durand	Professeur	Univ. de Montpellier
Victor Poupet	Maître de conférence	Univ. de Montpellier
Rapporteurs		
Nicolas Ollinger	Professeur	Univ. d'Orléans
Véronique Terrier	Maître de conférence	Univ. de Caen
Président du jury		
Miklos Molnar	Professeur	Univ. de Montpellier
Examinateurs		
Jarkko Kari	Professor	Univ. of Turku
Gaétan Richard	Maître de conférence	Univ. de Caen
Guillaume Theyssier	Chargé de recherche	CNRS

Université Montpellier

Remerciements

La rédaction de ce mémoire fut sans aucun doute le travail le plus long et le plus intense de mes trois années de thèse. C'est pourquoi je tiens à remercier avec la plus grande attention toutes les personnes ayant contribué, directement ou non, à cette réalisation.

Mes premiers remerciements vont à mes directeurs de thèse, Bruno Durand et Victor Poupet pour leur aide. Il m'aurait été très difficile de terminer cette thèse sans leur oreille attentive et leurs retours toujours constructifs. Je remercie particulièrement Victor qui m'a guidé scientifiquement pendant ces trois années et qui m'a permis de m'intégrer dans le monde des enseignants-chercheurs.

Je remercie sincèrement Nicolas Ollinger, Gaétan Richard et Véronique Terrier qui ont accepté d'être rapporteurs de ma thèse et qui ont pris le temps de relire avec précision mon mémoire. Leurs remarques et commentaires m'ont permis de voir mon travail sous un nouveau jour et de rendre mon mémoire, je l'espère, un peu moins indigeste.

Je tiens à remercier chaleureusement Miklos Molnar et Guillaume Theyssier d'avoir accepté de participer à mon jury de thèse et surtout Jarkko Kari d'avoir fait le déplacement depuis Turku pour venir assister à ma soutenance. Je tiens aussi à remercier tous les membres et ex-membres de l'équipe ESCAPE qui m'ont accueilli et qui m'ont offert un cadre de travail des plus agréables. Je remercie notamment mes co-bureaux Sabrina, Guilhem, Julien et Swan qui m'ont supporté tous les jours depuis plusieurs années et que j'ai probablement empêché de travailler plus d'une fois. Je remercie aussi bien entendu tous les autres amis que j'ai pu trouvé au LIRMM, Florian, François, Julien, Jessie, Valentin, Clément, Guillerme et Namrata pour toutes les discussions scientifiques et philosophiques que nous avons pu avoir aux tournesols, devant un tableau ou autour d'un jeu de rôle.

Il m'est impossible d'oublier Nicolas et Laurie pour leur aide précieuse et sans qui je me serais probablement perdu sous une montagne de documents administratifs avant même de commencer ma thèse.

Je remercie encore une fois Gaétan Richard et Véronique Terrier qui ont su me donner le goût de la recherche dès mon premier stage. C'est sans aucun doute cette expérience qui m'a donné l'envie d'écrire cette thèse sur les automates cellulaires.

Je n'en serais pas là non plus sans tous mes camarades de l'ENS Lyon qui m'ont permis de relâcher la pression lorsque j'en avais besoin. Je ne compte plus les innombrables heures passées à jouer, à manger, à boire, à papoter, à faire du ski ou à

visiter des châteaux. Il m'est impossible de faire une liste complète de toutes les raisons que j'ai de les remercier. En espérant n'oublier personne.. merci à Audrey, Élie, Étienne, Fred, Lambin, Thomas, Mathias, Rémi, Robin, Samantha, Sarah, Thomas, Vincent et enfin à Alvaro qui me subit depuis plus longtemps que les autres.

Mes derniers remerciements vont à ma famille qui m'a toujours soutenu et particulièrement à Camille pour m'avoir aidé, relu et surtout supporté, non seulement au cours de ma thèse mais depuis maintenant de nombreuses années. Je n'aurais jamais pu arriver aussi loin sans son soutien et ses encouragements aux divers moments où la motivation me faisait défaut.

Table des matières

Chapitre 1

Introduction

Les automates cellulaires sont le fruit d'une collaboration entre J. von Neumann et S. Ulam dans le milieu des années 50 [vN66]. Initialement conçus dans le but d'étudier les mécanismes d'auto-reproduction, ils sont constitués d'une infinité d'éléments simples, appelés *cellules*, disposés régulièrement dans un espace qui peut être à une ou plusieurs dimensions. Les cellules ne disposent que d'une mémoire finie représentée par un *état* et évoluent de manière synchrone en fonction des états de leurs voisines selon une règle commune.

La simplicité mathématique des automates cellulaires, pourtant capables de produire des comportements complexes et difficiles à prévoir, en fait rapidement un modèle privilégié pour l'étude de nombreux phénomènes dans des disciplines aussi diverses que la physique [Den88, Tof84], la biologie [EE93], l'urbanisme [RNSL96] ou l'épidémiologie [Whi07].

C'est à la fin des années 60 que ce modèle commence à être plus formellement étudié grâce notamment à deux résultats majeurs. Le premier est dû à G. Hedlund [Hed69] qui donne une caractérisation purement topologique des automates cellulaires en tant que fonctions agissant sur l'espace des configurations, ouvrant ainsi la voie à une analyse mathématique des automates cellulaires en tant que systèmes dynamiques discrets. Le second résultat est dû à A. R. Smith III qui montre que les automates cellulaires en une dimension (les cellules étant disposées sur une ligne) sont capables de simuler le fonctionnement d'une machine de Turing [Smi71]. Ce résultat permet de démarrer une étude de la puissance de calcul des automates cellulaires, en les comparant notamment aux modèles existants (machines de Turing, machines RAM, circuits, etc.) dans le but de comprendre le gain apporté par le parallélisme massif.

C'est dans ce deuxième axe de recherche, qui s'intéresse aux automates cellulaires en tant que modèles de calcul parallèle, que s'inscrivent les travaux de cette thèse.

Résultats antérieurs

La plupart des travaux dans ce domaine s'intéressent aux automates cellulaires unidimensionnels d'une part parce que l'espace des cellules peut ainsi être immédiatement assimilé aux rubans des machines de Turing ce qui permet de comparer les deux modèles sur des entrées et sorties identiques et d'autre part parce que la théorie de la calculabilité passe en général par des codages à l'aide de mots finis unidimensionnels.

Les travaux de A. R. Smith III montrent que tout calcul effectué par une machine de Turing à un ou plusieurs rubans peut être réalisé par un automate cellulaire unidimensionnel en un temps identique et en utilisant le même espace de calcul. À l'inverse, on peut remarquer qu'un calcul effectué par un automate cellulaire en temps f et espace g peut être réalisé par une machine de Turing à un ruban en temps fg et espace g, par « balayages » de l'espace de calcul. Bien que ces remarques laissent penser qu'il existe de nombreux problèmes sur lesquels le parallélisme des automates cellulaires apporte un gain de temps important (d'un facteur égal à la taille de l'espace utilisé), il est très difficile d'obtenir de tels résultats qui reviennent à prouver des bornes inférieures de complexité. Une exception notable concerne les classes de langages reconnus en temps réel, c'est-à-dire le temps le plus court tel que le résultat du calcul ait pu dépendre de toutes les lettres du mot en entrée. F. Hennie a en effet montré que dans le cas des machines de Turing à un ruban les langages reconnus en temps réel sont les mêmes que ceux reconnus en temps linéaire et qu'ils sont rationnels [Hen65], tandis que A. Rosenberg a montré l'existence d'un langage reconnu en temps linéaire par une machine de Turing à plusieurs rubans qui n'était pas reconnu en temps réel sur ce même modèle [Ros67]. Ces deux résultats permettent de déterminer que les automates cellulaires reconnaissent strictement plus de langages que les machines de Turing en temps réel.

En dehors des comparaisons avec les autres modèles de calcul, plusieurs travaux s'intéressent aux relations entre les différentes classes de complexité purement cellulaires. Ici encore les classes correspondant au temps réel et au temps linéaire sont particulièrement étudiées, souvent dans le but d'avancer sur la résolution d'une question ouverte depuis 1972 consistant à déterminer si ces deux classes sont égales pour les automates cellulaires unidimensionnels [Smi72]. Un des résultats les plus significatifs dans cette direction, dû à O. Ibarra et T. Jiang, montre que ces classes sont égales si et seulement si la classe des langages reconnus en temps réel est close par retournement de l'entrée (miroir) [IJ88].

Toujours en une dimension, deux voisinages ont été particulièrement considérés : le voisinage bidirectionnel « usuel » permettant aux cellules de l'automate de tenir compte des états de leurs voisines dans chacune des deux directions lors des transitions, et le voisinage unidirectionnel selon lequel les cellules n'ont accès qu'à l'état de leur voisine dans une seule direction. O. Ibarra et T. Jiang ont par exemple montré que tout langage reconnu en temps linéaire par un automate cellulaire bidirectionnel était également reconnu par un automate cellulaire unidirectionnel [IJ88]

tandis que C. Choffrut et K. Culik ont montré que tout langage reconnu en temps réel par un automate cellulaire bidimensionnel était reconnu en temps linéaire par un automate unidirectionnel mais que les automates unidirectionnels étaient strictement plus faibles que les automates bidirectionnels en temps réel [CC84]. V. Poupet a toutefois montré que la diversité des voisinages en une dimension se réduisait à ces seules deux classes [Pou05], tout voisinage étant équivalent au voisinage usuel s'il s'étend dans les deux directions et au voisinage unidirectionnel sinon.

Enfin, pour compléter l'étude des classes de complexité sur les automates cellulaires unidimensionnels, quelques techniques de nature algorithmique ont été développées. En particulier, comme pour les machines de Turing, des théorèmes d'accélération linéaire et constante ont été établis en reprenant les idées des constructions utilisées sur les autres modèles de calcul [MR92], mais également des techniques purement parallèles n'ayant pas d'équivalent sur les modèles séquentiels comme la synchronisation de cellules (ligne de fusiliers) [Maz86, Maz96].

En parallèle de ces travaux en une dimension, certaines études ont également été menées sur les automates cellulaires en deux dimensions ou plus. Si les automates cellulaires en deux dimensions pouvaient sembler irréalisables en pratique il y a quelques décennies, la miniaturisation et la baisse des coûts de production des processeurs ont permis le développement rapide du calcul parallèle. Bien qu'imaginés un demi-siècle plus tôt, les automates cellulaires bidimensionnels sont aujourd'hui un excellent modèle théorique pour l'analyse des possibilités d'objets tels que les *Field-programmable gate array* (FPGA) [PG99, KMH01] ou les applications de calcul générique sur processeurs graphiques (GPGPU) [BC07, FNI17, PA08] qui disposent de milliers d'éléments de calcul disposés régulièrement sur une grille plane.

En deux dimensions la situation est toutefois sensiblement plus complexe. Au cours d'un calcul les informations peuvent se déplacer dans toutes les directions et il peut être difficile d'assurer que deux informations se rencontrent. Le choix du voisinage utilisé, qui définit directement le graphe de communication des cellules, influe donc fortement sur les possibilités algorithmiques de l'automate en ce qui concerne les classes de faible complexité où les déplacements d'information doivent être faits de manière optimale.

La plupart des travaux sur les automates cellulaires en deux dimensions concernent les automates fonctionnant sur les voisinages de von Neumann (quatre plus proches voisines) ou de Moore (huit plus proches voisines). Ces voisinages représentent tous les deux des extensions naturelles en deux dimensions du voisinage usuel unidimensionnel. Une grande partie des constructions employées en une dimension se généralise donc assez facilement aux automates fonctionnant sur ces voisinages, comme par exemple l'accélération constante et l'accélération linéaire [Ter04]. Il faut noter toutefois que l'accélération constante est obtenue de manière très différente sur ces deux voisinages, chaque technique pouvant être vue comme une extension de la technique utilisée en une dimension.

Il existe cependant aussi des résultats en deux dimensions qui dépassent ceux disponibles en une dimension. En effet V. Terrier est parvenue à montrer que les

classes du temps linéaire et du temps réel sont différentes lorsqu'on considère les automates fonctionnant avec le voisinage de Moore [Ter99]. Elle a ensuite étendu ce résultat à une classe plus large de voisinages, mais sans toutefois réussir à inclure le voisinage de von Neumann [Ter04]. L'étude des autres voisinages reste très incomplète mais on peut néanmoins noter des résultats encourageants comme un théorème de M. Delacourt et V. Poupet [DP07] qui apporte une première notion d'équivalence entre certains voisinages en deux dimensions. Les théorèmes d'accélération constante et linéaire ont aussi été étendus à des familles de voisinages sans toutefois parvenir à une généralisation complète [Ter04].

Organisation du mémoire

Mes travaux s'inscrivent directement à la suite de ces résultats et sont organisés en six chapitres. Le chapitre 2 pose les définitions formelles liées aux automates cellulaires et à la reconnaissance de langages. Le chapitre 3 présente des constructions classiques ainsi que deux résultats inspirés de constructions existantes mais néanmoins propres à cette thèse. Ces constructions sont essentielles pour permettre au lecteur de se familiariser avec l'algorithmique des automates cellulaires et pour simplifier de nombreuses démonstrations dans le reste du mémoire.

Le quatrième chapitre concerne les théorèmes d'accélération. La première partie comporte la preuve du théorème d'accélération constante pour une classe restreinte de voisinages qui ressemblent à celui de von Neumann. La deuxième partie est dédiée à la preuve du théorème d'accélération linéaire en deux dimension. Ce théorème est valable pour tous les voisinages et sa preuve nécessite l'emploi de nombreuses techniques propres aux automates cellulaires. Comme pour les voisinages usuels, le but est de compresser l'entrée de l'automate pour pouvoir ensuite calculer plus rapidement. Avec un voisinage quelconque il est très difficile de procéder à la compression en temps optimal. C'est une méthode nouvelle d'approximation d'un voisinage qui permet d'effectuer cette compression en perdant suffisamment peu de temps pour aboutir au résultat escompté.

La suite du mémoire présente mes travaux sur les liens entre les différentes classes de complexité usuelles en fonction des diverses variables du modèle, à savoir le voisinage et la dimension. Nous présentons tout d'abord une version généralisée d'un résultat de V. Terrier séparant les classes du temps réel et du temps linéaire pour les automates bidimensionnels avec certains voisinages. Nous nous intéressons ensuite à l'apport en terme de puissance de calcul, d'une dimension supplémentaire. Notamment nous essayons de comparer la puissance des automates de dimension d et $(d+1)$ en terme de reconnaissance de langages de dimension d. Nous arrivons d'ailleurs à conclure que cette différence de dimension apporte une différence de puissance pour les dimensions supérieures à deux.

Je me suis ensuite concentré sur le temps réel en deux dimensions. Les résultats suivants portent donc sur les différences apportées par le choix du voisinage sur cette classe particulière. J'exhibe dans un premier temps une classe de voisinages qui

définissent chacun des classes de temps réel incomparables deux à deux. Je présente ensuite une démonstration qu'une généralisation du voisinage de von Neumann était possible, en gardant exactement la même puissance de calcul. C'est enfin grâce à ce résultat que nous avons pu démontrer la supériorité stricte de ce voisinage par rapport à une classe assez vaste de voisinages.

Enfin, quelques discussions sont proposées à propos de la légitimité de l'intérêt porté au temps réel. En effet, il est connu en dimension un que les langages reconnaissables en moins de temps que le temps réel sont triviaux. Ce n'est cependant pas le cas en dimension deux. Nous survolerons alors rapidement quelques propriétés de ces classes très restreintes.

Chapitre 2

Définitions

2.1 Automate cellulaire

Un automate cellulaire est composé d'une infinité de cellules identiques, disposées sur un graphe régulier qui dans ce manuscrit sera toujours la grille $\mathbb{Z}^d$. Chacune de ces cellules ne prendre qu'un nombre fini d'états, c'est-à-dire qu'elle ne dispose que d'une mémoire finie. De plus, chacune de ces cellules évolue en même temps de manière discrète. À chaque étape du calcul, chaque cellule passe dans un nouvel état déterminé par une unique fonction de transition qui s'applique à toutes les cellules. Cette fonction est locale, c'est-à-dire que le nouvel état d'une cellule donnée ne dépend que des états d'un nombre fini de cellules voisines.

Définition 2.1. *Un* automate cellulaire *est un quadruplet* $\mathcal{A} = (d, \mathcal{Q}, \mathcal{V}, \delta)$ *où*
- d *est la dimension de l'automate ;*
- $\mathcal{Q}$ *est un ensemble fini d'états que les cellules de l'automate peuvent prendre ;*
- $\mathcal{V}$ *est un sous-ensemble fini de* $\mathbb{Z}^d$ *qu'on appelle voisinage ;*
- $\delta : \mathcal{Q}^{\mathcal{V}} \to \mathcal{Q}$ *est la fonction de transition locale de l'automate.*

On appelle configuration d'un automate $\mathcal{A}$ toute application $\mathfrak{C}$ de $\mathbb{Z}^d$ dans $\mathcal{Q}$. Cette application associe à chaque cellule $c \in \mathbb{Z}^d$ l'état $\mathfrak{C}(c)$ dans lequel elle se trouve. Lors d'une étape de calcul de l'automate $\mathcal{A}$ cette cellule va passer dans un nouvel état en fonction des états des cellules dans son voisinage, noté $\mathcal{V}(c)$ et définit comme suit :

$$\mathcal{V}(c) = c + \mathcal{V} = \{c + v | v \in \mathcal{V}\}$$

Le voisinage de la cellule c constitue donc l'ensemble des cellules qu'elle peut « voir », ce sont les cellules qui peuvent influer sur le prochain état de c. Concrètement la cellule voit une restriction de la configuration $\mathfrak{C}$ qu'on note comme suit :

$$\mathcal{V}_{\mathfrak{C}}(c) : \left\{ \begin{array}{ccc} \mathcal{V} & \to & \mathcal{Q} \\ v & \mapsto & \mathfrak{C}(c+v) \end{array} \right.$$

Cette cellule va donc passer dans l'état $\delta(\mathcal{V}_{\mathfrak{C}}(c))$ lors de la prochaine étape de temps. Ainsi on peut définir une fonction de transition globale de l'automate :

$$\Delta(\mathfrak{C}) : \left\{ \begin{array}{rcl} \mathbb{Z}^d & \rightarrow & \mathcal{Q} \\ c & \mapsto & \delta(\mathcal{V}_{\mathfrak{C}}(c)) \end{array} \right.$$

Dans la suite du manuscrit, nous associerons souvent l'automate $\mathcal{A}$ et sa fonction de transition globale Δ et, par abus de notation, nous noterons $\mathcal{A}$ cette fonction de transition. On confondra aussi étape de calcul et étape de temps. Ainsi, la configuration $\mathcal{A}(\mathfrak{C})$ sera appelée successeur de la configuration $\mathfrak{C}$ et, si cette dernière était la configuration initiale de l'automate, on dira que $\mathcal{A}$ était dans la configuration $\mathfrak{C}$ au temps 0, $\mathcal{A}(\mathfrak{C})$ au temps 1 et $\mathcal{A}^i(\mathfrak{C})$ au temps i.

2.2 Voisinages

En dimension d et étant donné un voisinage $\mathcal{V}$, on utilisera les notations suivantes :

$$\begin{array}{rrcccl} \forall c \in \mathbb{Z}^d, & \mathcal{V}^0(c) & = & \{c\} & & \\ \forall c \in \mathbb{Z}^d, & \mathcal{V}^1(c) & = & \mathcal{V}(c) & = & \{c+v | v \in \mathcal{V}\} \\ \forall E \subseteq \mathbb{Z}^d & \mathcal{V}(E) & = & \bigcup_{e \in E} \mathcal{V}(e) & = & \{e+v | e \in E, v \in \mathcal{V}\} \\ \forall c \in \mathbb{Z}^d, k \in \mathbb{N}, & \mathcal{V}^{k+1}(c) & = & \mathcal{V}(\mathcal{V}^k(c)) & & \\ \forall k \in \mathbb{N}, & \mathcal{V}^k & = & \mathcal{V}^k(0) & & \\ \forall k \in \mathbb{Z}, & k.\mathcal{V} & = & \{kv | v \in \mathcal{V}\} & & \\ & \mathcal{V}^\infty & = & \bigcup_{k \in \mathbb{N}} \mathcal{V}^k & & \end{array}$$

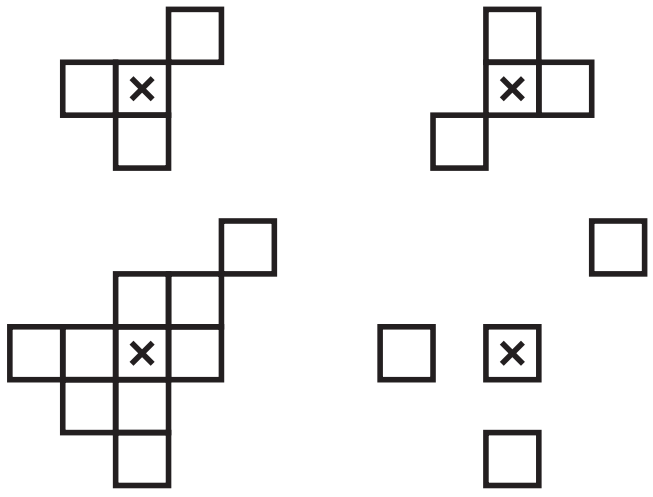

FIGURE 2.1 – Le voisinage triangle $\mathcal{V}_T$ (à gauche), $-\mathcal{V}_T$ (à droite), ${\mathcal{V}_T}^2$ (en bas à gauche) et $2\mathcal{V}_T$.

L'utilisation de la notation $\mathcal{V}^k$ peut paraître abusive puisqu'elle correspond à ce qui est usuellement considéré comme la somme d'ensembles ($\mathcal{V}^{k+1} = \mathcal{V} + \mathcal{V}^k$).

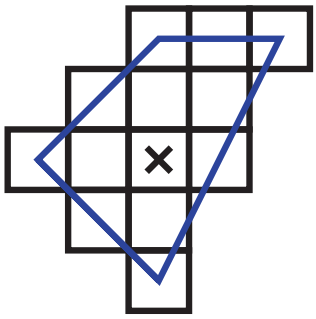

FIGURE 2.2 – Un voisinage en deux dimensions et l'enveloppe convexe associée. Le point $(0, 0)$ est représenté par une croix, il est appelé origine et sera noté 0.

Ce choix est fait pour éviter toute ambiguïté avec la notation $k.\mathcal{V}$ qui correspond exactement à une multiplication par un entier, mais pas à une somme de k termes. La distinction entre ces deux notions est illustrée sur un exemple dans la figure 2.1. De plus, le voisinage étant utilisé comme une fonction de $\mathbb{Z}^d$ dans les parties de $\mathbb{Z}^d$ il n'est pas choquant d'utiliser ce genre de notation pour designer les applications successives de $\mathcal{V}$.

Définition 2.2 (Proportion d'un point du voisinage)**.** *On appelle* proportion *d'un vecteur* (a, b) *de* $\mathbb{Z}^2$ *le rapport* $\frac{a}{b}$.

2.2.1 Enveloppes convexes

Lors de l'étude de l'impact du choix du voisinage sur la puissance de calcul associée aux automates cellulaire, il est intéressant, et souvent plus pratique aussi, de s'intéresser à la forme générale d'un voisinage plus qu'à sa composition exacte. Pour cela nous allons introduire la notion d'enveloppe convexe :

Définition 2.3. *L'*enveloppe convexe *d'un voisinage* $\mathcal{V} = \{v_1, ..., v_n\}$*, notée* $\mathrm{CH}(\mathcal{V})$ *(pour Convex Hull), est définie par :*

$$\mathrm{CH}(\mathcal{V}) = \left\{ \sum_{i=1}^{n} \lambda_i v_i \quad \middle| \quad \forall i, \lambda_i \in [0, 1] \cap \mathbb{Q} \quad et \quad \sum_{i=1}^{n} \lambda_i \leq 1 \right\}.$$

Cet ensemble correspond à toutes les combinaisons linéaires rationnelles des sommets de $\mathcal{V}$. De ce fait il correspond à l'ensemble des vecteurs de déplacements possibles des informations au sein de l'automate. Bien que cette notion ne soit pas encore très claire, elle sera expliquée plus en détail prochainement et est au centre de nombreux algorithmes proposés dans ce manuscrit. En particulier, on peut remarquer que cette enveloppe constitue un polyèdre dont les sommets ont tous des coordonnées entières. De plus, ces sommets sont des éléments du voisinages et par abus de notation nous les appellerons les sommets du voisinage. Il est important de noter que les autres points de l'enveloppe convexe, même s'ils ont des coordonnées

entières, ne sont pas nécessairement des éléments du voisinage. Les voisinages qui contiennent tous ces points seront appelés convexes :

Définition 2.4. *Un voisinage $\mathcal{V}$ est dit* convexe *si $\mathcal{V} = \mathrm{CH}(\mathcal{V}) \cap \mathbb{Z}^d$.*

2.2.2 Complétude et symétries

Le voisinage correspondant aux moyens de communication dans l'automate, une propriété importante est la possibilité pour toutes les cellules de communiquer entre elles. En effet, si l'état d'une cellule c à l'étape suivante dépend des états actuels des cellules de $\mathcal{V}(c)$, une étape plus tard il dépendra des états des cellules de $\mathcal{V}^2(c)$ et ainsi de suite. Il est donc possible de définir une pseudo distance entre deux cellule en fonction du voisinage. Il est important de remarquer qu'avec un voisinage qui n'est pas symétrique, la « distance » entre deux cellules n'est pas forcément la même dans les deux sens. On peut aussi se demander s'il existe des cellules qui ne sont pas à une distance finie l'une de l'autre, c'est-à-dire des cellules qui ne peuvent pas communiquer entre elles. Lorsque l'on s'intéresse à l'algorithmique réalisable avec des automates cellulaires, il est plus pratique de considérer des voisinages qui permettent à chaque cellule d'agir sur toutes les autres. Ces voisinages sont appelés complets :

Définition 2.5. *Un voisinage $\mathcal{V}$ est dit* complet *si $\mathcal{V}^\infty = \mathbb{Z}^d$.*

Parmi ces voisinages qui permettent à toutes les cellules de communiquer avec toutes les autres, on peut naturellement s'intéresser à ceux qui permettent une communication aussi efficace dans un sens que dans l'autre.

Définition 2.6. *Un voisinage $\mathcal{V}$ est dit* symétrique *si $-\mathcal{V} = \mathcal{V}$, c'est-à-dire s'il est symétrique par rapport à l'origine. De plus on dit que $\mathcal{V}$ est* fortement symétrique *si $\mathcal{V} = \{((-1)^{p_1}x_1, (-1)^{p_2}x_2, \ldots, (-1)^{p_d}x_d) \mid (x_1, \ldots, x_d) \in \mathcal{V}, (p_1, \ldots, p_d) \in \{0,1\}^d\}$, avec d la dimension du voisinage. Cette définition signifie que la donnée des points de coordonnées positives suffit à définir tout le voisinage.*

Dans ce manuscrit nous considérerons qu'une cellule est toujours capable de se voir elle-même et donc que tous les voisinages contiennent le point $(0, 0)$. De plus la plupart des résultats que nous démontrerons concernent les voisinages complets. En deux dimensions, deux voisinages bénéficient d'une attention toute particulière et ont été longuement étudiés dans la littérature :

Définition 2.7. *Le voisinage de* von Neumann *est le voisinage $\mathcal{V}_{vN}$ des quatre plus proches voisins (en plus de la cellule elle même) :*

$$\mathcal{V}_{vN} = \{(0,0), (-1,0), (0,-1), (1,0), (0,1)\}$$

Le voisinage de Moore *est le voisinage $\mathcal{V}_M$ des huit plus proches voisins :*

$$\mathcal{V}_M = \{(0,0),(-1,0),(0,-1),(1,0),(0,1),(1,1),(-1,1),(1,-1),(-1,-1)\}$$

Ces deux voisinages sont illustrés sur la figure 2.3.

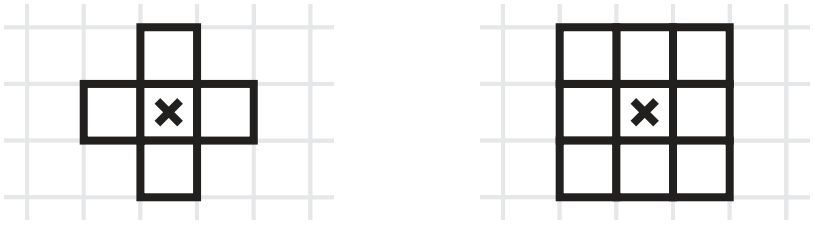

FIGURE 2.3 – Le voisinage de von Neumann à gauche et celui de Moore à droite

On peut remarquer d'ailleurs que ces deux voisinages sont complets, fortement symétriques et convexes.

2.3 Reconnaissance de langages

Afin d'étudier les automates cellulaire en tant que modèle de calcul, ce manuscrit traite principalement de la reconnaissance de langage avec ces automates. Puisque les automates cellulaires permettent de travailler en plusieurs dimensions, les langages considérés dans ce manuscrit seront eux aussi multidimensionnels. Le premier choix nécessaire pour définir la reconnaissance est celui de la manière dont on donne un mot en entrée à un automate cellulaire. Une solution communément utilisée est celle d'écrire le mot avec les états de l'automate cellulaire, dans une partie finie de l'automate, en complétant la configuration avec un état particulier qu'on note habituellement # et qu'on considère quiescent, c'est-à-dire qu'une cellule dans cet état ne changera pas d'état si toutes les cellules de son voisinage sont aussi dans l'état #.

Définition 2.8 (Ensemble des mots de dimension d)**.** *On note Σ^{*^d} l'ensemble des mots finis de dimension d sur l'alphabet Σ.*

Définition 2.9 (Proportion d'un mot)**.** *On appelle* proportion *d'un mot $w \in \Sigma^2$ de taille $(n \times m)$ le rapport $\frac{n}{m}$.*

Définition 2.10 (Écriture d'un mot)**.** *Étant donné un mot $w \in \Sigma^{*^d}$ de taille $(n_1, n_2, \ldots, n_d)$, on définit $\mathfrak{C}_w$ la configuration associée à w comme suit :*

$$\mathfrak{C}_w(x_1,x_2,\ldots,x_d) = \begin{cases} w(x_1,x_2,\ldots,x_d) & \text{si } x_1 \in [\![0,n_1-1]\!],\ldots,x_d \in [\![0,n_d-1]\!] \\ \# & \text{sinon.} \end{cases}$$

Comme pour les automates finis, le mot sera considéré comme accepté si l'automate entre dans un état d'acceptation. Cependant, le caractère massivement parallèle des automates cellulaires se prête mal à cette notion d'acceptation. Aussi il est généralement choisi de considérer une cellule particulière qui décidera de l'acceptation du mot. Avec ces considérations, il est possible de définir deux notions de reconnaissance :

Définition 2.11 (Reconnaissance forte)**.** *On dit qu'un automate cellulaire* $\mathcal{A} = (d, \mathcal{Q} \cup \{q_a, q_r\}, \mathcal{V}, \delta)$ *reconnaît le langage* $L \subset \Sigma^{*^d}$ *si*

- $\Sigma \cup \{\#\} \subset \mathcal{Q}$
- $w \in L \Rightarrow \exists t : \mathcal{A}^t(\mathfrak{C}_w)(0) = q_a$
- $w \notin L \Rightarrow \exists t : \mathcal{A}^t(\mathfrak{C}_w)(0) = q_r$
- q_a *est un état persistant, c'est-à-dire qu'une cellule qui rentre dans cet état n'en change plus jamais.*
- q_r *est un état persistant.*

Cette première notion de reconnaissance force l'automate à arrêter son calcul de lui-même. En effet, une fois qu'un état de refus ou d'acceptation a été atteint il ne peut plus être changé, le résultat est donc définitif. Cela permet de définir aisément le temps nécessaire à la reconnaissance d'un langage.

Définition 2.12 (Temps de reconnaissance)**.** *On dit qu'un langage d-dimensionnel* L *est reconnu en temps* $f : \mathbb{N}^d \to \mathbb{N}$ *par un automate* $\mathcal{A}$ *si pour tout mot* w *de taille* $(n_1, \ldots, n_d)$*, la cellule origine rentre dans un état d'acceptation ou de refus avant l'étape* $f(n_1, \ldots, n_d)$*.*

La deuxième définition possible de la reconnaissance est directement liée au temps de reconnaissance et s'exprime comme suit :

Définition 2.13 (Reconnaissance faible)**.** *On dit qu'un automate cellulaire* $\mathcal{A} = (d, \mathcal{Q} = \mathcal{Q}_r \cup \mathcal{Q}_a, \mathcal{V}, \delta)$*, avec* $\mathcal{Q}_a$ *et* $\mathcal{Q}_r$ *disjoints, reconnaît le langage* $L \subset \Sigma^{*^d}$ *en temps* f *si* $\Sigma \cup \{\#\} \subset \mathcal{Q}$ *et que pour tout mot* w *de taille* $(n_1, \ldots, n_d)$*,* $\mathcal{A}^{f(n_1,\ldots,n_d)}(\mathfrak{C}_w)(0) \in \mathcal{Q}_a \leftrightarrow w \in L$*.*

Cette notion de reconnaissance est moins contraignante, mais permet des abus via le choix de la fonction de temps. En effet, en utilisant des fonctions de temps non récursives on pourrait reconnaître des langages non récursifs. Dans ce manuscrit, cette définition ne sera utilisée que pour une fonction de temps particulière qui est le temps réel pour éviter tout problème de ce genre.

2.4 Temps réel et linéaire

Définition 2.14 (Temps réel)**.** *Le* temps réel *pour un voisinage* $\mathcal{V}$*, noté* $\mathrm{TR}_\mathcal{V}$*, est la fonction suivante :*

$$\mathrm{TR}_\mathcal{V} : \left\{ \begin{array}{rcl} \mathbb{N}^d & \to & \mathbb{N} \\ (n_1, \ldots, n_d) & \mapsto & \min(t \mid [\![0, n_1 - 1]\!] \times \cdots \times [\![0, n_d - 1]\!] \subset \mathcal{V}^t) \end{array} \right.$$

Il s'agit en fait du nombre minimal d'étapes nécessaire à un automate cellulaire avec le voisinage $\mathcal{V}$ pour que l'origine aie vu tout le mot d'entrée w. En effet, avant le temps réel, l'état de l'origine ne dépend que d'une portion du mot d'entrée. Par exemple les temps réels pour les voisinages en deux dimensions de von Neumann et de Moore, notés TR_{vN} et TR_M correspondent aux fonction suivantes :

$$\begin{aligned} \mathrm{TR}_{vN}(n,m) &= n+m-2 \\ \mathrm{TR}_M(n,m) &= \max(n,m)-1 \end{aligned}$$

On peut remarquer que le temps réel pour le voisinage de Moore est bien plus petit que le temps réel pour le voisinage de von Neumann, cela est dû au fait que le voisinage de Moore est plus grand. En d'autres termes le voisinage de Moore permet plus de communications entre les cellules et donc aux informations de se déplacer plus vite. Le choix du temps réel comme temps de reconnaissance permet de compenser cette différence en accordant un temps plus petit aux voisinages plus gros.

L'inconvénient de cette définition est que, pour reconnaître un langage en temps réel, il faut décider de l'acceptation au moment où la totalité de l'entrée est lue mais potentiellement sans avoir d'information sur les bords de l'entrée. C'est-à-dire que l'origine n'a aucun moyen de savoir si la totalité du mot est lue ou pas. C'est pour cela que nous utiliserons la notion de reconnaissance faible pour définir la classe des langages reconnus en temps réel.

Définition 2.15 (Classe du temps réel)**.** *On note* $\mathrm{CA}_{\mathcal{V}}(\mathrm{TR})$ *l'ensemble des langages qui peuvent être reconnus en temps réel (pour la définition faible de la reconnaissance) par un automate cellulaire dont le voisinage est le voisinage* $\mathcal{V}$*. On dira alors qu'un langage* $L \in \mathrm{CA}_{\mathcal{V}}(\mathrm{TR})$ *peut être reconnu en temps réel par un automate avec le voisinage* $\mathcal{V}$ *ou encore plus simplement qu'il peut être reconnu en temps réel avec le voisinage* $\mathcal{V}$*.*

Plus généralement, on définit une classe pour chaque fonction de temps mais cette fois-ci avec la définition forte de la reconnaissance. Le but étant de parler de fonctions de temps plus grandes que le temps réel où il devient raisonnable de demander à un automate de reconnaître la fin de son calcul.

Définition 2.16. *On note* $\mathrm{CA}_{\mathcal{V}}(f)$ *l'ensemble des langages qui peuvent être reconnus en temps* f *(pour la définition forte de la reconnaissance) par un automate cellulaire avec le voisinage* $\mathcal{V}$*.*

Plus particulièrement, la classe des langages reconnaissables en temps linéaire est étudiée dans cette thèse :

Définition 2.17 (Temps linéaire)**.** *On note* $\mathrm{CA}_{\mathcal{V}}(\mathrm{TL})$ *la classe des langages reconnus en temps linéaire par un automate avec le voisinage* $\mathcal{V}$ *:*

$$\mathrm{CA}_{\mathcal{V}}(\mathrm{TL}) = \bigcup_{k\in\mathbb{N}} \mathrm{CA}_{\mathcal{V}}(k\,\mathrm{TR}_{\mathcal{V}}).$$

Chapitre 3

Outils

Ce troisième chapitre présente des constructions classiques qui ont pour but principal de simplifier les constructions dans le reste du mémoire. Les quatre premières sections rappellent des résultats bien connus et largement utilisés dans la littérature. Les deux autres sections comportent des résultats qui sont propres à nos travaux, mais s'appuient sur des outils usuels. L'avant dernière section porte sur une construction unidimensionnelle inspirée d'une construction de O. Ibarra et T. Jiang [IJ88]. Enfin le dernier résultat de ce chapitre amène une méthode générale pour intégrer un précalcul sans perte de temps. Cette méthode a été notamment utilisée par W. Beyer [Bey69] puis par beaucoup d'autres, mais à chaque fois en l'adaptant au cas précis dans lequel elle était utilisée.

3.1 Outils de base

3.1.1 Groupement et simulation

Pour comparer les voisinages entre eux, il est souvent utile de simuler le comportement d'un automate avec un autre qui n'utilise pas le même voisinage. Il est tout d'abord important de préciser que, dans ce mémoire, le mot « simulation » aura un sens assez flou. En effet, il existe plusieurs moyens de définir formellement une simulation([Oll02, Rap98]). Ici simuler signifie qu'il est possible de retrouver le comportement de l'automate simulé au sein de l'automate simulant. Ceci peut être fait en considérant seulement certaines étapes de calcul ou en faisant une projection des états du simulant vers ceux du simulé par exemple. La simulation présentée ici est assez simple et se base sur un groupement préalable.

Étant donné un automate $\mathcal{A} = (d, \mathcal{Q}, \mathcal{V}, \delta)$, on définit un automate $\mathcal{A}' = (d, \mathcal{Q}', \mathcal{V}', \delta')$ qui en simule le comportement. Supposons que $\mathcal{V} \subset \mathcal{V}'^k$, c'est à dire que chaque cellule dans $\mathcal{V}$ est à « distance » au plus k avec le voisinage $\mathcal{V}'$. L'automate $\mathcal{A}'$ utilise alors l'ensemble d'états suivant : $\mathcal{Q}' = (\mathcal{Q} \cup \square)^{\mathcal{V}^k}$. On voit alors un état q de $\mathcal{A}'$ comme un ensemble de sous états (des états de $\mathcal{A}$), chacun pouvant être associé à un vecteur de $\mathcal{V}'^k$, on dira qu'il s'agit de la position du sous-état dans l'état q.

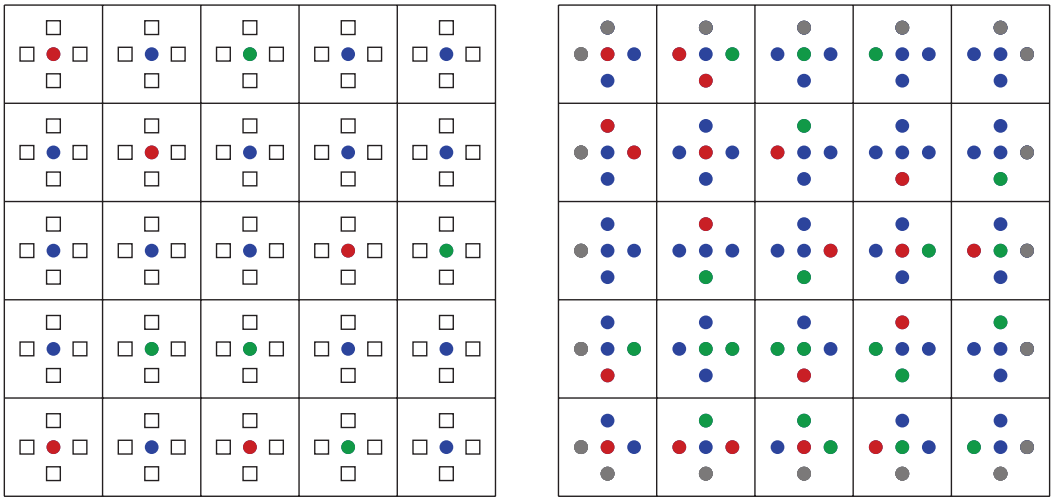

FIGURE 3.1 – Simulation du voisinage de Moore avec celui de von Neumann : à l'état initial tous les emplacements sont vides sauf les centraux. Au temps suivant, chaque emplacement vide a été rempli avec l'état du centre voisin. Avec ces informations il est possible de simuler une étape d'un automate avec le voisinage de Moore.

L'état $\square$ représente un emplacement vide dans une cellule de $\mathcal{A}'$. De plus, une cellule ne contient jamais d'état $\square$ en position 0. Étant donné un état q de l'automate $\mathcal{A}'$ on note alors $q(v)$ l'état de $\mathcal{A}$ en position v dans q. L'idée est que les cellules vont d'abord regrouper les informations de leurs voisines (créant de la redondance) pendant k étapes, afin de pouvoir ensuite calculer l'état suivant en position 0 avec la fonction de transition δ de l'automate $\mathcal{A}$ appliquée à ces sous états.

Soit $\mathfrak{C}$ une configuration dans laquelle aucune cellule n'a d'emplacement vide à des position dans $\mathcal{V}^n$. Chaque cellule va grâce à ses voisines remplir les emplacements qui sont à des positions de $\mathcal{V}^{n+1}$ comme suit :

$$\mathcal{A}'(\mathfrak{C})(c)(v + v_n) = \mathfrak{C}(c + v)(v_n).$$

Il pourrait exister plusieurs décompositions possibles $(v + v_n)$ avec $v \in \mathcal{V}$ et $v_n \in \mathcal{V}^n$ pour un emplacement donné. Cependant, si au départ toutes les cellules n'ont que l'emplacement 0 qui n'est pas vide, les diverses décompositions possibles correspondent à chaque fois aux mêmes sous états, la même information étant dupliquée dans de nombreux états.

Lorsque les cellules n'ont plus d'emplacement vide, la transition de l'automate $\mathcal{A}'$ se fait de la manière suivante :

$$\mathcal{A}'(\mathfrak{C})(c)(v) = \begin{cases} \square & \text{si } v \neq 0 \\ \delta(\mathfrak{C}(c)(0), \mathfrak{C}(c)(v_1), \ldots, \mathfrak{C}(c)(v_{|V|})) & \text{sinon} \end{cases}$$

Chaque cellule effectue donc l'équivalent d'une étape de calcul dans l'automate $\mathcal{A}$, ce qui est rendu possible grâce aux informations collectées pendant les k étapes précédentes. Ainsi, on considère que l'automate $\mathcal{A}'$ effectue une étape de l'automate

$\mathcal{A}$ en $k+1$ étapes, en identifiant l'état dont tous les emplacements sont vides sauf un et l'état correspondant dans l'automate $\mathcal{A}$. En fait, il est possible de faire la dernière étape de groupement et l'étape de simulation en une seule étape de calcul, on peut alors simuler une étape de $\mathcal{A}$ en k étapes de $\mathcal{A}'$. En particulier, si on rajoute qu'une cellule de $\mathcal{A}'$ dont l'état en position 0 est un état acceptant passe immédiatement dans un état acceptant et n'en sort plus, tout langage fortement reconnu par $\mathcal{A}$ l'est aussi par $\mathcal{A}'$. De plus, si ce langage était reconnu en temps f par $\mathcal{A}$, il l'est en temps kf par $\mathcal{A}'$. Cette construction nous permet alors de montrer le théorème suivant :

Théorème 3.1. *Étant donné deux voisinages complets $\mathcal{V}$ et $\mathcal{V}'$ de même dimension, on a* $\mathrm{CA}_{\mathcal{V}}(\mathrm{TL}) = \mathrm{CA}_{\mathcal{V}'}(\mathrm{TL})$.

En effet, le fait de considérer des voisinages complets assure l'existence de constantes k et k' telles que $\mathcal{V} \subset \mathcal{V}'^k$ et $\mathcal{V}' \subset \mathcal{V}^{k'}$. Puisque tous les voisinages que nous considérons sont complets, la classe du temps linéaire ne dépend plus que de la dimension.

Définition 3.2 (Temps linéaire)**.** *On note* $\mathrm{CA}_d(\mathrm{TL})$ *la classe des langages reconnaissables en temps linéaire en dimension d par n'importe quel voisinage complet.*

3.1.2 Signaux

Comme tous les calculs se font localement dans un automate cellulaire, il est naturel de vouloir faire transiter les informations utiles qui sont contenues dans une configuration donnée. Pour cela, on fait transiter l'information sous forme de signal. Une fois de plus cette notion est assez informelle et sert à décrire une grande variété de comportements [Fis65, Ric08]. Concrètement, un signal est une succession dans le temps de cellules qui prennent une à une un état particulier. On dit alors que cet état se déplace le long de ces cellules. Lorsque ces cellules sont régulièrement espacées on parlera de signal. L'espacement entre ces cellules sera appelé vecteur de déplacement du signal.

Les signaux les plus faciles à construire sont ceux qui se déplacent selon un vecteur du voisinage. En effet, si v est un vecteur du voisinage, la cellule $(c-v)$ peut prendre l'état de la cellule c dans la configuration précédente. Ainsi les informations se déplacent en fait selon les vecteurs opposés à ceux qui sont dans le voisinage.

Il est aussi possible d'envoyer des informations à des vitesses plus faibles que la vitesse maximale. En effet, dans certains cas on veut construire des signaux qui se déplacent de moins d'une case par tour. Bien que cela ne soit pas possible à cause du caractère discret des automates cellulaires, il est possible de se déplacer d'une case (ou selon un vecteur du voisinage quelconque v) seulement une étape de calcul sur deux. Pour cela on considère deux états q et q' qui représentent la même information mais se déplacent différemment. Un état q dans la cellule c engendrera le passage dans l'état q' de la cellule $(c-v)$ dans la configuration suivante de l'automate et un état q' engendra le passage de la cellule c elle-même dans l'état q.

Au cours de nombreuses constructions, il est important d'envoyer des signaux dans des directions qui ne correspondent à aucun vecteur du voisinage. Dans ce cas,

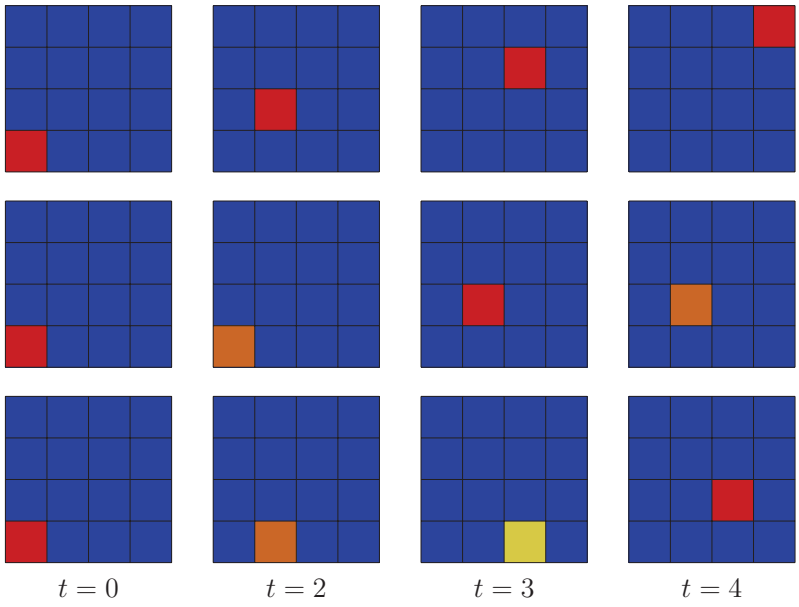

FIGURE 3.2 – Exemple de trois signaux. Le premier se déplace à vitesse maximale dans la direction $(1, 1)$ avec le voisinage de Moore. Le second se déplace dans cette même direction mais à vitesse $\frac{1}{2}$. Le troisième se déplace dans la direction $(2, 1)$ à vitesse maximale avec le voisinage de von Neumann.

il est impossible d'avoir un signal «parfait» mais il est possible d'en construire un dont la direction asymptotique est celle voulue. En fait, il est possible d'envoyer un signal dans n'importe quelle direction qui correspond à un vecteur rationnel de l'enveloppe convexe du voisinage. Considérons un vecteur v rationnel de $\mathrm{CH}(\mathcal{V})$, alors, par définition de l'enveloppe rationnelle, ce vecteur est une combinaison linéaire rationnelle de vecteurs de $\mathcal{V} : v = \Sigma\lambda_i v_i$. On peut donc trouver un multiple kv de v qui est une combinaison entière de ces vecteurs. On crée alors un signal qui se déplace $k\lambda_1$ fois selon le vecteur v_1, puis $k\lambda_2$ selon le vecteur v_2 et ainsi de suite. Ainsi toutes les k étapes le signal sera exactement au même endroit que s'il s'était réellement déplacé selon le vecteur v. Le reste du temps le signal n'est qu'à une distance bornée de l'endroit où il devrait se trouver, qui ne dépend pas de la distance parcourue, ce qui est important pour la suite de ce mémoire. Pour faire cela, le signal doit se «souvenir» du nombre d'étapes qui se sont écoulées depuis la dernière fois qu'il est passé sur le bon axe, ce qui nécessite k états différents.

3.1.3 Couches

Outre le parallélisme inhérent au modèle, certaines constructions nécessitent d'effectuer plusieurs tâches en parallèle et en un même endroit de l'espace, par exemple des envois de signaux ou des groupements. Lorsque ces tâches sont indépendantes sauf dans des cas très particuliers (par exemple la rencontre de deux signaux), il

est plus facile de définir un automate différent pour chacune de ces tâches, puis de déterminer ce qui se passe lors de ces cas particuliers. Pour effectuer plusieurs tâches en parallèle nous allons considérer tout simplement l'automate produit suivant :

Définition 3.3. *Étant donnés deux automates* $\mathcal{A} = (d, \mathcal{Q}, \mathcal{V}, \delta)$ *et* $\mathcal{A}' = (d, \mathcal{Q}', \mathcal{V}, \delta')$*, on appelle automate produit l'automate* $\mathcal{B} = (d, \mathcal{Q}_\mathcal{B}, \mathcal{V}, \delta_\mathcal{B})$ *où :*

— $\mathcal{Q}_\mathcal{B} = \mathcal{Q} \times \mathcal{Q}'$.

— $\delta_\mathcal{B}((q_1, q'_1), (q_2, q'_2), \ldots, (q_{|V|}, q'_{|V|})) = (\delta(q_1, q_2, \ldots, q_{|V|}), \delta'(q'_1, q'_2, q'_{|V|}))$.

On dit alors que l'automate $\mathcal{B}$ fonctionne avec deux couches en parallèle. Il est possible ensuite de faire interagir ces couches en redéfinissant la fonction de transition dans des cas particuliers. Cette méthode de construction est largement utilisée dans ce mémoire.

3.1.4 Espace de calcul

Au cours des constructions proposées dans ce mémoire, il est important de pouvoir faire certaines hypothèses quand à l'espace de calcul utilisé par les automates cellulaires considérés. Le but de cette section est de montrer que grâce à une construction astucieuse il est possible de restreindre fortement l'espace utilisé. Cette construction est assez classique et existe dans de nombreux modèles de calculs tels que les machines de Turing.

Proposition 3.4. *Étant donné un langage* L *reconnu par un automate* $\mathcal{A} = (d, \mathcal{Q}, \mathcal{V}, \delta)$*, si* $\mathcal{V}$ *est fortement symétrique et convexe, il est possible de reconnaître le langage* L *avec un automate* $\mathcal{A}'$ *dans lequel toutes les cellules avec une coordonnée négative restent dans l'état* $\#$*.*

Pour plus de clarté cette construction est présentée ici en deux dimensions mais peut être généralisée en toute dimension. L'idée de cette construction est de replier l'espace de calcul sur lui même selon l'axe des ordonnées puis selon celui des abscisses. Comme ces deux constructions sont similaires, nous n'allons expliciter que la première. La figure 3.3 illustre cette notion de repliage du plan sur lui même.

Un repliage selon l'axe des ordonnées consiste à replier le demi plan d'abscisse négative par dessus le demi plan d'abscisse positive. Chaque cellule d'abscisse positive va donc recevoir deux états au lieu d'un seul. L'ensemble d'état passe alors de $\mathcal{Q}$ à $\mathcal{Q} \times \mathcal{Q}$ créant ainsi deux couches. La première couche va se comporter exactement comme l'automate $\mathcal{A}$. La deuxième couche correspond à la couche repliée et est donc à l'envers, une cellule initialement d'abscisse x se retrouvant sur la même ligne mais avec l'abscisse $(-x-1)$. La colonne d'abscisse -1 se retrouve au dessus de la colonne d'abscisse 0 et ainsi de suite. Les états de cette couche vont donc évoluer selon une règle symétrique à δ, ce qui est possible aisément puisque le voisinage est lui même symétrique selon l'axe considéré. Le seule difficulté repose sur les cellules c dont le voisinage contient des cellules d'abscisse négative. Dans ce cas, il faut lors de l'application de la fonction de transition δ ne pas prendre en compte ces cellules

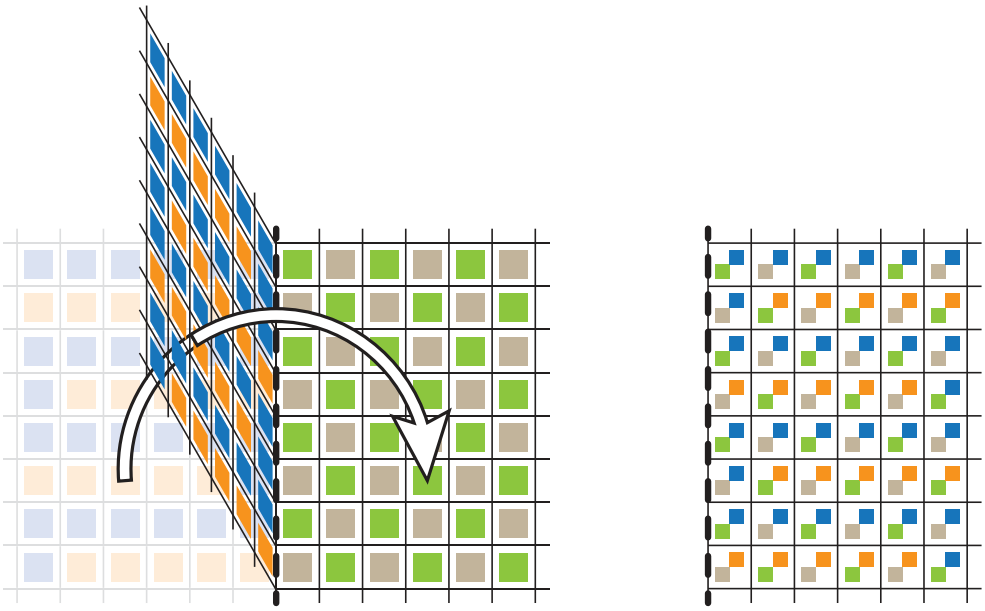

FIGURE 3.3 – Repliage du plan sur lui même, avec une pliure au niveau de l'axe des ordonnées.

(dont l'état restera toujours #) mais celles qui leur correspondent sur l'autre couche. En effet, la première couche joue aussi le rôle de couche retournée pour la deuxième. Ainsi aucune des deux couches n'a besoin d'informations situées dans une cellule d'abscisse négative. Il faut aussi vérifier que la cellule sur laquelle l'information a été repliée est bien elle aussi dans le voisinage de la cellule c. Ceci est toujours le cas grâce à la symétrie et à la convexité du voisinage. En effet, si une cellule $c = (x_c, y_c)$ a besoin de l'information d'une cellule $d = (x, y)$ dont l'abscisse y est négative, l'information se trouve en fait dans la cellule $d' = (x, -y - 1)$. Comme le voisinage est symétrique par rapport à l'axe des abscisses, la cellule $(x, y_c + (y_c - y))$ est dans le voisinage de la cellule c et donc par convexité la cellule $(x, -y - 1)$ l'est aussi. Grâce à cette construction, les états de la première couche des cellules d'abscisse positive sont exactement ceux des cellules correspondantes dans l'automate $\mathcal{A}$. Ainsi les états pris par l'origine dans $\mathcal{A}$ sont ceux de la première couche de l'origine dans $\mathcal{A}'$, c'est cette seule couche qui décidera de l'acceptation ou du refus du mot d'entrée. On a donc un automate $\mathcal{A}'$ qui reconnaît exactement le même langage que $\mathcal{A}$, en le même temps, mais sans utiliser de cellule d'abscisse négative.

En effectuant un deuxième repliage similaire selon l'axe des ordonnées on a le résultat voulu.

3.2 Synchronisation passive

Le lemme décrit dans cette section est celui qui sera le plus utilisé dans ce mémoire. Il décrit une construction qui peut déjà être retrouvée dans de nombreux articles dans des cas particuliers. Le but de ce lemme technique est de fournir une méthode générale pour effectuer deux calculs à la suite sans avoir recours à une synchronisation des cellules entre les deux. Avec une machine séquentielle, il est très facile d'effectuer deux programmes à la suite en commençant le deuxième dès que l'exécution du premier est terminée. Avec un modèle fortement parallèle tel que les automates cellulaires un nouveau problème se pose. Lors de la plupart des précalculs que nous effectuons, toutes les cellules ne terminent pas leur calcul à la même étape. Or le choix d'une entrée synchrone présente en entier initialement nous pousse à écrire des algorithmes qui nécessitent cette synchronisation initiale des cellules.

La difficulté dans ce genre de construction est donc de reconnaître le moment où les cellules sont toutes prêtes pour pouvoir lancer le calcul à proprement parler. Puisque la classe de complexité qui nous intéresse principalement est celle du temps réel, ces précalculs sont souvent réalisés en un temps sensiblement inférieur au temps réel. L'utilisation d'algorithmes de synchronisation pour déterminer le moment où il est possible de démarrer le second calcul est alors impossible. Le problème de synchronisation des cellules d'un automate cellulaire a en effet été beaucoup étudié et particulièrement pour les automates en une dimension [Maz96]. Il existe des bornes minimales sur le temps d'exécution d'un algorithme de synchronisation avec un automate cellulaire qui interdisent d'espérer une synchronisation assez rapide pour être utilisée dans des algorithmes en temps réel.

L'algorithme que nous allons présenter ici permet de simuler un calcul avec une entrée donnée de manière synchrone sans avoir recours à une synchronisation extérieure. Cette technique a déjà été utilisée dans de nombreux résultats de la littérature, à chaque fois en l'adaptant au cas particulier étudié. L'idée est de faire commencer le calcul à chaque cellule dès lors qu'elle et ses voisines sont prêtes et contiennent les informations nécessaires. L'algorithme n'est alors plus effectué de manière complètement synchrone ce qui risque de mener à des erreurs d'exécution. Ce problème est généralement réglé en regardant précisément les constructions effectuées, et en exploitant la régularité du précalcul, et notamment l'ordre dans lequel les cellules terminent ce précalcul. Le résultat décrit ici utilise la même technique mais en s'intéressant à un cadre bien plus général. Ce lemme fait partie d'un article publié à ICALP 2016 [GP16] avec V. Poupet.

Lemme 3.5. *Étant donné un automate $\mathcal{A}$, avec un voisinage $\mathcal{V}$ complet, il existe un automate $\mathcal{A}'$ avec le même voisinage qui simule le comportement de l'automate $\mathcal{A}$ même si l'entrée est donnée de manière asynchrone, de telle sorte que toutes les cellules de $\mathcal{A}'$ contiennent l'état de l'automate simulé au moins aussi vite que si le calcul avait démarré de manière synchrone au moment ou la dernière cellule avait reçu son entrée.*

Démonstration. Dans cette preuve nous allons présenter un algorithme de simulation. Il est important de bien noter que nous allons parler différemment d'étapes de calcul et de temps. Une étape de temps correspond à une étape dans l'automate $\mathcal{A}'$, tandis qu'une étape de calcul correspond à une étape dans l'automate $\mathcal{A}$. Ainsi une cellule de $\mathcal{A}'$ qui sera prête au temps n aura terminé son précalcul et effectué 0 étapes de calcul. Ensuite, cette cellule effectuera une étape si toutes ses voisines sont aussi prêtes. D'une manière générale une cellule c qui a effectué i étapes de calcul en effectuera une de plus lors de la prochaine étape de temps si chacune de ses voisines a effectué au moins i étapes de calcul. En effet, si ses voisines ont effectué moins d'étapes, elle est obligée de les attendre pour continuer la simulation. Si ses voisines sont en avance, nous allons faire en sorte que celles-ci se souviennent de leur état à l'étape i pour que la cellule c puisse effectuer son calcul. Cependant, le nombre d'états étant fini, il est impossible pour une cellule donnée de se souvenir de tous ses états antérieurs. Nous nous heurtons donc à la première difficulté : comment être sûrs qu'une cellule se souvient de suffisamment de ses états antérieurs pour que ses voisines en retard ne soient pas bloquées ?

La réponse à cette question réside dans la complétude du voisinage qui permet de borner le nombre d'étape dont une cellule doit se souvenir. En effet, comme le voisinage $\mathcal{V}$ est complet, ACV l'est aussi. Donc il existe un entier k tel que $\mathcal{V} \in (-\mathcal{V})^k$. Cela signifie qu'étant donné une cellule c, toutes ses voisines la voient en au plus k étapes. Pour notre simulation cela signifie que si la cellule c a effectué i étapes de calcul seulement, alors aucune de ses voisines ne peut en avoir effectué plus de $(i+k)$. C'est là notre borne sur le retard que peut prendre une cellule par rapport à ses voisines. Il va alors être nécessaire et suffisant pour chaque cellule de se souvenir de ses k états antérieurs dans la simulation de $\mathcal{A}$. Maintenant que nous avons montré qu'il était possible de garder les informations nécessaires à tout moment, il nous faut trouver un moyen de les exploiter. En plus de ces informations, chaque cellule va disposer d'un compteur qui correspond au nombre d'étapes de calcul qu'elle a effectué modulo $(2k + 1)$. Puisque ce compteur est borné indépendamment du temps ou de la taille de l'entrée, il est facile de l'implémenter au sein des états de notre automate $\mathcal{A}'$. Grâce à ce compteur une cellule va pouvoir déterminer quelles informations utiliser chez les cellules voisines. En effet, comme l'écart entre deux voisines ne peut excéder k étapes de calcul, la donnée de ce compteur modulo $(2k+1)$ permet de déterminer avec exactitude l'écart de calcul entre les deux cellules.

Désormais, nous allons nous intéresser plus précisément à la vitesse à laquelle cet automate va simuler $\mathcal{A}$. Considérons t_0 le temps auquel la dernière cellule devient prête. À cette étape de temps, toutes les cellules sont prêtes, donc toutes les cellules qui n'avaient pas encore effectué d'étape de calcul peuvent en simuler une. Ainsi au temps $(t_0 + 1)$ toutes les cellules ont au moins simulé une étape du calcul. Au temps suivant chaque cellule qui n'en a effectué qu'une seule ne peut être en avance sur aucune de ses voisines, et peut donc à nouveau effectuer une étape de calcul. Une rapide récurrence montre alors le résultat annoncé : au temps $(t_0 + t)$ chaque cellule de $\mathcal{A}'$ a effectué au moins t étapes de calcul de $\mathcal{A}$ ce qui correspond au temps

qu'aurait mis l'automate $\mathcal{A}$ si son entrée avait été donnée de manière synchrone au temps t_0.

□

3.3 Marqueurs flous

Le premier résultat prouvé ici traite de langages dit « à marqueurs ». Il s'agit là de montrer qu'il est possible de préparer l'espace de calcul via un pré calcul pour simplifier ensuite la reconnaissance de certains langages. De plus, on parvient à montrer que sous certaines hypothèses ce pré calcul peut être effectué sans perdre de temps grâce à une compression. Cette construction s'inspire fortement de celle proposée par O. Ibarra et T. Jiang [IJ88]. Elle a été publiée à STACS 2015 [GP15b] dans un article écrit en collaboration avec V. Poupet.

Le résultat concerne les automates en une dimension bien qu'il soit possible de le généraliser en dimension supérieure. Dans ce manuscrit seul le résultat en une dimension sera utilisé. C'est pourquoi, par souci de clarté aussi, ce résultat est présenté dans ce cadre particulier. De plus, on utilisera le voisinage complet usuel puisqu'il est équivalent à tous les autres.

On considère ici une certaine forme de reconnaissance de langage en temps réel. L'idée est d'aider l'automate dans son calcul en lui fournissant une information supplémentaire au début du calcul. Étant donné deux rationnels α et β inférieurs à 1, on dira qu'un langage L est reconnu avec un $\alpha - \beta-$marqueur s'il existe un automate qui reconnaît le langage L en temps réel, à condition qu'au début du calcul une cellule soit marquée et que cette cellule soit comprise entre $\lfloor \alpha n \rfloor$ et $\lfloor \beta n \rfloor$ où n est la taille du mot. Formellement, si L est composé de mots sur l'alphabet Σ, on peut définir un langage $L^{[\alpha,\beta]}$ sur l'alphabet $\Sigma \times \{0,1\}$ qui contient tous les mots dont la première composante est un mot de L, et qui n'ont qu'un seul 1 sur la deuxième, situé sur une cellule comprise entre $\lfloor \alpha n \rfloor$ et $\lfloor \beta n \rfloor$ où n est la taille du mot.

Théorème 3.6. *Soit α et β deux rationnels avec $0 \leq \alpha < \beta \leq 1$, alors $L^{[\alpha,\beta]}$ est reconnaissable en temps réel si et seulement si L est reconnaissable en temps réel.*

Démonstration. Un des sens de l'équivalence est plutôt facile à montrer. En effet, supposons que L soit reconnaissable en temps réel avec un automate cellulaire 1D. Pour reconnaître $L_{\alpha,\beta}$ il suffit de reconnaître L sans se préoccuper de la deuxième composante de l'entrée et, en parallèle, de vérifier qu'une seule case était marquée et qu'elle était au bon endroit dans le mot. Pour cela l'idée principale est d'envoyer deux signaux vers la gauche depuis chaque cellule marquée. Le premier signal se déplacera à vitesse β et le second à vitesse α. Comme α et β sont rationnels et plus petits que 1, il est possible de construire de tels signaux. En même temps, la cellule la plus à droite envoie vers la gauche un signal à vitesse maximale pour reconnaître le temps réel. Puisque β est plus grand que α, le premier signal devrait arriver à l'origine en premier. De plus, si ce signal arrive avant le temps réel n, cela signifie

qu'il s'est déplacé d'au plus $\lfloor \beta n \rfloor$ cellules, et donc que la cellule qui l'a envoyé est située à gauche de celle de coordonnée $\lfloor \beta n \rfloor$. De même, l'arrivée du deuxième signal avant le temps réel signifie que la cellule qui a envoyé le signal se situe à gauche de la cellule de coordonnée $\lfloor \alpha n \rfloor$. Ainsi si l'origine voit arriver un signal à vitesse β et aucun à vitesse α avant le temps réel, le mot peut être accepté en fonction de l'acceptation du mot sur la première couche. Si aucun signal n'arrive, ou si plusieurs signaux arrivent, alors le mot est refusé quoi qu'il se passe sur l'autre couche.

La deuxième partie de la preuve consiste à construire un automate capable de reconnaître L à partir d'un automate capable de reconnaître $L^{[\alpha,\beta]}$. Pour cela nous allons dans un premier temps construire un automate capable de reconnaître L avec l'aide de marqueurs sur un ensemble de cellules indépendant de la taille de l'entrée. Ensuite nous montrerons qu'il est possible de modifier cet automate pour construire ces marqueurs pendant le calcul grâce à une compression.

Soit n_0 le plus petit entier tel que $1 + \frac{1}{2^{n_0}} < \frac{\beta}{\alpha}$. Considérons alors l'ensemble M des entiers dont la représentation binaire est telle que tous les 1 sont sur les (n_0+1) bits de poids fort (voir figure 3.4) :

$$M = \{x \times 2^k \mid x \in [\![0, 2^{n_0+1}-1]\!], k \in \mathbb{N}\} = [\![0, 2^{n_0}-1]\!] \cup \{x \times 2^k \mid x \in [\![2^{n_0}, 2^{n_0+1}-1]\!], k \in \mathbb{N}\}$$

Cet ensemble contient tous les entiers jusqu'à $(2^{n_0+1} - 1)$, ainsi que n'importe quel produit d'un entier dans $[\![2^{n_0}, 2^{n_0+1}[\![$ et d'une puissance de 2.

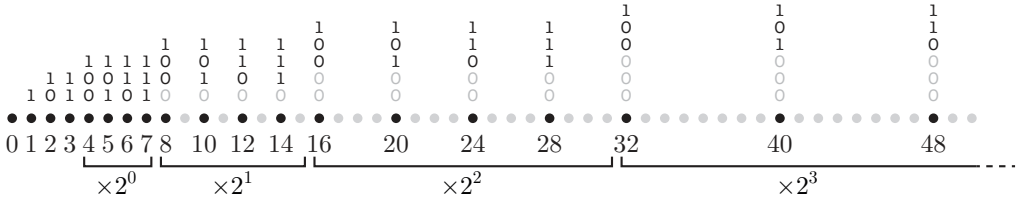

FIGURE 3.4 – L'ensemble M pour $n_0 = 2$.

Nous allons maintenant nous intéresser au rapport entre deux éléments consécutifs de M. On note $(m_i)_{i \in \mathbb{N}}$ les éléments de M dans l'ordre croissant. Nous avons alors, pour tout $m_i \geq 2n_0$:

$$1 + \frac{1}{2^{n_0+1} - 1} \leq \frac{m_{i+1}}{m_i} \leq 1 + \frac{1}{2^{n_0}} \tag{3.1}$$

La borne inférieure correspond au rapport entre le dernier élément d'un bloc (voir figure 3.4) et le premier élément du bloc suivant. La borne supérieure correspond quand à elle au rapport entre les deux premiers éléments d'un bloc. Par définition de n_0, nous avons que

$$\forall m_i \geq 2^{n_0}, \frac{m_{i+1}}{m_i} < \frac{\beta}{\alpha}.$$

Ainsi les intervalles $[\frac{m_i}{\beta}, \frac{m_i}{\alpha}]$ et $[\frac{m_{i+1}}{\beta}, \frac{m_{i+1}}{\alpha}]$ se chevauchent et on peut vérifier que

$$[2^{n_0}, +\infty[\subseteq \bigcup_{m \in M} \left[\frac{m}{\beta}, \frac{m}{\alpha}\right].$$

De ces équations nous pouvons déduire que

$$\forall x \geq 2^{n_0}, \exists m \in M, \lfloor \alpha x \rfloor \leq m \leq \lfloor \beta x \rfloor.$$

Comme M contient aussi tous les entiers inférieurs à 2^{n_0}, nous avons le résultat suivant :

Lemme 3.7. $\forall n \in \mathbb{N}, \exists m \in M, \quad \lfloor \alpha n \rfloor \leq m \leq \lfloor \beta n \rfloor$

Cet ensemble M est donc construit de telle sorte que si toutes les cellules de M sont marquées au temps initial, alors quelle que soit la taille du mot d'entrée, une des cellules marquées constituera un marqueur valide pour reconnaître $L^{[\alpha,\beta]}$. L'idée est alors de construire un automate qui va tester, en parallèle, pour chaque cellule de M si l'entrée avec ce marqueur constitue un mot de $L^{[\alpha,\beta]}$. Le problème qui se pose ici est que le nombre d'éléments de M inférieurs à un n donné n'est pas borné indépendamment de n. Pour résoudre ce problème nous allons nous intéresser au nombre d'éléments de M qui sont pertinents à un temps donné. Effectivement, lorsqu'une cellule apprend que le mot est suffisamment grand, elle est capable de déterminer que certains éléments de M étaient inférieurs à αn sans même connaître la taille finale du mot. Ainsi chaque cellule n'aura à chaque instant qu'un nombre borné de simulations à effectuer en parallèle.

De l'équation 3.1 nous pouvons déduire que : $\forall i, k \in \mathbb{N}$,

$$m_i \left(1 + \frac{1}{2^{n_0+1} - 1}\right)^k \leq m_{i+k} \tag{3.2}$$

Définissons alors k_0 comme étant le plus petit entier tel que :

$$\frac{1}{\alpha} \leq \left(1 + \frac{1}{2^{n_0+1} - 1}\right)^{k_0} \tag{3.3}$$

Ces équations signifient que quel que soit i, si $m_{i+k_0+1} \leq n$ alors $m_{i+1} \leq \lceil \alpha n \rceil$ et donc que $m_i < \lceil \alpha n \rceil$. Alors, pour un mot suffisamment long pour avoir une lettre en position m_{i+k_0+1}, le marqueur situé à la position m_i n'est pas pertinent en tant que candidat pour être un marqueur de $L^{[\alpha,\beta]}$. Ainsi, il n'est jamais nécessaire pour une cellule donnée d'effectuer plus que $k_0 + 1$ simulation en même temps.

Nous allons maintenant pouvoir décrire le fonctionnement de notre automate. Supposons qu'on donne à notre automate un mot w en entrée et que toutes les cellules dont les indices sont dans M sont marquées. Notre automate va commencer une simulation de l'automate $\mathcal{A}$ qui reconnaît $L^{[\alpha,\beta]}$ comme si aucune cellule n'était marquée. En même temps chaque cellule marquée va commencer une simulation

comme si elle était la seule cellule marquée. Chaque cellule qui «voit» une simulation marquée en m_i se met à calculer cette simulation à son tour, en parallèle de toutes les autres qu'elle était déjà en train d'effectuer. Pour cela elle utilise la simulation «vierge» qu'elle était en train d'effectuer. En effet, puisqu'elle n'est pas encore entrée en contact avec la simulation m_i c'est que le marqueur n'a pas encore eu d'influence sur son état dans la simulation, et donc que son état dans la simulation vierge et dans la simulation m_i en question sont les mêmes.

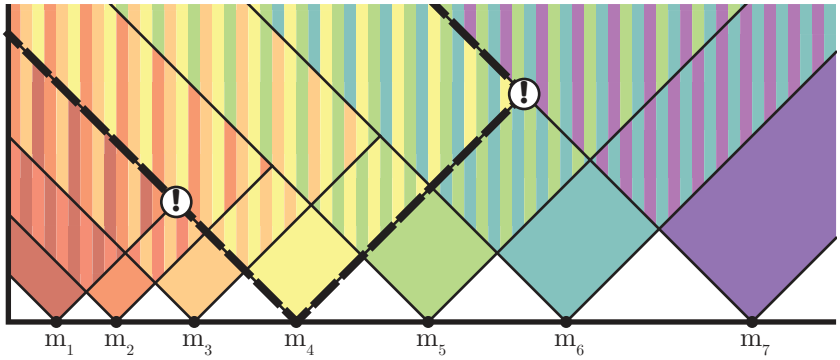

FIGURE 3.5 – Diagramme espace-temps de la simulation. Chaque cône représente une simulation différente, et chaque cellule qui débute un cône était marquée comme appartenant à M.

De cette manière chaque simulation est commencée dès le début et se propage dans l'automate dans les deux sens à vitesse maximale comme illustré sur la figure 3.5. Cependant, chaque cellule ne va effectuer en parallèle que $k_0 + 1$ simulations. Une cellule qui rentre en contact avec une $(k_0 + 2)$-ème simulation venant de la droite va continuer celle ci, et abandonner la dernière simulation qu'elle a vu venant de la gauche. En effet il s'agit de la simulation qui lui vient de gauche la plus lointaine. Notons l'indice de la cellule ayant commencé cette simulation m_j, alors l'indice de la cellule ayant commencé la $(k_0 + 2)$-ème simulation est au moins m_{j+k_0+1} puisqu'il s'agit de la simulation qui vient du plus loin à droite. Ainsi, l'existence même de cette simulation atteste que $m_{j+k_0+1} \leq n$ et donc que le marqueur en position m_j n'était pas pertinent. C'est pour cette même raison que si une cellule voit une $(k_0 + 2)$-ème simulation arriver de la gauche, elle ne continue pas cette simulation.

La figure 3.5 illustre un tel comportement. Chaque cône de couleur correspond à une simulation différente dont le marqueur est situé à l'origine du cône. Les zones hachurées en plusieurs couleurs représentent les zones ou plusieurs simulations ont lieu en même temps. Pour plus de clarté, la simulation vierge commune à toutes les cellules n'est pas représentée. Ici $k_0 = 2$ donc chaque cellule ne va pouvoir effectuer que 3 simulations en parallèle. Les points du diagramme indiqués par des ⓘ sont les points où un conflit apparaît, c'est-à-dire qu'une cellule rencontre une 4-ème simulation. Dans chaque cas, la cellule arrête la simulation qui correspond

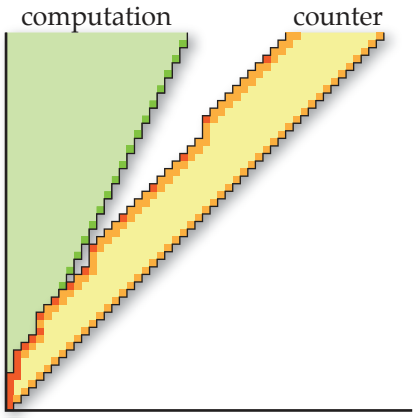

FIGURE 3.6 – Diagramme espace-temps de la compression en vert, et du compteur en jaune.

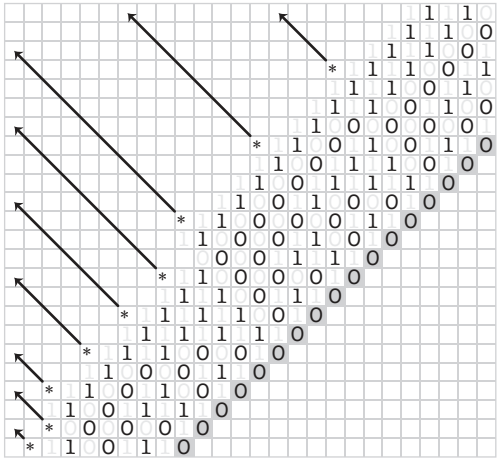

FIGURE 3.7 – Détail du compteur binaire en diagonale sur le diagramme espace temps.

au cône commençant le plus à gauche, et donc au plus petit m_i correspondant. Par construction de M, on sait qu'au moins un marqueur sera situé dans l'intervalle $[\alpha n, \beta n]$, ce qui nous assure que, si le mot d'entrée est un mot de L, alors une des simulation le reconnaîtra comme un mot de $L^{[\alpha,\beta]}$. Notre automate va donc accepter tous les mots pour lesquels au moins une simulation donne une réponse positive.

Maintenant nous allons expliquer comment construire ces marqueurs, sans pour autant ralentir le calcul. Nous allons procéder à l'aide d'une compression pour gagner du temps au début du calcul. On s'intéresse très fortement au diagramme espace temps produit par une telle compression. Celui-ci est illustré en figure 3.6. En parallèle de cette compression, l'automate génère un compteur en binaire qui va numéroter les diagonales du diagramme espace temps. Chaque diagonale correspond à une cellule contenant de l'information compressée. Sur chaque diagonale numérotée va se déplacer un signal dont le rôle est de tester si le numéro est un entier de M, ce qui est facile à faire avec un automate fini et donc faisable par un signal qui se déplace à vitesse maximale. Ensuite ce signal va transmettre cette information vers la gauche jusqu'à la première cellule compressée de cette diagonale (voir figure 3.6 et 3.7). Ainsi, chaque cellule de M sera marquée au moment où les informations qu'elle contient sont compressées et donc avant que le calcul commence pour elle. Le calcul effectué une fois les informations compressées peut donc s'effectuer comme si les cellules voulues avaient été marquées dès le départ.

À partir d'un automate A reconnaissant $L^{[\alpha,\beta]}$ nous avons donc construit un nouvel automate qui commence par compresser son entrée, en marquant certaines cellules, puis qui va simuler le calcul de $\mathcal{A}$ sur plusieurs entrées comme décrit précédemment via les cônes présentés ici. Cet automate va alors reconnaître le langage L sans aucun marqueur.

□

Chapitre 4

Accélérations

Dans ce chapitre nous allons démontrer deux théorèmes d'accélération, c'est-à-dire des théorèmes qui permettent, étant donné un automate reconnaissant un langage, d'en construire un autre qui reconnaît le même langage en moins de temps. Ces résultats démontrent la stabilité de la classe du temps réel et de la classe du temps linéaire. On montrera tout d'abord que l'ajout d'un temps constant de calcul n'aide pas à la reconnaissance de langages avec les voisinages losanges, une classe de voisinage dont fait partie le voisinage de von Neumann. On montrera ensuite que, quel que soit le voisinage, tous les langages reconnaissables en temps linéaire sont en fait reconnaissable en temps $(1+\epsilon)$ TR pour n'importe quel ϵ strictement positif. Ce résultat est le plus important de cette thèse et fait intervenir plusieurs techniques significatives du travail avec des automates cellulaires en dimension supérieure à un.

4.1 Accélération Constante

Le premier résultat de ce chapitre traite de la stabilité de la classe des langages reconnus en temps réel. En effet, telle que nous l'avons définie cette classe ne contient que les langages qui peuvent être reconnus en exactement le temps réel. Cette contrainte semble beaucoup plus stricte que celle imposée pour être reconnu en temps linéaire. La première question qui se pose est de savoir si l'ajout d'un nombre constant d'étapes de calcul permet de reconnaître des langages qui ne sont pas reconnaissable en temps réel. Cela n'est généralement pas le cas pour les modèles de calcul étudiés dans la littérature [Wag86]. Plus récemment il a été montré par V. Poupet dans [Pou06] que ce n'était pas non plus le cas pour certains automates cellulaires en deux dimensions, à savoir, ceux dont le voisinage a la forme d'un rectangle. Il est aussi connu que ce résultat est vrai pour les automates cellulaires utilisant le voisinage de von Neumann.

Dans cette section, dont le contenu a été rédigé sous forme d'un article présenté à AUTOMATA 2016 [Gra16], nous allons montrer que cette propriété est vraie pour une autre classe de voisinages, ceux dont l'enveloppe convexe a la forme d'un losange. Ces voisinages sont en fait une généralisation du voisinage de von Neumann et sont

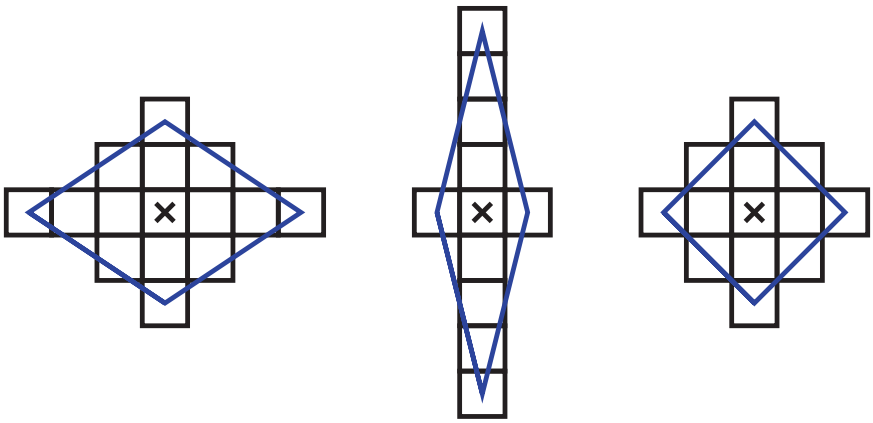

FIGURE 4.1 – Trois voisinages losanges et leur enveloppes convexes

définis comme suit :

Définition 4.1 (Voisinages losanges)**.** *Le* voisinage losange de largeur a et de hauteur b*, noté* $\mathcal{V}_{a,b}$ *est le plus petit voisinage convexe contenant les points* $(a, 0)$*,* $(-a, 0)$*,* $(0, b)$*,* $(0, -b)$*. On peut alors remarquer que le voisinage de von Neumann est exactement* $\mathcal{V}_{1,1}$*.*

Le théorème est alors le suivant :

Théorème 4.2. *Étant donnés un voisinage losange* $\mathcal{V}$ *et un entier positif* k*,* $\mathrm{CA}_{\mathcal{V}}(\mathrm{TR}+k) = \mathrm{CA}_{\mathcal{V}}(\mathrm{TR})$*.*

Comme dans tous les autres modèles, l'idée de la preuve est de construire un automate dans lequel chaque cellule essaye de «deviner» son propre état avec un temps d'avance. Certaines cellules vont réussir à deviner correctement leur état futur et, par effet de bord, leurs voisines vont réussir à leur tour. Ainsi, si l'information se propage assez vite, la cellule origine sera capable de deviner son futur état une ou plusieurs étapes en avance.

Afin de simplifier la construction nous allons montrer qu'il est possible de deviner une étape en avance. Pour en deviner plusieurs, il suffira alors d'utiliser plusieurs fois la construction que nous allons présenter.

Démonstration. Fixons un voisinage losange $\mathcal{V}$ de largeur a et de hauteur b, un langage L et un automate $\mathcal{A}$, avec le voisinage $\mathcal{V}$ qui reconnaît L en une étape de

plus que le temps réel. Nous allons construire un automate $\mathcal{A}'$ qui va simuler le comportement de $\mathcal{A}$, tout en essayant de deviner des informations supplémentaires. En fait, aucune cellule ne va être capable de deviner correctement son état au temps 2 dès le temps 1. À la place les cellules de $\mathcal{A}'$ vont essayer de deviner au temps n les états au temps $(n-1)$ des voisines de leurs voisines. On remarque aisément qu'une cellule qui a deviné correctement tous ces états pourra calculer l'état au temps n de toutes ses voisines et donc son propre état au temps $(n+1)$. De plus, seule la cellule origine a réellement besoin de deviner son état une étape en avance. Un des avantages de cette cellule est de connaître les états de toutes les cellules en dessous d'elle et à sa gauche. En effet, ces cellules sont en dehors de l'espace de calcul et leur état sera toujours #. Ainsi, nous avons juste besoin de deviner les états des cellules qui se trouvent dans le quart nord-est. Pour les mêmes raisons, les cellules qui sont au bord du mot (en haut et à droite) vont réussir à anticiper correctement les états de toutes les cellules dans ce quart de plan qui seront toujours #.

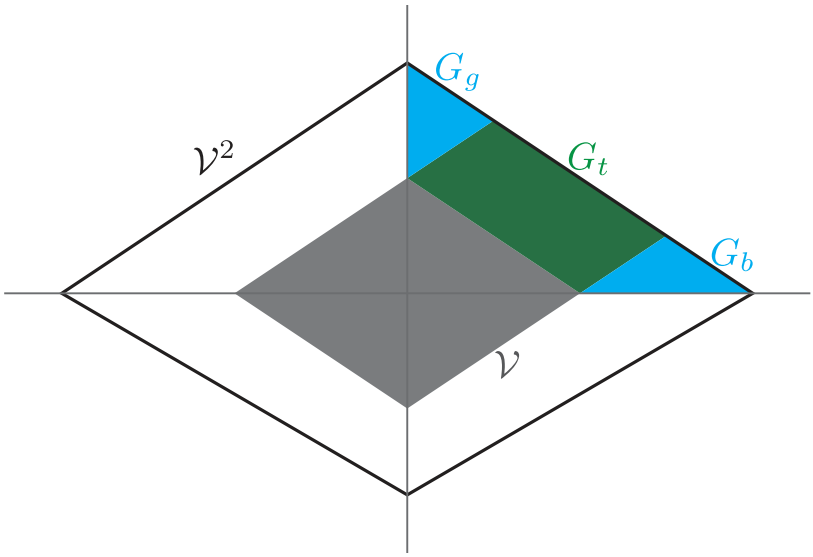

FIGURE 4.2 – Une partition d'un quart de $\mathcal{V}^2$

Nous allons maintenant découper le quart nord-est de $\mathcal{V}^2$ en trois parties : G_t, G_g et G_b. Comme illustré sur la figure 4.2, G_g est l'ensemble des cellules de $\mathcal{V}^2$ qui ne sont pas dans $\mathcal{V}$, qui sont dans le quart nord-est, et dont un voisin se situe en dehors de $\mathcal{V}^2$ et dans le quart nord-ouest. C'est justement ce voisin qui gène les cellules qui veulent deviner des états dans G_g. Encore pour des raisons d'espace de calcul, ce problème est automatiquement résolu pour les cellules de la colonne de gauche. La définition de G_b est similaire mais en considérant les cellules dont un voisin est dans le quart sud-est. G_t pour finir est l'ensemble des cellules du quart nord-est de $\mathcal{V}^2$ qui ne sont ni dans $\mathcal{V}$ ni dans G_b ni dans G_g. Plus formellement on peut définir ces trois ensembles ainsi :

$$G_t = \{(xa, yb) \in \mathbb{Z}^2 \mid 1 < x + y \leq 2; 0 \leq x; 0 \leq y; x \leq y + 1; y \leq x + 1\}$$

$$G_g = \{(xa, yb) \in \mathbb{Z}^2 \mid 1 < x + y \leq 2; 0 \leq x; 0 \leq y; x \leq y + 1; y \geq x + 1\}$$

$$G_b = \{(xa, yb) \in \mathbb{Z}^2 \mid 1 < x + y \leq 2; 0 \leq x; 0 \leq y; x \geq y + 1; y \leq x + 1\}$$

Toutes les cellules c de $\mathcal{A}'$ vont essayer de deviner les états des cellules de $G_t(c)$, seules celles de la colonne la plus à gauche vont s'intéresser aux cellules de $G_g(c)$ et celles de la ligne du bas aux cellules de $G_b(c)$.

Chaque cellule c de $\mathcal{A}'$ va donc, au temps n, contenir :

— l'état de la cellule c au temps n dans $\mathcal{A}$.

— l'état de chaque cellule de son voisinage au temps $(n-1)$ dans $\mathcal{A}$.

— une estimation de l'état de chaque cellule de $G_t(c)$ (et pour certaines $G_g(c)$ et $G_b(c)$) au temps $(n-1)$ dans $\mathcal{A}$.

Maintenir une simulation de $\mathcal{A}$ et se souvenir des états de ses voisines à chaque étape ne pose pas de problème. Maintenir l'estimation, de sorte qu'elle devienne juste au bout d'un certain moment est plus compliqué. Lors de la première étape de calcul toutes les estimations de toutes les cellules seront #. Ensuite, à chaque étape de calcul n, chaque cellule c mettra à jour ses estimations en fonction de la règle de transition de l'automate $\mathcal{A}$. Pour mettre à jour l'estimation de la cellule i, nous avons donc besoin des états des cellules de $\mathcal{V}(i)$ au temps $(n-2)$. Puisque chaque cellule contient l'état de ses voisines au temps précédent, les états des cellules de $\mathcal{V}^2(c)$ au temps $(n-2)$ sont connus de c. Les états des cellules de $\mathcal{V}(i)$ qui ne sont pas dans $\mathcal{V}^2(c)$ ne sont donc pas connus de c. Cependant certaines estimations pour ces états sont accessibles. La cellule c ne va se servir que des estimations des cellules $c + (a, 0)$ et $c + (0, b)$ qui suffisent pour chaque cellule de $G_t(c)$. Formellement, on peut définir $\mathcal{V}(i)$ comme suit :

$$\mathcal{V}(i) = \{i + (xa, yb) \in \mathbb{Z}^2 \mid \|x\| + \|y\| \leq 1\}$$

On note $i = c + (i_x a, i_y b)$ une cellule de $G_t(c)$. Par conséquent, les rationnels i_x et i_y vérifient les inégalités décrites lors de la définition de G_t.

Soit une cellule u dans $\mathcal{V}(i)$. On va montrer que $u \in \mathcal{V}^2(c) \cup G_t(c + (a, 0)) \cup G_t(c + (0, b))$. On a alors $u = c + ((i_x + u_x)a, (i_y + u_y)b)$, avec :

$$\begin{array}{rcl} i_x + i_y & \leq & 2 \\ i_x & \geq & 0 \\ i_y & \geq & 0 \\ i_x & \leq & i_y + 1 \\ i_y & \leq & i_x + 1 \\ \|u_x\| + \|u_y\| & \leq & 1 \end{array}$$

Supposons $i_x + u_x < 0$, c'est-à-dire que la cellule u est à gauche de la cellule c. On a alors :

$$\begin{aligned} \|u_y\| - u_x &\leq 1 \\ i_y - i_x &\leq 1 \\ \|i_x + u_x\| + \|i_y + u_y\| &\leq -i_x - u_x + i_y + \|u_y\| \\ &\leq i_y - i_x + \|u_y\| - u_x \\ &\leq 2 \end{aligned}$$

Et donc que $u \in \mathcal{V}^2(c)$.

Le cas $i_y + u_y < 0$ est similaire.

Dans le cas où la cellule u est en haut à droite de la cellule c et en dehors de $\mathcal{V}^2(c)$ on peut supposer sans perdre de généralité que $i_x + u_x > i_y + u_y$. Dans ce cas on montre que $u \in G_t(c + (a, 0))$, de manière équivalente, que $u - (a, 0) \in G_t(c)$. Nous avons donc 5 propriétés à vérifier :

$$i_y + u_y \geq 0$$

$$\begin{aligned} i_x + u_x + i_y + u_y &\geq 2 \\ i_x + u_x - 1 &\geq 0 \end{aligned}$$

$$\begin{aligned} i_x + u_x - 1 + i_y + u_y &\leq i_x + i_y + u_x + u_y - 1 \\ &\leq 2 \end{aligned}$$

$$\begin{aligned} i_x &\leq i_y + 1 \\ u_x &\leq 1 - \|u_y\| \\ u_x &\leq 1 + u_y \\ i_x + u_x - 1 &\leq i_y + u_y + 1 \end{aligned}$$

$$\begin{aligned} i_y + u_y &\leq i_x + u_x \\ &\leq i_x + u_x - 1 + 1 \end{aligned}$$

Les cellules de la colonne de gauche doivent aussi mettre à jour les estimations des états des cellules de G_g, ce qui est faisable de la même façon car $c+(0, b)$ contient aussi des estimations pour les cellules de G_g et car les états des cellules à gauche de

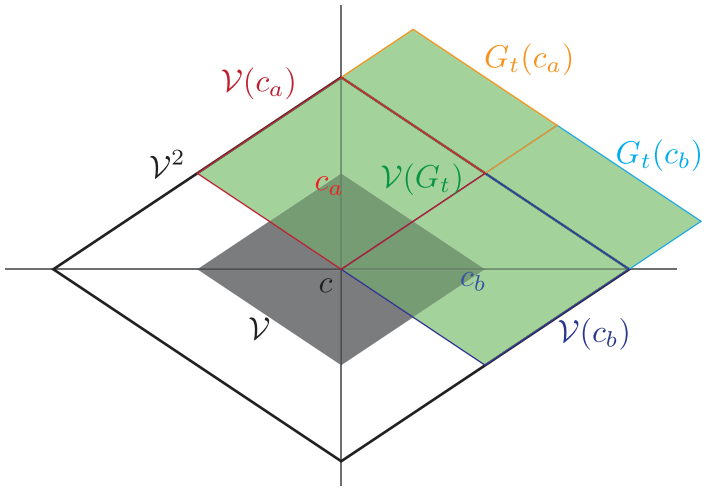

FIGURE 4.3 – Les informations nécessaire à chaque étape

c seront toujours #. Les mêmes arguments fonctionnent pour les cellules de la ligne du bas et pour l'origine (qui est sur les deux à la fois).

Ainsi, si les estimations de $c + (a, 0)$ et $c + (0, b)$ sont justes au temps n, les estimations de c sont justes au temps $(n + 1)$. Après la première étape de calcul, toutes les cellules à l'extérieur du mot ont des estimations correctes. De plus, les cellules de la ligne du haut et celles de la colonne de droite ont aussi des estimations correctes. En effet, on peut remarquer que toutes les cellules de G_t ont une ordonnée et une abscisse strictement positives. Enfin, les cellules qui ont dans leur voisinage la cellule la plus en haut à droite font aussi des estimations correctes. Appelons cette cellule ω.

Toutes les cellules de $\mathcal{V}^n(\omega)$ ont leur voisine la plus haute et leur voisine la plus à droite, soit dans $\mathcal{V}^{n-1}(\omega)$ soit sur la dernière ligne ou la dernière colonne de l'image, soit au-delà. On voit donc qu'au temps n, les estimations des cellules de $\mathcal{V}^n(\omega)$ sont correctes. Or, par définition du temps réel, l'origine est dans $\mathcal{V}^{\mathrm{TR}}(\omega)$ et donc, au temps réel, les estimations de l'origine sont correctes ce qui lui permet de prévoir son état un temps de calcul à l'avance.

□

4.2 Accélération linéaire

Cette section est dédiée à la preuve du théorème le plus significatif du mémoire, le théorème d'accélération linéaire pour tous les voisinages complets. Ces travaux ont été présentées à ICALP 2016 [GP16]. Le théorème présenté ici s'intéresse à la puissance gagnée en multipliant le nombre d'étapes de calcul par un facteur constant. Ce théorème très important permet, entre autre, de justifier l'étude de cette classe de complexité et non pas des classes correspondant à deux fois, trois fois ou quatre fois le temps réel par exemple. Ce résultat est, encore une fois, connu depuis longtemps dans

le cadre des automates en une dimension [IKM85, MR92] et en deux dimensions pour les automates qui utilisent un voisinage dont la forme est un rectangle ou qui utilisent le voisinage de von Neumann [Ter04]. Le théorème démontré dans ce mémoire étend ces résultats à tous les voisinages complets. Cependant cette généralisation a un prix et le théorème annoncé n'est pas tout à fait celui présenté dans la littérature :

Théorème 4.3 (V. Terrier [Ter04])**.** *Étant donné un voisinage avec un seul sommet dans le quart positif $\mathcal{V}$, un réel $\epsilon > 0$ et un langage L. Si L est reconnu par un automate cellulaire en 2D avec le voisinage $\mathcal{V}$ en temps*

$$(n, m) \mapsto \mathrm{TR}_{\mathcal{V}}(n, m) + f(n, m)$$

pour une certaine fonction $f : \mathbb{N}^2 \to \mathbb{N}$, alors le langage L peut être reconnu en temps

$$(n, m) \mapsto \mathrm{TR}_{\mathcal{V}}(n, m) + \lceil \epsilon f(n, m) \rceil$$

par un automate cellulaire avec le voisinage $\mathcal{V}$.

Théorème 4.4. *Étant donné un voisinage complet $\mathcal{V}$, un réel $\epsilon > 0$ et un langage L. Si L est reconnu par un automate cellulaire en 2D avec le voisinage $\mathcal{V}$ en temps*

$$(n, m) \mapsto \mathrm{TR}_{\mathcal{V}}(n, m) + f(n, m)$$

pour une certaine fonction $f : \mathbb{N}^2 \to \mathbb{N}$, alors le langage L peut être reconnu en temps

$$(n, m) \mapsto \lceil (1 + \epsilon) \mathrm{TR}_{\mathcal{V}}(n, m) + \epsilon f(n, m) \rceil$$

par un automate cellulaire avec le voisinage $\mathcal{V}$.

Ces deux versions sont très similaires mais diffèrent néanmoins en un point. Avec le théorème général, il n'est pas possible d'accélérer un calcul dont le temps d'exécution est de la forme $\mathrm{TR} + f$ si f est négligeable devant le temps réel. Cependant, une autre version plus faible mais assez répandue du théorème original ne portant que sur les langages reconnus en temps linéaire existe. L'énoncé est alors le même pour la version généralisée de ce résultat :

Corollaire 4.5. *Étant donné un voisinage complet $\mathcal{V}$, un réel $\epsilon > 0$ et un langage L. Si L est reconnu par un automate cellulaire en 2D avec le voisinage $\mathcal{V}$ en temps*

$$(n, m) \mapsto k \, \mathrm{TR}_{\mathcal{V}}(n, m)$$

pour un certain entier k, alors le langage L peut être reconnu en temps

$$(n, m) \mapsto \lceil (1 + \epsilon) \mathrm{TR}_{\mathcal{V}}(n, m) \rceil$$

par un automate cellulaire avec le voisinage $\mathcal{V}$.

Démonstration. Pour démontrer le théorème 4.4 nous allons, à partir d'un automate $\mathcal{A}$, construire un automate $\mathcal{A}'$ qui va simuler le comportement de $\mathcal{A}$ de telle sorte qu'il reconnaisse le même langage en un temps linéairement plus court grâce à un précalcul qui permet d'accélérer la simulation.

4.2.1 Remarques préliminaires

Pour simplifier la démonstration du théorème nous mettons en avant les remarques suivantes :

Lemme 4.6. *Il est suffisant de montrer le théorème à une constante additive près, c'est à dire que le langage L est reconnaissable en temps*

$$(n, m) \mapsto \lceil (1 + \epsilon) \operatorname{TR}_{\mathcal{V}}(n, m) + \epsilon f(n, m) \rceil + O(1)$$

Démonstration. S'il est possible de montrer ce résultat à une constante additive près, il est assez facile de s'en débarrasser. Pour cela, il suffit de choisir un nouveau ϵ' plus petit que le ϵ voulu de telle sorte que pour (n, m) assez grand, nous avons

$$(1 + \epsilon)TR_{\mathcal{V}}(n, m) + \epsilon f(n, m) > (1 + \epsilon')TR_{\mathcal{V}}(n, m) + \epsilon' f(n, m) + C$$

Toutes les entrées trop petites pour faire marcher l'inégalité peuvent être gérées en temps réel par l'automate puisque leur nombre est fini. □

Lemme 4.7. *Il est suffisant de démontrer le théorème pour les voisinages convexes.*

Démonstration. Étant donné un voisinage $\mathcal{V}$ complet, on considère $\mathcal{V}'$ le voisinage convexe qui a la même enveloppe que $\mathcal{V}$. Les fonctions de temps réel pour ces deux voisinages diffèrent d'au plus une constante. De plus, un automate avec le voisinage $\mathcal{V}$ peut en simuler un avec le voisinage $\mathcal{V}'$ avec un retard constant (voir [DP07] pour plus de détails). Ainsi le résultat du théorème se transfère d'un voisinage à l'autre en perdant au plus un temps constant ce qui peut ensuite être arrangé comme expliqué dans le lemme précédent. □

Dans le reste de la démonstration nous supposerons que $\mathcal{V}$ est un voisinage convexe.

La première étape effectuée par l'automate que nous présentons est la compression de l'entrée. Cette étape a pour but de rassembler les états de la configuration initiale en direction de l'origine, de sorte que chaque cellule en contienne un groupe $(k \times k)$ avec un $k > \frac{1}{\epsilon}$. Le déroulé de cette compression est illustré sur la figure 4.5.

Il est assez facile d'effectuer une compression avec les voisinage usuels de von Neumann et de Moore en envoyant toutes les informations vers l'origine le plus vite possible et en les stockant dès qu'elles arrivent au bon endroit. Cependant, avec un voisinage plus général, il n'est pas possible de savoir *a priori* quelle direction les informations doivent suivre pour se déplacer vers l'origine à vitesse maximale. En toute généralité, cette direction dépend de la proportion $\frac{n}{m}$ de l'entrée. Nous allons d'abord travailler en considérant que la proportion de l'entrée est fixée et montrer que, dans ce cas, il est possible de compresser en temps optimal avec n'importe quel voisinage complet. En accomplissant en parallèle un nombre fini de telles compressions, il sera ensuite possible de compresser en temps presque optimal. En effet,

chaque compression correspond à une proportion donnée et chaque entrée aura une proportion suffisamment proche d'une de ces proportions pour que le temps perdu soit faible. C'est là l'origine du facteur $\epsilon\,\mathrm{TR}$ dans le théorème.

4.2.2 Compression d'une entrée de proportion fixée

Tout d'abord, remarquons qu'il est possible d'effectuer une compression d'un facteur k avec un voisinage tel que celui illustré dans la figure 4.4. Avec ce voisinage, compresser l'entrée consiste simplement à déplacer les états depuis le coin en haut à droite vers l'origine en bas à gauche. Ce faisant l'automate groupe les états par paquets de $(1 \times k)$,$(k \times 1)$ ou $(k \times k)$ lorsque les informations arrivent au bout de leur chemin, c'est à dire qu'elles ne peuvent plus avancer car les cellules dans lesquelles elles devraient aller sont déjà remplies (voir figures 4.4 et 4.5).

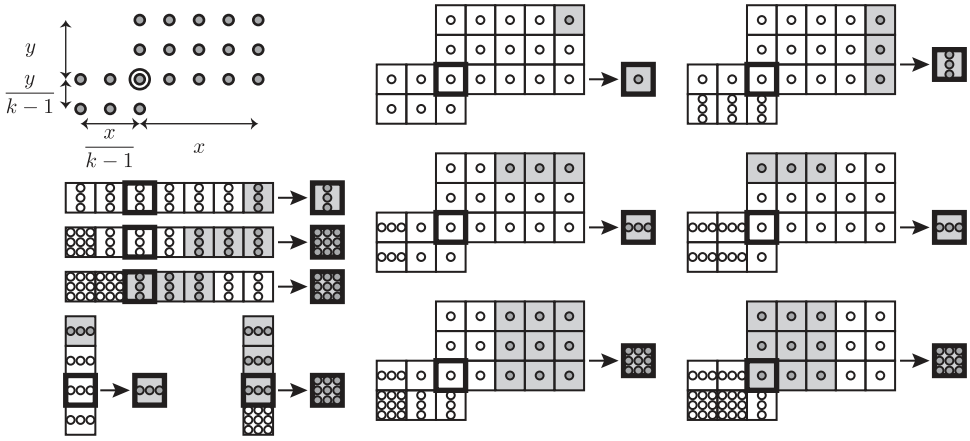

FIGURE 4.4 – Règles pour la compression par un facteur 3, avec le voisinage représenté en haut à gauche. L'information que la cellule conserve comme son nouvel état est représentée en gris. Les informations se déplacent vers le coin en bas à gauche. En regardant en bas à gauche, une cellule détermine quelles informations elle doit laisser passer et lesquelles elle doit conserver. Les règles de la colonne de gauche correspondent à des cas simplifiés lorsque la cellule en question a déjà groupé toute l'information nécessaire dans une direction et, par conséquent, que seules les cellules qu'elle voit dans la direction restante sont significatives.

Nous pouvons remarquer que, pour pouvoir compresser d'un facteur k, une cellule a besoin de voir $\frac{1}{k-1}$ fois aussi loin en bas à gauche qu'elle est capable de voir en haut à droite. En effet, cette information est nécessaire pour savoir quand faire passer les informations et quand il est nécessaire d'en conserver et de commencer à les grouper. Comme l'enveloppe convexe d'un voisinage complet $\mathcal{V}$ contient une boule autour de l'origine (sans quoi $(\mathcal{V}^p)_{p\in\mathbb{N}}$ ne s'agrandirait pas dans toutes les directions), elle contient son image homothétique de ratio $-\frac{1}{k_0}$ pour un certain k_0 et

tous les entiers supérieurs à ce k_0 (voir figure 4.6). Ainsi, avec un tel voisinage une compression par n'importe quel facteur $k > k_0$ est possible.

Pour compresser des entrées de proportion $\frac{n}{m} = \alpha$ avec un voisinage complet quelconque $\mathcal{V}$, nous nous intéressons au plus grand rectangle $[0, x] \times [0, y]$ avec $x, y \in \mathbb{Q}$ et $\frac{x}{y} = \alpha$ qui est inclus dans l'enveloppe convexe de $\mathcal{V}$ (voir figure 4.7). En reprenant le k_0 défini précédemment, le rectangle $[-\frac{x}{k-1}, 0] \times [-\frac{y}{k-1}, 0]$ est aussi inclus dans l'enveloppe convexe de $\mathcal{V}$ pour n'importe quel $k > k_0$. Les rectangles ainsi définis ont des tailles rationnelles mais pas forcément entières. Lorsque nous nous intéressons au voisinage $\mathcal{V}^p$, toutes ces tailles sont multipliées par un facteur p. Ainsi, avec un p suffisamment grand et bien choisi, le rectangle considéré est de taille entière dans le voisinage $\mathcal{V}^p$. Notons de plus que le temps réel, pour des entrées de proportion α est exactement le même sur le voisinage $\mathcal{V}^p$ que sur le voisinage composé uniquement des deux rectangles (ceux aux coordonnées entières). L'algorithme de compression présenté dans les figures 4.4 et 4.5 s'effectue donc en temps $\frac{k-1}{k} \operatorname{TR}_{\mathcal{V}^p} + O(1)$ avec le voisinage V^p. Un automate avec le voisinage $\mathcal{V}$ peut simuler une étape de calcul de l'automate avec $\mathcal{V}^p$ en p étapes de temps. Puisque $\operatorname{TR}_{\mathcal{V}^p} = \lceil \frac{1}{p} \operatorname{TR}_{\mathcal{V}} \rceil$, la compression peut s'effectuer en temps $\frac{k-1}{k} \operatorname{TR}_{\mathcal{V}} + O(1)$ avec le voisinage $\mathcal{V}$.

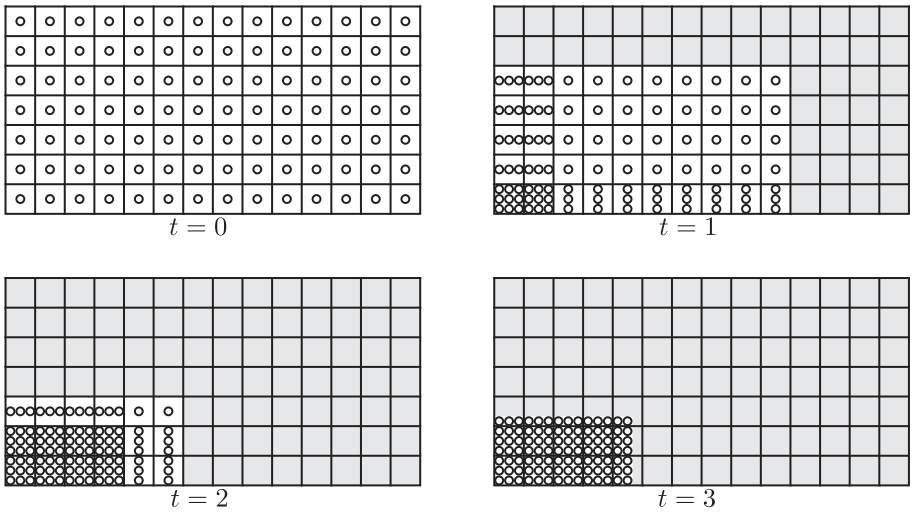

FIGURE 4.5 – Compression d'une entrée de taille (14×7) avec le voisinage et les règles de la Figure 4.4.

4.2.3 Compression d'une entrée quelconque

Nous allons maintenant nous intéresser à des entrées de proportions quelconques. Comme nous l'avons expliqué précédemment, dans le cas ou l'entrée est de proportion α, la direction optimale dans laquelle nous pouvons envoyer l'information pour la compresser est définie par la diagonale du plus grand rectangle $[0, x] \times [0, y]$ qui est

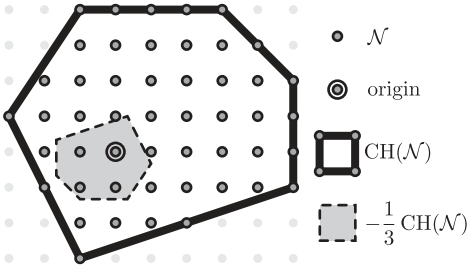

FIGURE 4.6 – Un exemple de voisinage pour lequel un compression par un facteur $k = 4$ est possible.

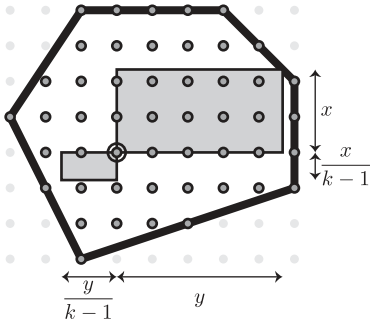

FIGURE 4.7 – Le sous ensemble du voisinage qui est utilisé pour la compression des entrées de proportion $\frac{n}{m} = 2$ par un facteur $k = 4$.

inclus dans l'enveloppe de $\mathcal{V}$ et qui vérifie $\frac{x}{y} = \alpha$. Remarquons ensuite que, comme $\mathcal{V}$ est un voisinage complet, il existe un $\eta > 0$ tel que $[0, \eta] \times [0, \eta] \subseteq \mathrm{CH}(\mathcal{V})$. Ceci implique le fait que chaque rectangle maximal selon une proportion a au moins une dimension supérieure à η. Les coins (x, y) de ces rectangles maximaux sont disposés sur une ligne brisée à l'intérieur de l'enveloppe convexe qui part d'un axe et rejoint l'autre. On peut donc aisément en sélectionner un ensemble fini S de points de cette ligne qui contient les extrémités et tel que deux points de S soient espacés d'au plus $\epsilon\eta$ (voir figure 4.8).

Pour chaque proportion $\alpha = \frac{x}{y}$ avec $(x, y) \in S$, l'automate $\mathcal{A}'$ effectue une compression de l'entrée en simulant le voisinage composé des rectangles maximaux comme décrit dans la section précédente. Toutes les compressions s'effectuent en parallèle. Il est important de remarquer que même les compressions dont la proportion ne correspond pas à l'entrée s'effectuent correctement. Cependant, dans ce cas le temps de compression est plus élevé et non-optimal vis-à-vis du voisinage $\mathcal{V}$. Nous allons donc vérifier qu'au moins une de ces compression s'effectue avec suffisamment peu de retard pour avoir le résultat escompté : $(\frac{k-1}{k} + \epsilon)\,\mathrm{TR}$. Une compression selon le vecteur qui correspond exactement à la proportion de l'entrée prendrait un temps $\frac{k-1}{k}\,\mathrm{TR}$. Une compression par un des vecteurs les plus proches de l'optimal dans S (et donc qui diffère d'au plus $\epsilon\eta$ du vecteur optimal) envoie chaque état de l'entrée à au plus $\epsilon\eta\,\mathrm{TR}$ de leur destination en temps $\frac{k-1}{k}\,\mathrm{TR} + O(1)$. Le temps supplémentaire nécessaire à la compression est un temps de « correction » pendant lequel les informations se déplacent dans une seule direction, verticalement ou horizontalement, dépendant de la position du vecteur utilisé par rapport à l'optimal. En choisissant le bon, il est possible d'effectuer cette dernière étape en temps au plus $\epsilon\,\mathrm{TR} + O(1)$ puisque l'information peut se déplacer à vitesse au moins η. Ce comportement est illustré en figure 4.9. Ainsi, quelle que soit l'entrée, au moins une des compressions sera effectuée intégralement avant le temps $(\frac{k-1}{k} + \epsilon)\,\mathrm{TR} + O(1)$.

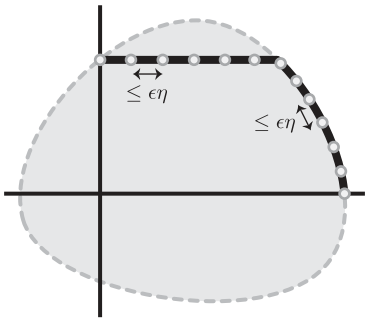

FIGURE 4.8 – Un exemple d'ensemble S qui peut être utilisé pour définir les compressions effectuées par $\mathcal{A}'$. La ligne noire est l'ensemble des coins des rectangles maximaux quelle que soit la proportion d'entrée. Le long de cette ligne nous avons choisi des points à distance au plus $\epsilon\eta$ les uns des autres.

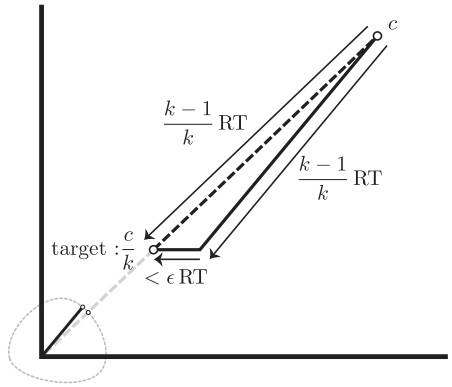

FIGURE 4.9 – Illustration d'une compression d'une entrée de proportion quelconque. La direction optimale est représentée en trait plein. La direction la plus proche dans S est représentée en pointillé. Après $\mathrm{TR}_{\mathcal{V}}$ étapes de calcul, l'information est à une distance de au plus $\epsilon\eta\,\mathrm{TR}_{\mathcal{V}}$ de l'endroit ou elle devrait être si elle s'était déplacée selon le vecteur optimal. Le segment à parcourir pour corriger cette erreur est donc parcourable en au plus $\epsilon\,\mathrm{TR}_{\mathcal{V}}$ étapes.

4.2.4 Simulation sur une entrée compressée

On note ρ la fonction qui associe à chaque cellule de $\mathcal{A}'$ l'ensemble des cellules de $\mathcal{A}$ dont elle reçoit les états initiaux après la compression par un facteur k :

$$\forall c \in \mathbb{Z}^2, \quad \rho(c) = \{kc + (x, y) \mid x, y \in [\![0, k-1]\!]\}$$

Par soucis de simplification nous étendons cette notation aux ensemble de cellules ainsi : $\rho(S) = \bigcup_{c \in S} \rho(c)$.

Les états de $\mathcal{A}$ contenus dans le voisinage d'une cellule c de $\mathcal{A}'$ sont ceux qui correspondent aux cellules de $\rho(c + \mathcal{V})$. Pour être capable de simuler k étapes de l'automate original $\mathcal{A}$, la cellule c de $\mathcal{A}'$ doit être capable de voir les états qui correspondent aux cellules de $(\rho(c) + \mathcal{V}^k)$ de $\mathcal{A}$. Bien que cette contrainte soit respectée pour certains voisinages, comme les rectangulaires, ce n'est pas le cas en toute généralité (voir Figure 4.10).

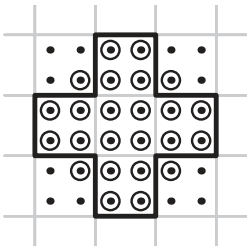

FIGURE 4.10 – Compression par un facteur $k = 2$ avec le voisinage de von Neumann. La ligne épaisse représente $\rho(c + \mathcal{V})$, les états visibles en une étape par la cellule centrale c. Les cellules de $\rho(c) + \mathcal{V}^k$ sont entourées individuellement, elles contiennent les états nécessaire pour effectuer deux étapes de calcul de l'automate original $\mathcal{A}$ pour chaque cellule de $\rho(c)$.

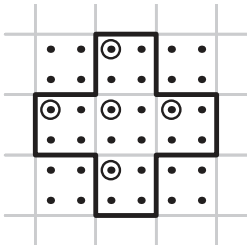

FIGURE 4.11 – La ligne épaisse représente l'ensemble $\rho(c + \mathcal{V})$. Les cellules entourées correspondent à celles de $(c' + k\mathcal{V})$, pour c' la cellule en haut à gauche (entourée aussi) de $\rho(c)$.

Cependant, quel que soit le voisinage, on peut vérifier que $\rho(c) + k\mathcal{V} \subseteq \rho(c + \mathcal{V})$. En effet, pour tout vecteur $v \in \mathcal{V}$, si l'état de la cellule c' de $\mathcal{A}$ est contenu dans la cellule c de $\mathcal{A}'$ après la compression, la cellule $(c + v)$ de $\mathcal{A}'$ contient l'état de la cellule $(c' + kv)$ de $\mathcal{A}$ (voir Figure 4.11).

La solution à ce problème de régularité peut être résolu grâce au lemme suivant :

Lemme 4.8. $\forall k \in \mathbb{N}, \exists \alpha \in \mathbb{N}, \quad \mathcal{V}^\alpha + k\mathcal{V} = \mathcal{V}^\alpha + \mathcal{V}^k$

Démonstration. L'inclusion $\mathcal{V}^\alpha + k\mathcal{V} \subseteq \mathcal{V}^\alpha + \mathcal{V}^k$ est évidente pour tout α. Pour l'autre inclusion, il faut choisir un α suffisamment grand pour que $\alpha + k > |\mathcal{V}|(k-1)$. En

effet, n'importe quel élément $x \in \mathcal{V}^\alpha + \mathcal{V}^k$ peut s'écrire comme la somme de $(\alpha + k)$ éléments de $\mathcal{V}$. Par conséquent, au moins un de ces éléments doit apparaître au moins k fois ce qui prouve que $x \in \mathcal{V}^\alpha + k\mathcal{V}$. □

Grâce à ce lemme, nous pouvons choisir β tel que $\mathcal{V}^\beta + k\mathcal{V} = \mathcal{V}^\beta + \mathcal{V}^k$. Nous allons donc modifier le comportement de l'automate $\mathcal{A}'$ de sorte que, pendant les β premières étapes de calcul, chaque cellule stocke les états de ses voisines. Ainsi au temps β chaque cellule contient tous les états initiaux des cellules de $\mathcal{V}^\beta$. À partir de cette étape, chaque cellule effectue le calcul décrit dans cette section en même temps pour chacun de ces états. À l'étape $\beta + t$, la cellule c contient les états des cellules de $c + \mathcal{V}^\beta$ au temps t dans l'automate $\mathcal{A}'$ avant la modification. En effet, au temps $\beta+t$, les cellules de $c+\mathcal{V}$ contiennent tous les états qui aurait été ceux des cellules de $c+\mathcal{V}^{\beta+1}$ au temps t et donc l'information nécessaire pour mettre à jour tous les états en une étape. Au final, cette modification rajoute un temps de calcul β à l'automate, temps nécessaire au départ pour grouper les états puisqu'ensuite aucun temps n'est perdu par rapport à l'automate précédemment décrit. Ainsi, après la compression de l'entrée, chaque cellule c de $\mathcal{A}'$ contient les états initiaux dans l'automate $\mathcal{A}$ des cellules de $(\rho(c) + \mathcal{V}^\beta)$. À partir de cette étape, chaque cellule peut simuler le calcul dans l'automate $\mathcal{A}$ k fois plus vite. En effet, supposons que toutes les cellules de $\mathcal{A}'$ à temps $t_0 + t$ contiennent les états à temps t des cellules de $\mathcal{A}$ qui sont dans $(\rho(c) + \mathcal{V}^\beta)$. Alors, les cellules voisines de c dans $\mathcal{A}'$ contiennent les états des cellules de $\rho(c + \mathcal{V}) + \mathcal{V}^\beta$ au temps t dans l'automate $\mathcal{A}$. Grâce au lemme 4.8, nous avons :

$$\rho(c + \mathcal{V}) + \mathcal{V}^\beta \quad \supseteq \quad \rho(c) + k\mathcal{V} + \mathcal{V}^\beta \quad \supseteq \quad \rho(c) + \mathcal{V}^{\beta+k}$$

Avec toutes ces informations, la cellule c peut donc calculer les états dans $\mathcal{A}$ des cellules de $\rho(c) + \mathcal{V}^\beta$ au temps $(t + k)$.

4.2.5 Temps total

Nous avons maintenant presque terminé la description de l'automate $\mathcal{A}'$. Le seul point restant est la transition entre la phase de compression et la phase de calcul accéléré. Celle-ci se fait à l'aide du lemme 3.5 du chapitre 3. Ce lemme nous permet d'affirmer qu'on peut construire l'automate $\mathcal{A}'$ de telle sorte que la reconnaissance du mot d'entrée se fasse en un temps borné par la somme des temps nécessaire à chaque étape pour s'effectuer. La première étape de «rassemblement» des informations prend un temps de calcul constant β. La deuxième étape, celle de compression par un facteur k, prend un temps $(\frac{k-1}{k} + \epsilon)\,\mathrm{TR}_\mathcal{V} + O(1)$. Enfin, la simulation, une fois toutes les informations compressées, se fait en temps $\frac{1}{k}(\mathrm{TR}_\mathcal{V} + f) + O(1)$. Le temps total de calcul de l'automate $\mathcal{A}'$ est donc

$$(\frac{k-1}{k} + \epsilon)\,\mathrm{TR}_\mathcal{V} + \frac{1}{k}(\mathrm{TR}_\mathcal{V} + f) + O(1) \leq (1 + \epsilon)\,\mathrm{TR}_\mathcal{V} + \epsilon f + O(1)$$

Le terme en $O(1)$ peut être éliminé grâce au lemme 4.6, ce qui permet de conclure la preuve du théorème 4.4.

□

Chapitre 5

Relations entre classes de langages

Il est assez facile de voir que le modèle des automates cellulaires est équivalent à celui des machines de Turing en terme de puissance de calcul. Cependant ce n'est pas vrai, comme l'on pouvait s'en douter, pour ce qui est de la vitesse de calcul. En effet, le parallélisme massif permet un gain de temps important pour la résolution de certains problèmes qui l'est d'autant plus dans l'étude de classes de faible complexité comme celles du temps réel ou du temps linéaire. Une question naturelle est alors d'expliciter les relations entre ces deux classes. En particulier, la question de l'égalité entre le temps linéaire et le temps réel en une dimension est une question ouverte depuis longtemps mais qui résiste toujours aux chercheurs. Cette question s'exprime assez clairement en dimension un grâce à un résultat de V. Poupet [Pou05] qui prouve que la classe des langages temps réel ne dépend pas du voisinage utilisé en une dimension et donc, comme pour le temps linéaire, que tous les voisinages complets permettent de définir la même classe de langages. Ce n'est, par contre, pas le cas en deux dimensions comme l'atteste un résultat de V. Terrier [Ter99]. La première section de ce chapitre est une adaptation de ce résultat dans un cadre légèrement plus général mais qui servira dans la suite du manuscrit.

5.1 Sommets limitants

Dans cette section nous allons montrer le premier résultat négatif de ce manuscrit. En nous basant sur une construction de V. Terrier [Ter04], nous allons montrer que pour certains voisinages il existe des langages qui sont reconnaissables en temps linéaire mais pas en temps réel.

Pour certains voisinages, on dit qu'un sommet de l'enveloppe convexe est limitant s'il existe des entrées dans lesquelles une cellule est obligée de se déplacer selon ce vecteur à chaque étape de temps pour atteindre l'origine en temps réel. On peut caractériser ces sommets en fonction de l'enveloppe convexe du voisinage de la manière suivante.

Définition 5.1 (Sommet limitant)**.** *Un sommet $v = (a, b)$ d'un voisinage $\mathcal{V}$ est dit*

0	0	0	0	0	0	0	1	0	1	0	0
1	0	0	1	0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	1	0	0	0	0

0	0	0	0	0	0	0	1	0	1	1	0
1	0	0	1	0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	1	0	0	0	0

FIGURE 5.1 – Deux mots de taille 12×12, celui de gauche est dans L_T, pas celui de droite. Les lettres colorées en jaunes sont celles des codes binaires des coordonnées de la lettre à vérifier. Cette lettre est encadrée en rouge.

limitant *si*

— *v est un sommet de l'enveloppe convexe de $\mathcal{V}$.*
— *Le rectangle $[\![0, a]\!] \times [\![0, b]\!]$ est inclus dans $\mathcal{V}$.*

Dans tout ce chapitre, on va s'intéresser à un langage défini par V. Terrier [Ter99].

Définition 5.2. *Étant donné un mot W de taille $(n \times m)$ sur l'alphabet $\{0, 1\}$, on peut considérer une ligne ou une colonne comme la notation binaire d'un nombre. Formellement on note :*

$$x = \sum_{i=2}^{\lceil \frac{m}{2} \rceil} 2^i W(n-1, m-i)$$

$$y = \sum_{j=2}^{\lceil \frac{n}{2} \rceil} 2^j W(n-i, m-1)$$

On définit alors le langage L_T des mots binaires bidimensionnels tels que $W(n - x, m - y) = 1$.

Théorème 5.3 (V. Terrier [Ter04])**.** *Le langage L_T ne peut pas être reconnu en temps réel par un automate avec un voisinage qui contient un sommet limitant.*

Démonstration. La raison fondamentale de cette impossibilité réside dans le parcours des informations dans l'automate au cours du calcul. Lorsque le mot d'entrée est de même proportion α que le sommet limitant $v = (v_x, v_y)$, les informations contenues dans les cellules les plus éloignées de l'origine doivent se déplacer selon ce

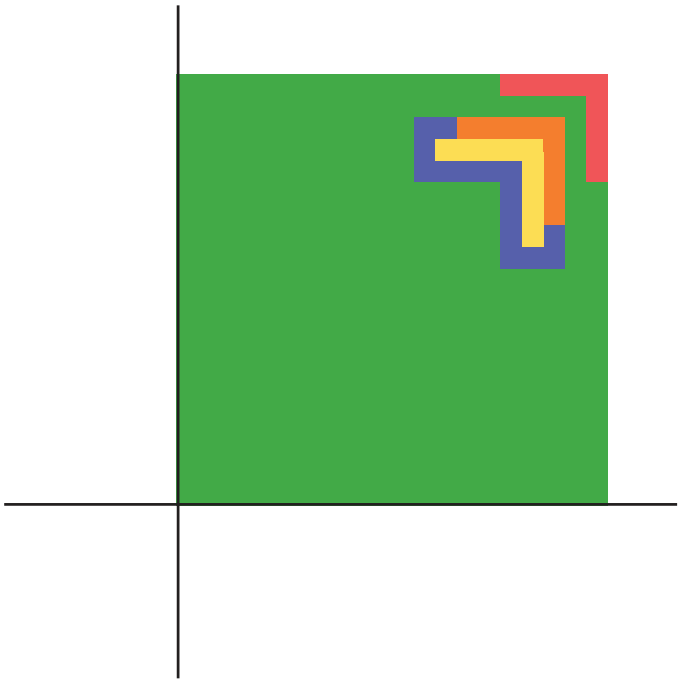

FIGURE 5.2 – Un mot bidimensionnel carré (en vert), l'ensemble des cellules de U en rouge, $K(2)$ en orange, $K(3)$ en jaune et $T(2)$ en bleu (avec le voisinage de Moore).

vecteur v à vitesse maximale pour pouvoir impacter l'état de l'origine au temps réel. Au cours de ce transit, ces informations vont croiser un nombre restreint de cellules, trop petit pour contenir des informations complètes sur le reste de l'entrée. Ainsi certaines entrées seront indiscernables du point de vue de ces cellules alors que les codes binaires peuvent désigner n'importe quelle cellule de l'entrée. Cela va amener une erreur dans la reconnaissance de L_T. Plus formellement, pour une entrée w de taille $n \times m$ nous allons définir les notations suivantes :

$$\begin{aligned}
&\tau \text{ est le temps réel avec le voisinage } \mathcal{V} \text{ sur l'entrée } w\\
&U = \{(x,m) \mid x \in [n - \lceil \log(n) \rceil - 1, n]\} \cup \{(n,y) \mid y \in [m - \lceil \log(m) \rceil - 1, m]\}.\\
&K(t) = \mathcal{V}^{-t}(U) \cap \mathcal{V}^{\tau - t}(0).\\
&T(t) = \mathcal{V}(K(t+1)) \backslash K(t).
\end{aligned}$$

Un exemple avec le voisinage de Moore peut être trouvé dans la figure 5.2

L'ensemble U contient toutes les informations utiles des codes binaires situés sur la dernière colonne et sur la dernière ligne. $K(t)$ est l'ensemble des cellules qui, au temps t, dépendent de l'état initial de U et ont un impact sur la cellule origine à la fin du calcul. $T(t)$ est l'ensemble des cellules qui, au temps t, ne dépendent pas de l'état initial de U et sont nécessaire au calcul de l'état de $K(t+1)$ au temps $t+1$. En effet, comme les cellules de $T(t)$ ont un impact sur l'état de celles de $K(t+1)$, elles ont un impact sur le résultat du calcul et donc seraient dans $K(t)$ si elles dépendaient de

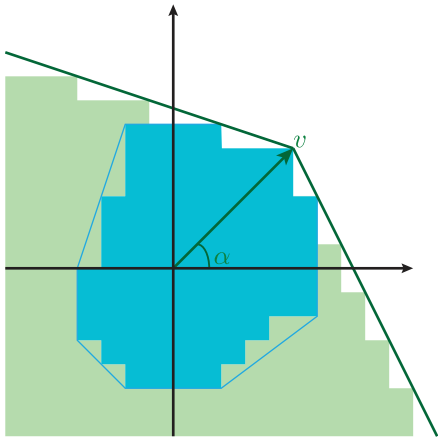

FIGURE 5.3 – Un voisinage avec un sommet limitant et le cône correspondant à ce sommet.

l'état initial de U. On remarque que $K(\tau) = \{0\}$ et, par conséquent, que connaître les états respectifs des cellules de $K(t)$ au temps t suffit à déterminer le résultat du calcul. Par définition de $T(t)$, la connaissance de l'état des cellules de $T(t)$ et de $K(t)$ au temps t suffit à calculer l'état de celles de $K(t+1)$ au temps $t+1$. Or, $K(0) = U$, et donc la donnée de l'état initial de U, ainsi que des états des cellules de $T(t)$ au temps t suffit à déterminer les états de tous les $K(t)$ au temps t, donc l'état de l'origine au temps τ, et ainsi le résultat du calcul. On note $T_w(t)$ les états des cellules dans $T(t)$ au temps t sur l'entrée w. On peut aussi remarquer que $T_w(t)$ ne dépend pas de l'état initial des cellules de U, mais uniquement du reste de w. On dit que deux mots w_1 et w_2 sont équivalents si pour tout t, $T_{w_1}(t) = T_{w_2}(t)$. Pour borner le nombre de classes d'équivalence de ces mots, nous allons nous intéresser à la taille de chaque ensemble $T(t)$ et donc à la taille des ensembles $K(t)$.

Comme v est un sommet de l'enveloppe convexe de $\mathcal{V}$, celui ci correspond à un angle dans l'enveloppe. Cela nous permet de définir un cône qui contient toute l'enveloppe convexe de $\mathcal{V}$ (voir figure 5.3). Étant donné une cellule c, on note $C(c)$ le cône de même angle et de même orientation ancré à la cellule c (ce cône contient la cellule c, comme illustré dans la figure 5.4). De même on note $C^{-1}(c)$ le cône orienté dans l'autre sens par rapport à c, et contenant toujours c (un exemple est aussi dans la figure 5.4).

Comme l'entrée est de proportion α, il existe un rationnel q tel que $(n, m) = qv$. On peut alors remarquer que $\tau = \lceil q \rceil$. En effet, le point le plus éloigné de l'origine dans la direction α et dans $\mathcal{V}^q(0)$ est le point (qv). Or, toute l'entrée est inclue dans $\mathcal{V}^\tau$, et donc en particulier le point (n, m). De plus, comme v est un sommet limitant, toute puissance du voisinage qui contient (n, m) contient tout le mot d'entrée. Cette propriété va nous aider à borner la taille des $K(t)$, en commençant par une approximation de $K(0)$. On note u la cellule $(n - \log(n), m - \log(m))$. On peut vérifier que $C^{-1}(u)$ contient $K(0) = U$. De même $C^{-1}(u - tv)$ contient $\mathcal{V}^t(U)$, et $C((\tau - t)v)$

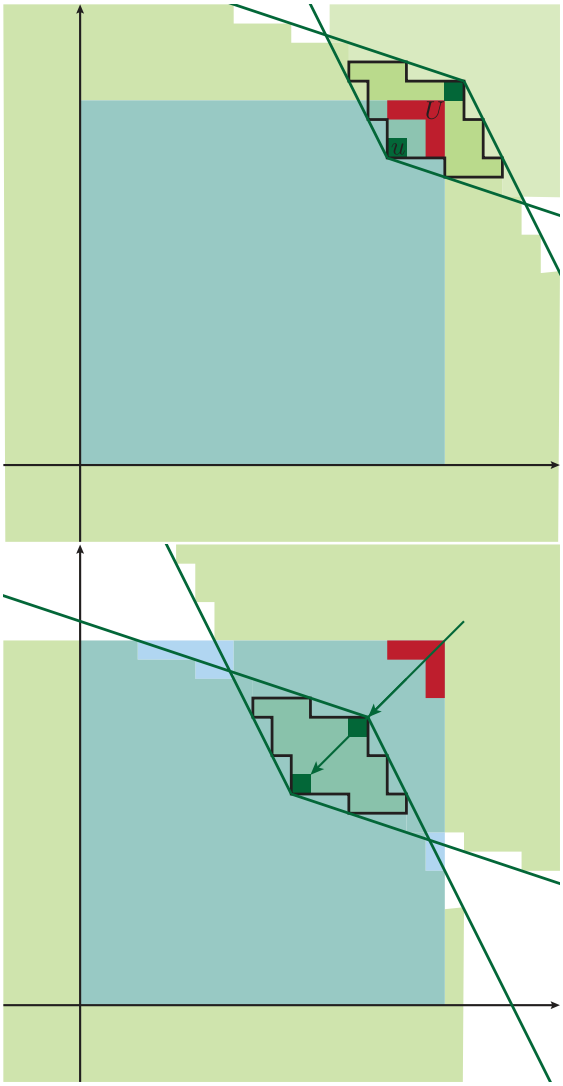

FIGURE 5.4 – Les deux cônes d'approximation, au temps 0 et au temps 1.

contient $\mathcal{V}^{\tau-t}(0)$, donc $K(t)$ est inclus dans $C^{-1}(u-tv) \cap C((\tau-t)v)$. Ces deux cônes sont symétriques et leurs origines sont espacées d'une distance qui ne dépend pas de t. Cette distance est de l'ordre de $|v| + \log(n) + \log(m)$, et donc la taille de $K(t)$ est de l'ordre de $\log(n) + \log(m)$, quel que soit t, et $T(t)$ est au plus de taille $|v| \times |K(t+1)|$.

Notre remarque précédente sur le temps réel nous permet aussi de montrer que $\tau < \lceil \frac{n}{v_x} \rceil$. Ainsi le nombre de couple (cellule, temps) dont l'état est donné dans une suite $(T_w(t))_{t\in\mathbb{N}}$ est de l'ordre de $n\log(n)$ (comme le rapport $\frac{n}{m}$ est fixé on peut se passer de la variable m à l'intérieur des O). Donc, le nombre de classes d'équivalence est au maximum $|\mathcal{A}|^{O(n\log(n))}$.

Il y a $2^{\frac{nm}{4}}$ sous-ensembles différents de $[\![0, \frac{n}{2}]\!] \times [\![0, \frac{m}{2}]\!]$. Chaque mot w défini un de ces sous ensembles en regardant les bits dans les cellules de $[\![0, \frac{n}{2}]\!] \times [\![0, \frac{m}{2}]\!]$. Ainsi, pour

des mots de taille $n \times m$ assez grands, il existe deux mots w et w' qui définissent deux ensembles différents mais qui sont dans la même classe d'équivalence. Ces deux mots diffèrent sur une case $d = (x, y) \in [0, \frac{n}{2}] \times [0, \frac{m}{2}]$ sinon ils définiraient les mêmes ensembles. Le mot w dans lequel on modifie U pour y mettre les codes de $(n-x, m-y)$ est aussi dans la même classe d'équivalence et de même si on modifie pareillement w'. Ces deux mots sont soit tous les deux acceptés soit tous les deux refusés, alors qu'un des deux devrait être accepté, celui qui a un 1 dans la cellule d, et l'autre devrait être refusé. Ce qui amène à la contradiction voulue, l'automate $\mathcal{A}$ ne reconnaît pas le langage L_T. □

La démonstration présentée ici diffère légèrement de celle présentée par V. Terrier qui était plus précise dans le comptage du nombre de cellule dans $T(t)$. En fait la démonstration telle qu'écrite dans ce mémoire permet de prouver un résultat légèrement plus fort que celui annoncé. En effet, il n'est pas nécessaire que le code soit situé exactement sur la dernière ligne et la dernière colonne de l'entrée. Il suffit, pour que la preuve fonctionne, que les codes des deux coordonnées soient compris dans un rectangle de taille $O(\log(n)\log(m))$ dans le coin en haut à droite de l'entrée. De plus, il n'est pas nécessaire non plus que ce code puisse désigner n'importe quelle cellule de l'entrée. L'argument de comptage nécessite simplement que l'ensemble des cellules codables soit de taille $O(nm)$. Enfin, on remarque que l'on peut restreindre un tel langage aux mots de la proportion adéquate (la proportion du sommet limitant considéré). Plus tard dans ce mémoire, nous nous servirons de ce résultat pour prouver l'impossibilité de reconnaître des langages qui correspondent à ces critères.

5.2 Séparation utilisant les sommets limitants

Considérons maintenant le voisinage de von Neumann, qui n'a pas de sommet limitant. On peut alors prouver le résultat suivant :

Proposition 5.4 (V. Terrier [Ter04])**.** *Le langage L_T peut être reconnu en temps réel par un automate avec le voisinage de von Neumann.*

La preuve de ce théorème est due à V. Terrier et peut être trouvée dans [Ter99]. Nous en rappellerons ici les grandes lignes, afin de pouvoir plus facilement en proposer une version modifiée dans le chapitre suivant. L'automate va effectuer trois tâches en parallèle. Tout d'abord, il faut numéroter les lignes et les colonnes en partant de la fin. Pour cela, en commençant par la cellule du coin en haut à droite, chaque colonne va incrémenter le numéro de la colonne précédente. Le i-ème bit de la colonne l se calcule aisément en fonction du i-ème de la colonne $(l-1)$ et du $(i-1)$-ème de la colonne l. Ainsi le i-ème bit de la colonne l est écrit au temps $(l+i)$. Pendant ce temps, l'automate recopie sur chaque ligne et sur chaque colonne une copie de la dernière ligne ou colonne. Ainsi la l-ème colonne obtient sa copie au temps l. Dès que ce recopiage est effectué, un signal de comparaison est lancé du bord comparant le numéro et la copie. Avec la méthode de comptage précédente,

ce signal dispose effectivement de toutes les informations nécessaires au moment d'effectuer la comparaison bit à bit. Enfin, sur chaque ligne et chaque colonne deux signaux vont partir des extrémités pour trouver et marquer le milieu de ladite colonne ou ligne. Lorsque le signal de comparaison atteint ce milieu il a terminé sa tâche et, si la comparaison était juste, il continue son chemin en attendant de croiser un autre signal similaire. Ainsi, le signal qui correspond à la colonne $(n - x)$ sera parti au temps x de la cellule $(n - x, m)$ et celui qui correspond à la ligne $(m - y)$ sera parti au temps y de la cellule $(n, m - y)$. Ces deux signaux se rencontreront donc au temps $(x + y)$ à la case $(n - x, m - y)$. Si cette cellule a pour état 1, ces signaux se transforment en un signal d'acceptation qui se propage jusqu'à l'origine.

Théorème 5.5 (V.Terrier [Ter04])**.** *Étant donné un voisinage $\mathcal{V}$ complet avec un sommet limitant, il existe un langage reconnaissable en temps linéaire avec $\mathcal{V}$ mais pas en temps réel.*

Démonstration. Le langage L_T étant reconnaissable en temps réel avec le voisinage de von Neumann, il est reconnaissable en temps linéaire avec tout voisinage complet $\mathcal{V}$. Comme nous avons montré qu'il n'était pas reconnaissable en temps réel avec un voisinage avec un sommet limitant, il constitue un témoin du théorème. □

Les deux sections suivantes portent sur un problème similaire mais légèrement différent : comment se comportent des automates qui disposent d'une dimension de calcul plus grande que la dimension de l'entrée ? En effet on peut tout à fait considérer un langage en dimension d comme un langage de dimension $d' > d$ dont tous les mots sont de taille 1 dans les dimensions rajoutées.

5.3 Entre la dimension un et la dimension deux

Dans cette section nous allons nous intéresser à de la reconnaissance de langages en une dimension (1D) avec des automates cellulaires en deux dimensions (2D) et qui utilisent le voisinage de Moore. Certaines études ont déjà été menées dans la littérature sur la reconnaissance de mots 1D lorsqu'ils sont transformés en image 2D selon différentes représentations (ligne à ligne, fil de Hilbert, ...[DM02]). Ces études s'intéressent aussi à la puissance qu'apporte la deuxième dimension mais dans ces travaux l'entrée est transformée pour couvrir une surface en deux dimensions. Dans le cadre de nos travaux, les deux automates de dimensions différentes vont travailler sur la même entrée, disposée exactement de la même manière. De cette façon on peut notamment remarquer que le temps réel est le même quelle que soit la dimension considérée. Les résultats de cette section ont fait l'objet d'un article cosigné avec V. Poupet et présenté à STACS 2015 [GP15b].

Même en ne considérant que des calculs en temps réel, il est possible pour un automate avec le voisinage de Moore d'utiliser un nombre de cellules quadratique en la taille de l'entrée en même temps, en comptant uniquement celles qui ont

une influence sur le résultat du calcul. Cela n'est pas le cas pour un automate en une dimension, et à priori pas le cas non plus si l'on considère une entrée en deux dimensions. Le but de ces travaux est donc de réussir à déterminer quelle puissance on peut gagner grâce à cet espace. On revient sur la notion de langage marqué présentée dans la section 3.3. Étant donné un langage marqué on défini le langage sans marque suivant :

Définition 5.6. *Étant donné un langage marqué* $L \subseteq (\Sigma \times \{0,1\})^*$*, on définit* $\widetilde{L} \subseteq \Sigma^*$ *comme le langage obtenu en enlevant les marques des mots de* L *(on peut donc le voir comme une projection de* L *sur la première composante).*

Ce qui nous permet de formuler les théorèmes suivants :

Théorème 5.7. *Étant donné un langage* $L \subseteq (\Sigma \times \{0,1\})^*$ *de mots marqués à au plus une position,* $L \in \mathrm{CA}_1(\mathrm{TL}) \Rightarrow \widetilde{L} \in \mathrm{CA}_2(\mathrm{TL})$.

Démonstration. Soit $L \subseteq (\Sigma \times \{0,1\})^*$ un langage de mots avec au plus une position marquée et soit $\mathcal{A}$ un automate unidimensionnel qui reconnaît L en temps $n \mapsto 2n$. Nous allons construire un automate $\mathcal{A}'$, en deux dimensions, qui reconnaît $\widetilde{L}$ en temps linéaire. L'idée pour cela est de simuler l'automate $\mathcal{A}$ pour chaque position possible de la marque comme illustré sur la figure 5.5. En effet, si une de ces simulations aboutit à une acceptation, c'est qu'il existe une position pour laquelle le mot marqué appartient à L et donc que le mot appartient à $\widetilde{L}$. L'entrée de notre automate $\mathcal{A}'$ est donc un mot $w \in \Sigma^*$ de longueur n. Nous allons nous servir de la deuxième dimension de $\mathcal{A}'$ pour effectuer n simulations de l'automate A en même temps.

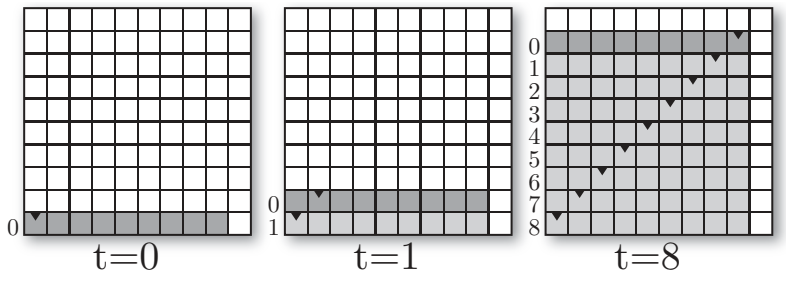

FIGURE 5.5 – Exécution de n simulations d'un automate unidimensionnel avec un automate bidimensionnel.

Au temps $t = 0$, l'automate $\mathcal{A}'$ va commencer une simulation de l'automate $\mathcal{A}$ sur la première ligne comme si la cellule la plus à gauche portait une marque. Cette opération va se faire sur une nouvelle couche de calcul de manière à ne pas effacer l'entrée. A chaque étape suivante l'entrée originale va être recopiée sur la ligne au dessus, en déplaçant la marque d'une cellule vers la droite, comme illustré sur la figure 5.5. Après ce recopiage, une nouvelle simulation commence sur la ligne en question, toujours sur une couche différente de celle qui sert à recopier l'entrée. Ainsi,

au temps $t = n$, la première ligne a effectué n étapes de simulation de l'automate $\mathcal{A}$ et la n-ème ligne commence une nouvelle simulation avec une marque sur la dernière cellule.

Lorsqu'une simulation est terminée sur une ligne, le résultat est envoyé le long de la première colonne en direction de l'origine. Au temps $t = 3n$, la simulation se termine sur la n-ème ligne. L'information met ensuite n étapes de calcul à arriver à l'origine de $\mathcal{A}'$. Ainsi au temps $t = 4n$, l'origine a reçu le résultat de chaque simulation et peut donc accepter le mot d'entrée si au moins une des simulations a mené à une acceptation. Le langage $\widetilde{L}$ est donc reconnu en temps $n \mapsto 4n$, ce qui, grâce au théorème d'accélération linéaire présenté précédemment, nous donne $\widetilde{L} \in \mathrm{CA}_2(\mathrm{TL})$.

□

Théorème 5.8. *Étant donné un langage $L \subseteq (\Sigma \times \{0,1\})^*$ de mots marqués à au plus une position, $L \in \mathrm{CA}_1(\mathrm{TR}) \Rightarrow \widetilde{L} \in \mathrm{CA}_M(\mathrm{TR})$.*

Démonstration. Le principe général de cette preuve est le même que pour le théorème précédent. Cependant, la contrainte de temps réel nous oblige à utiliser une construction plus astucieuse pour ne pas perdre de temps. En effet, l'algorithme précédent a besoin de temps supplémentaire pour recopier l'entrée, marquer les diverses positions possibles et ensuite ramener le résultat du calcul à l'origine sur la ligne la plus basse. Pour effectuer ces opérations sans perdre de temps, nous allons encore une fois utiliser une compression. Le temps passé à compresser sera regagné par la suite, le calcul pouvant être simulé plus efficacement. L'entrée va être recopiée et les marques distribuées pendant cette compression. Pour ne pas perdre de temps à faire redescendre le résultat vers la première ligne, nous allons effectuer une compression au milieu plutôt qu'une compression à gauche comme toutes celles qui ont été présentées jusqu'à présent. En effet, compresser de la sorte se fait un peu plus vite : les informations venant du bord droit ont moins de distance à parcourir, celle venant de la gauche en ont désormais autant à parcourir ce qui permet de mieux utiliser tout le parallélisme dont nous disposons. Ce temps gagné lors de la compression est en général perdu puisqu'il faut ensuite ramener le résultat du calcul à l'origine mais, dans ce cas particulier, ce déplacement vers la gauche peut être couplé avec un déplacement vers le bas sans perdre de temps.

Comme la compression va avoir lieu ligne par ligne, nous allons la décrire pour un automate 1D. De plus, pour pouvoir compresser au centre, nous allons avoir besoin de l'aide d'une marque située approximativement au milieu qui, grâce au théorème 3.6, pourra ensuite être enlevée. L'automate que nous présentons va compresser par un facteur 3 autour de cette marque comme illustré sur la figure 5.7. Comme les cellules ne savent pas à l'avance si elles sont à gauche ou à droite de la marque, l'automate va utiliser deux couches, une sur laquelle les informations se déplacent vers la gauche et une sur laquelle les informations se déplacent vers la droite. Les informations vont se compresser autour de la cellule marquée lorsqu'elles se déplacent dans la

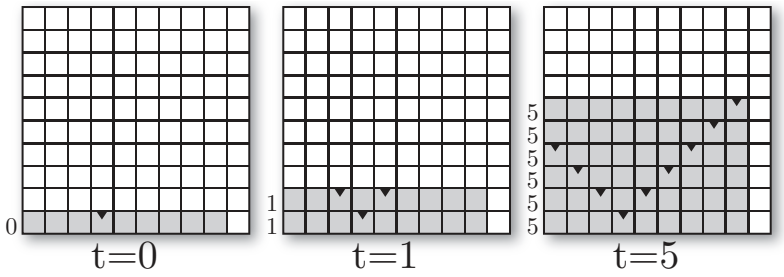

FIGURE 5.6 – Simulations en parallèle et en temps réel d'un automate cellulaire 1D. Chaque ligne comprend 6 copies de l'entrée, chacune avec une cellule marquée différente.

bonne direction. Celles qui partent dans la mauvaise direction ne vont jamais être compressées et n'interagiront pas avec le reste du calcul.

Supposons que la cellule marquée soit la cellule $\lfloor \alpha n \rfloor$, le temps nécessaire à cette compression est $\max(\frac{2}{3}\alpha n, \frac{2}{3}(1-\alpha)n)$. Une fois le calcul effectué, le résultat doit être envoyé vers l'origine à gauche de l'entrée. Ceci prend à nouveau un temps $\frac{2}{3}\alpha n$. Le temps de calcul, une fois l'entrée compressée, pour simuler un automate qui donne son résultat en temps réel n est donc $\frac{n}{3}$. Le temps total de calcul pour faire la compression, la simulation, puis ramener le résultat est donc

$$\max(\frac{2}{3}\alpha n, \frac{2}{3}(1-\alpha)n) + \frac{2}{3}\alpha n + \frac{n}{3}$$

Si α est plus grand que $\frac{1}{2}$, ce temps est plus grand que le temps réel n. Sinon, c'est l'autre terme qui l'emporte lors de la compression et le temps total est alors toujours égal à n.

La durée de la compression va servir à recopier l'entrée sur la deuxième dimension de l'automate. Comme cette compression ne dure pas n étapes, il est impossible de recopier les lignes une par une comme dans l'algorithme précédent. À la place, chaque ligne de l'automate 2D va contenir six lignes de l'automate 1D, compressées d'un facteur 3. De cette manière chaque cellule va, au final, contenir 18 états de l'automate $\mathcal{A}$. De plus, chaque recopiage va copier la version déjà compressée de la ligne précédente. Ainsi toutes les lignes seront compressées exactement en même temps, et à la fin de cette compression n copies de l'entrée seront présentes sur les $\frac{n}{6}$ premières lignes de l'automate 2D.

En même temps, on veut apposer une marque différente sur chaque copie de l'entrée. Contrairement à la construction précédente, nous allons commencer par marquer la cellule autour de laquelle se fait la compression. Lorsque cette cellule termine son groupement, c'est-à-dire qu'elle contient 18 états de $\mathcal{A}$, elle marque trois des six copies de l'entrée, chacune sur une cellule différente qu'elle contient. Ensuite, c'est le tour de ses voisines gauche et droite de la ligne du dessus. Lorsqu'elles seront compressées elles marqueront chacune trois des six copies présentent sur cette ligne. De cette manière on peut éviter qu'une copie soit marquée deux fois. C'est ensuite au

tour de la ligne encore au dessus d'effectuer la même manipulation, comme illustré sur la figure 5.6. Chaque copie est donc marquée avant la fin de la compression, moment où commence réellement le calcul.

Après cette compression, l'automate va effectuer une simulation accélérée de l'automate $\mathcal{A}$ sur chaque ligne, ce qui prend un temps $\frac{n}{3}$. La transition entre la compression et la simulation est une fois de plus assurée par le lemme de synchronisation 3.5.

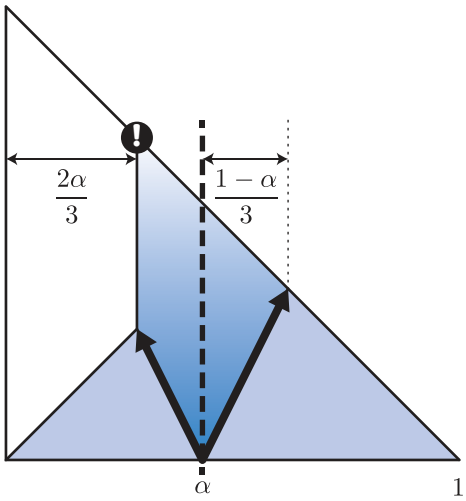

FIGURE 5.7 – Diagramme espace-temps représentant la compression autour d'une cellule marquée à la position $\lfloor \alpha n \rfloor$. Les flèches illustrent les points du diagramme où les informations de l'entrée sont compressées pour la première fois. Le point d'exclamation représente le site où le résultat du calcul apparaît.

Enfin, l'automate doit ramener les résultats de chaque simulation à l'origine. Pour cela, une fois la simulation terminée, chaque ligne va envoyer son résultat en direction de la cellule origine vers le bas et vers la gauche en même temps. Le temps nécessaire à ramener l'information sur la colonne de gauche est déjà décompté dans les n étapes de l'algorithme. Il faut donc que chaque résultat ait moins de distance verticale à parcourir que de distance horizontale. La simulation qui s'effectue le plus loin de l'origine s'effectue sur la $\frac{n}{4}$-ème ligne. Ainsi il faut s'assurer qu'après la compression, l'entrée soit intégralement à droite de la cellule $\frac{n}{4}$. Cette contrainte donne une borne sur la valeur minimale de α. Ainsi, prendre α entre $\frac{1}{3}$ et $\frac{1}{2}$ est suffisant pour que l'algorithme s'effectue correctement sans perdre de temps.

L'automate $\mathcal{A}'$ que nous avons décrit est donc capable de compresser une entrée en 1D, tout en la recopiant n fois, d'attribuer à chacune de ces copies une marque sur une cellule différente, puis de simuler l'automate $\mathcal{A}$ sur chacune de ces copies. Enfin, le mot est accepté si au moins une des simulations accepte le mot. Ce calcul est terminé à l'étape n puisque l'automate $\mathcal{A}$ reconnaît L en temps réel et donc termine son calcul en temps n. Ainsi, nous avons montré un automate $\mathcal{A}'$ capable de reconnaître $\widetilde{L}^{[\frac{1}{3},\frac{1}{2}]}$ en temps réel. En effet nous avons supposé la présence d'un

marqueur situé entre le tiers et la moitié du mot d'entrée. Le théorème 3.6 assure alors qu'il est possible de construire un automate $\mathcal{A}''$ qui reconnaît $\widetilde{L}$ en temps réel. □

5.3.1 Conséquences

Le théorème que nous avons prouvé peut paraître assez spécifique puisque cette notion de marqueurs a été très peu étudiée dans la littérature. Il est en fait possible de déduire de nombreux résultats de cette formalisation. Cette section est dédiée à quelques exemples des langages qu'il est possible de reconnaître grâce à ces marqueurs.

Corollaire 5.9. *La concaténation L_1L_2 de deux langages reconnaissables en temps réel en une dimension est reconnaissable en temps réel en deux dimensions avec le voisinage de Moore. Formellement :*

$$L_1 \in \mathrm{CA}_1(\mathrm{TR}) \wedge L_2 \in \mathrm{CA}_1(\mathrm{TR}) \Rightarrow L_1L_2 \in \mathrm{CA}_M(\mathrm{TR})$$

Démonstration. Étant donné un mot $w = uv$ avec une marque entre u et v, il est facile de vérifier en temps réel et en une dimension si $u \in L_1$ et $v \in L_2$ puisque ces langages sont reconnaissables en temps réel. La cellule marquée joue le rôle d'un # dans la première reconnaissance et de l'origine dans la deuxième. Grâce au théorème 5.8, ce langage sans la marque est reconnaissable en temps réel avec un automate en deux dimensions. □

Corollaire 5.10. *Si* $\mathrm{CA}_1(\mathrm{TR}) = \mathrm{CA}_M(\mathrm{TR})$ *alors* $\mathrm{CA}_1(\mathrm{TR})$ *est clos par concaténation.*

Sans l'hypothèse $\mathrm{CA}_1(\mathrm{TR}) = \mathrm{CA}_M(\mathrm{TR})$, la stabilité par concaténation de $\mathrm{CA}_1(\mathrm{TR})$ n'est toujours pas connue. En fait, il n'existe même aucun résultat qui assure que la concaténation de deux langages reconnaissables en temps réel est reconnaissable en temps linéaire. En effet l'algorithme naïf qui teste toutes les possibilités est quadratique.

Corollaire 5.11. *Si* $\mathrm{CA}_1(\mathrm{TR}) = \mathrm{CA}_M(\mathrm{TR})$ *alors pour tout langage L et tout $\alpha \in \mathbb{Q} \cap [0,1]$, si $L^{[\alpha,\alpha]} \in \mathrm{CA}_1(\mathrm{TR})$ alors $L \in \mathrm{CA}_1(\mathrm{TR})$.*

Démonstration. Par le théorème 5.8 nous avons $L^{[\alpha]} \in \mathrm{CA}_1(\mathrm{TR}) \Rightarrow L \in \mathrm{CA}_M(\mathrm{TR})$. L'hypothèse d'égalité nous donne le résultat annoncé. □

Ce corollaire signifie que si ces deux classe sont égales, alors les automates cellulaires en une dimension ne profitent pas de l'aide que peut leur apporter une marque à un endroit précis en temps réel. Ce genre de marque peut permettre de connaître à l'avance le milieu de l'entrée par exemple. Bien que nous n'ayons pas trouvé de contradiction avec ce fait, nous avons espoir de réussir à prouver que ces marques ajoutent de la puissance de calcul aux automates en temps réel.

Cette hypothèse $\mathrm{CA}_1(\mathrm{TR}) = \mathrm{CA}_M(\mathrm{TR})$ peut en fait être utilisée pour renforcer le théorème précédent de la manière suivante :

Corollaire 5.12. *Si* $\mathrm{CA}_1(\mathrm{TR}) = \mathrm{CA}_M(\mathrm{TR})$ *alors pour tout entier* $k \in \mathbb{N}$ *et tout langage* $L \subseteq (\Sigma \times \{0,1\})^*$ *de mots ayant au plus* k *positions marquées,* $L \in \mathrm{CA}_1(\mathrm{TR}) \Rightarrow \widetilde{L} \in \mathrm{CA}_M(\mathrm{TR})$.

Démonstration. La preuve se fait par récurrence sur l'entier k. Le cas $k = 1$ est une conséquence directe du théorème 5.8, et de l'hypothèse $\mathrm{CA}_1(\mathrm{TR}) = \mathrm{CA}_M(\mathrm{TR})$. Supposons que ce corollaire soit vrai pour les langages de mots avec au plus k marques. Soit L un langage de mots avec $(k+1)$ marques. On définit le langage $L' \subseteq (\Sigma \times \{0,1\}^2)^*$ à partir du langage L comme suit. Soit un mot $w \in L$, on change la première lettre marquée $(u_1, 1)$ par la même lettre marquée sur la deuxième couche de marqueurs $(u_1, 0, 1)$. Toutes les autres lettres marquées restent marquées sur la première couche mais pas sur la deuxième. Les lettres qui ne sont pas marquées restent ainsi. De cette manière, nous avons construit un mot w' dont la première marque est particulière. Puisque $L \in \mathrm{CA}_1(\mathrm{TR})$ on a $L' \in \mathrm{CA}_1(\mathrm{TR})$. En effet il suffit d'utiliser le même automate en considérant les deux couches de marquages comme une seule. L' peut donc être vu comme un langage de mots contenant une seule marque en considérant que ces mots sont écrits sur l'alphabet $\Sigma \times \{0,1\}$. Par une application du théorème 5.8, on prouve que $\widetilde{L'} \in \mathrm{CA}_M(\mathrm{TR})$ et donc $\widetilde{L'} \in \mathrm{CA}_1(\mathrm{TR})$. Ce langage est par construction le langage L dans lequel la première marque a été enlevée. Il ne contient donc que des mots qui contiennent au plus k marques. Par hypothèse d'induction, ce langage sans ses marques est donc aussi reconnaissable en temps réel. □

5.4 Entre la dimension deux et la dimension trois

Dans cette section, et cela sera la seule fois dans ce mémoire, nous allons nous intéresser à des automates cellulaires tridimensionnels. Nous avons étudié précédemment les relations entre les langages unidimensionnels reconnaissables avec un automate 1D et avec un automate 2D. Nous avons notamment discuté des conséquences d'une éventuelle égalité de ces deux classes de langages. Nous allons ici montrer que dans le cas des langages bidimensionnels, l'utilisation d'une troisième dimension permet d'accroître strictement la puissance de calcul. La clef de ce résultat est la connaissance de certains langages qui ne sont pas reconnaissables en temps réel avec certains voisinages en deux dimensions. Nous allons modifier le langage L_T présenté dans la section 5.1 de sorte qu'il soit reconnaissable en temps réel par un automate cellulaire en trois dimensions, avec le voisinage de Moore.

Pour cela, nous allons disposer différemment les lettres des mots de L_T. Tout d'abord on se restreint aux mots carrés. Considérons alors un mot de taille $(n \times n)$ et l'ensemble I des cellules de coordonnées paires :

$$I = \{(2x, 2y) \mid 2x \in [\![0, n-1]\!], 2y \in [\![0, n-1]\!]\}$$

L'ensemble I définit un sous mot que l'on peut aisément associer à un mot bidimensionnel en identifiant la cellule $(2x, 2y)$ à la cellule (x, y). On définit alors

le langage L_3 constitué des mots carrés bidimensionnels sur $\{0,1\}^2$ qui vérifient les contraintes suivantes. Le sous-mot constitué par les cellules de I est un mot de L_T sur la première couche. La deuxième couche est intégralement constituée de 0 sauf sur la ligne d'ordonnée $\lfloor \frac{m}{2} \rfloor$ et sur la colonne d'abscisse $\lfloor \frac{n}{2} \rfloor$.

Lemme 5.13. *Le langage L_3 n'est pas reconnaissable en temps réel avec le voisinage de Moore en deux dimensions.*

Démonstration. La démonstration de ce résultat est exactement la même que celle proposée pour le langage L_T dans la section 5.1. Tout d'abord, la restriction aux mots carrés ne pose aucun problème puisque la preuve ne considère que le cas des mots dont la proportion est la même que celle du sommet limitant. Pour le voisinage de Moore il s'agit donc des mots carrés. On remarque ensuite que l'ensemble des cellules dans lesquelles sont codées les coordonnées de la cellule à vérifier n'est plus le même que pour L_T, les cellules de I étant espacées les unes des autres. Cependant cet ensemble est lui aussi inclus dans un rectangle de taille $O(\log(n)^2)$, ce qui permet d'effectuer exactement le même dénombrement que dans la preuve précédente. Enfin, le nombre de cellules que les codes peuvent désigner est plus faible par rapport à la taille du mot que dans le cas de L_T. En effet, seules les cellules de I peuvent être désignées par ces codes. Encore une fois il est facile de vérifier que, même avec cette restriction, les codes peuvent désigner un ensemble de cellules de taille $O(n^2)$, ce qui permet d'utiliser exactement le même argument de comptage pour conclure à l'impossibilité de reconnaître L_3 avec le voisinage de Moore en 2D.

□

On va maintenant montrer que ce langage est reconnaissable avec un automate en trois dimensions.

Théorème 5.14. *Le langage L_3 est reconnaissable en temps réel par un automate tridimensionnel avec le voisinage de Moore.*

Démonstration. En deux dimensions il est impossible de reconnaître L_3 en temps réel car il n'est pas possible de stocker assez de données le long de la diagonale qui sépare l'origine du coin du mot. En trois dimensions il va être possible de résoudre ce problème en empilant les données sur cette diagonale. On définit alors le plan $\mathcal{P}$ des cellules au-dessus et en-dessous de cette diagonale :

$$\mathcal{P} = \{(x, x, z) \mid x \in [\![0, n-1]\!] \, , z \in \mathbb{Z}\}$$

L'objectif de notre construction est alors de stocker toutes les données du mot d'entrée dans ce plan, pour qu'elles soient plus facilement mises en relation avec les données des codes situés dans le coin du mot. Pour cela à chaque étape de calcul, chaque cellule (x, y, z) va dupliquer son état dans la cellule $(x+1, y-1, z+1)$ et dans la cellule $(x-1, y+1, z-1)$. De cette manière les informations contenues dans l'entrée vont petit à petit se propager dans l'espace, sans toutefois se superposer les unes les autres. Ainsi, chaque information contenue dans une cellule de I au

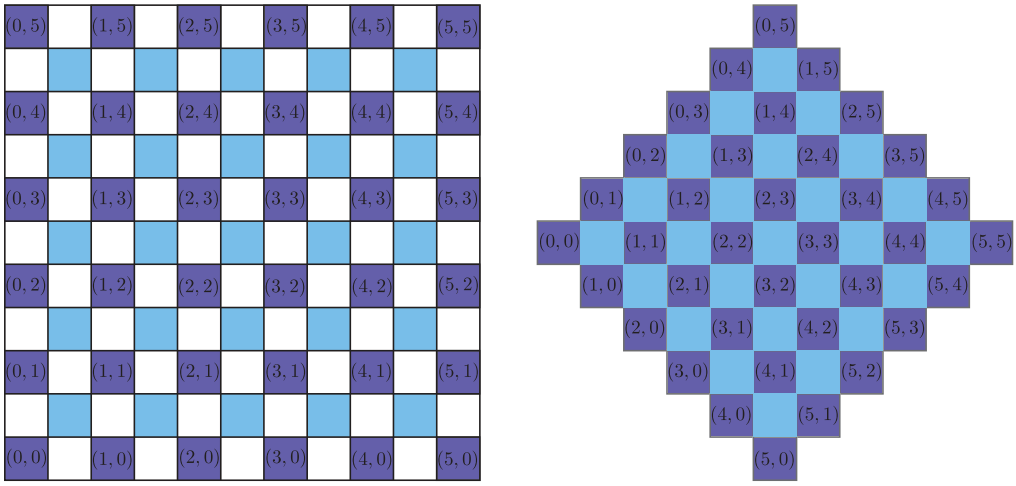

FIGURE 5.8 – À gauche un mot en deux dimensions de taille (11×11). En bleu foncé les cellules de I. À droite une vue du plan $\mathcal{P}$, la où se passe la majeure partie du calcul.

départ sera à une certaine étape inscrite dans une cellule du plan $\mathcal{P}$. On a illustré sur la figure 5.8 les emplacements où les informations se retrouvent au cours de ce déplacement. Les numérotations correspondent aux coordonnées initiales dans le sous mot des cellules de I. Ainsi une cellule numérotée (x, y) contient l'état initial de la cellule $(2x, 2y, 0)$ dans l'automate 3D.

En parallèle, l'automate va agir dans le plan $\mathcal{P}$ pour vérifier que le sous mot donné en entrée est un mot de L_T. Via le déplacement décrit plus haut, les informations des cellules de I vont toutes finir dans un carré situé sur le plan $\mathcal{P}$. On peut alors identifier ce carré à un mot bidimensionnel constitué de lignes et de colonnes (on peut s'aider des numérotations sur la figure 5.8). On peut aussi remarquer que les cellules numérotées (x, y) et $(x+1, y)$ dans le plan $\mathcal{P}$ sont adjacentes dans l'automate 3D. C'est aussi le cas des cellules (x, y) et $(x, y + 1)$. L'automate que nous allons construire va alors utiliser le même algorithme que celui qui a été présenté pour le voisinage de von Neumann en deux dimensions. Tout d'abord l'automate va générer des compteurs à partir de la cellule la plus éloignée pour numéroter les diagonales de $\mathcal{P}$ qui correspondent à des lignes et à des colonnes dans le sous mot initial I. En même temps l'automate va recopier les codes sur chacune de ces lignes et colonnes. Au début du calcul les codes ne sont pas dans le plan $\mathcal{P}$ et ils y arrivent bit par bit. L'automate va alors recopier ces codes dès qu'il en aura la possibilité, bit par bit. Comme en 2D, dès lors que le code commence à être recopié sur la ligne l, un signal est envoyé depuis le bord de cette ligne pour comparer le code au numéro de la ligne en question. Une fois arrivé au milieu de la ligne, le signal sait que la comparaison est terminée, et si elle était concluante, il se transforme en un signal de vérification. Il se passe exactement la même chose pour les colonnes. Enfin lorsque deux signaux

de vérifications se rencontrent ils se transforment en un signal d'acceptation ou de refus en fonction de la valeur de la cellule dans laquelle ils se sont rencontrés.

La dernière cellule de la ligne l reçoit l'information de l'entrée 2D dès le temps $(n-l)$, qui correspond exactement au moment ou le signal de comparaison va partir. De même, le j-ème bit des codes va arriver sur le plan $\mathcal{P}$ au temps $(j+1)$, et sera donc recopié sur la ligne l au temps $j+l+1$, ce qui encore une fois correspond exactement au moment où il devra être comparé avec le numéro de la ligne. Cet algorithme s'effectue donc au plus en temps n, temps qu'il faut pour parcourir $\frac{n}{2}$ lignes et $\frac{n}{2}$ colonnes.

Enfin il faut aussi vérifier en parallèle que les marques sur la deuxième couche soient bien mises sur la ligne du milieu et la colonne du milieu du mot. Pour cela chaque cellule vérifie localement qu'elle est au cœur d'une ligne ou d'une colonne marquée, si ce n'est pas le cas elle se met à propager un signal de refus. En même temps on envoie un signal depuis l'origine vers le coin opposé du mot, et un signal depuis ce coin vers l'origine pour trouver le milieu du mot. Ce deux signaux doivent croiser pour la première fois une cellule marquée au moment où ils se rencontrent. Si ce n'est pas le cas ils propagent un signal de refus.

L'automate ainsi construit reconnaît le langage L_3 en temps réel.

□

Le langage L_3 est donc un langage reconnaissable par un automate cellulaire en 3D avec le voisinages de Moore mais pas reconnaissable en 2D avec le même voisinage.

L'étude des relations entre ces classes, comme souvent lors de l'étude de classes de complexité, reste assez incomplète bien que nous ayons obtenu des résultats encourageants. La relation entre le temps réel et le temps linéaire reste cependant inconnue pour de nombreux voisinages en deux dimensions notamment pour le voisinage usuel de von Neumann. La méthode présentée ici pour prouver qu'un langage n'est pas reconnaissable en temps réel ne semble pas pouvoir fonctionner avec tous les voisinages. Cependant, cette preuve est loin d'exploiter toutes les contraintes liées aux automates cellulaires. Par exemple cette preuve serait valable pour des automates non homogènes, c'est-à-dire qu'on n'a nulle part utilisé le fait que chaque cellule effectuait le même calcul. Les seules contraintes exploitées sont la localité des informations ainsi que le fait que le nombre d'états est fini et donc qu'une quantité finie d'information peut être stockée localement. L'homogénéité est probablement la clé de résultats plus généraux mais jusqu'à présent très peu de preuves ont été capables d'exploiter cette caractéristique des automates cellulaires. Les relations entre dimensions ont été assez peu étudiées dans la littérature, du moins avec le point de vu adopté au cours de cette thèse. C'est pourquoi les résultats présentés restent dans un cadre assez spécifique. Le résultat de différence entre les dimensions deux et trois étant facilement généralisable aux dimensions supérieures, la question est presque entièrement fermée pour le voisinage de Moore. Il est en effet assez facile

de se convaincre que cette preuve peut se généraliser aux voisinages avec un sommet limitant. En fait le résultat est fortement lié à la connaissance d'un exemple de langage simple mais non reconnaissable en temps réel. C'est cette lacune dans la compréhension des limites du temps réel en une dimension qui nous empêche de conclure à propos de la relation entre les dimensions un et deux. Bien qu'imparfaits, les résultats de ce chapitre poussent à croire à la différence entre ces deux classes, la stabilité amenée par leur égalité semblant trop forte pour une classe aussi restreinte que le temps réel.

Chapitre 6

Relations entre voisinages

Après avoir étudié les relations entre le temps réel et le temps linéaire, il est naturel d'étudier les relations entre les différentes classes de temps réel. Comme le suggère le titre de cette thèse, cette étude a été menée en deux dimensions. On peut néanmoins remarquer que certains résultats, notamment ceux de différences entre voisinages, sont adaptables à des dimensions supérieures. En une dimension, il a déjà été montré que tous les voisinages complets étaient équivalent vis-à-vis du temps réel [Pou05]. En deux dimensions, il est possible de montrer que les langages reconnaissables avec un voisinage complet dépendent fortement de l'enveloppe convexe du voisinage. On cherche donc à classifier les voisinages en fonction de leur enveloppe convexe.

6.1 Différence en temps réel des voisinages avec un sommet limitant

La première section de ce chapitre porte encore une fois sur une adaptation d'une construction de V. Terrier qui montre qu'aucun voisinage ayant un sommet limitant ne définit la même classe de temps réel que le voisinage de von Neumann [Ter04]. La clé de ce résultat est le langage L_T, déjà exposé dans le chapitre 5, reconnaissable avec le voisinage de von Neumann mais avec aucun voisinage présentant un sommet limitant. La construction présentée dans ce mémoire permet de construire des langages qui sont reconnaissables avec un voisinage sans sommet dans une certaine direction mais pas avec un voisinage qui possède un tel sommet. Formellement, on prouve le théorème suivant :

Théorème 6.1. *Étant donnés deux voisinages $\mathcal{V}$ et $\mathcal{V}'$, si $\mathcal{V}$ admet un sommet limitant (x, y) de proportion $\frac{x}{y} = \alpha$ et que $\mathcal{V}'$ n'a aucun sommet (a, b) de proportion $\frac{a}{b} = \alpha$, alors il existe un langage L que l'on peut reconnaître en temps réel avec un automate avec le voisinage $\mathcal{V}'$ mais avec aucun automate avec le voisinage $\mathcal{V}$.*

Démonstration. Comme le laisse supposer la condition de « sommet limitant », le langage que nous allons proposer est fortement inspiré du langage L_T présenté dans le

chapitre précédent. Les modifications que nous allons apporter permettent de rendre ce langage reconnaissable avec $\mathcal{V}'$. Le point principal est de se restreindre aux mots de proportion α. Ce sont les mots de cette proportion qui sont considérés dans la preuve du théorème 5.3 et uniquement ceux-là. Puisque le voisinage $\mathcal{V}'$ n'a aucun sommet de proportion α, il va être impossible d'adapter la preuve à ce voisinage. On peut alors vérifier que les modifications du langage L_T n'affectent pas le principe du raisonnement de la preuve du théorème 5.3 pour le voisinage $\mathcal{V}$. En effet, bien que déformé le code sera toujours situé dans un rectangle en haut à droite de taille logarithmique. De plus la quantité de coordonnées encodables sera aussi réduite, mais restera linéaire en la taille de l'entrée. Le même argument de dénombrement est alors utilisable pour démontrer qu'aucun automate avec le voisinage $\mathcal{V}$ ne peut reconnaître ce langage en temps réel.

On définit v_1 comme étant le sommet de $\mathcal{V}'$ de plus petite proportion supérieure à α dans le quart de plan positif. Si aucun tel sommet n'existe, v_1 est le point le plus haut de $\mathcal{V}'$ sur l'axe des ordonnées. De même, on définit v_2 le sommet de $\mathcal{V}'$ de plus grande proportion inférieure à α, si aucun tel sommet n'existe v_2 est le point le plus à droite de $\mathcal{V}'$ sur l'axe des abscisses.

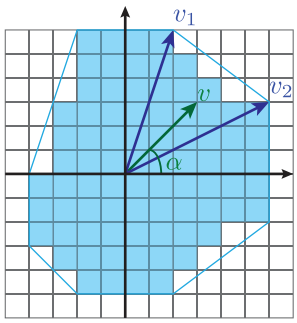

FIGURE 6.1 – Les sommets v_1 et v_2 pour un voisinage quelconque avec $\alpha = 1$.

Considérons maintenant $D = \{av_1 + bv_2 \mid a, b \in \mathbb{N}\}$ le cône induit par les vecteurs v_1 et v_2. Étant donnée une cellule c, on note $D_c = \{c - av_1 - bv_2 \mid a, b \in \mathbb{N}\}$ le sous ensemble de $\mathbb{Z}^2$ induit par les vecteurs $-v_1$ et $-v_2$ avec une origine en c. $D_c \cap D$ est l'ensemble de toutes les cellules qui se trouvent sur un chemin optimal entre l'origine et c en utilisant seulement les vecteurs v_1 et v_2. C'est cette nouvelle grille que nous allons utiliser comme espace de travail. Cet espace avantage le voisinage $\mathcal{V}'$ qui peut aisément faire transiter les informations d'une cellule à l'autre. En fait, le comportement de $\mathcal{V}'$ vis-à-vis de cette grille va ressembler fortement à celui du voisinage de von Neumann vis-à-vis de $\mathbb{Z}^2$. Sur cette grille, nous allons inscrire des coordonnées, en binaire, dans le coin en haut à droite. Le langage L que nous allons construire est donc écrit sur l'alphabet $\{0, 1\}^4$ et contient les mots de taille (n, m) qui vérifient les contraintes suivantes :

— Une seule cellule c a un 1 dans sa première composante ;

— La cellule c appartient à D ;
— Les cellules ayant un 1 dans leur deuxième composante sont exactement les cellules de D_c ;
— Les cellules qui ont un 1 dans la troisième composante forment deux segments qui partent de c et qui sont le long des bords de D_c. Les extrémités de ces segments ont un 1 dans leur quatrième composante ;
— $\frac{n}{m} = \alpha$;
— La quatrième composante de la cellule $c - xv_1 - yv_2$ où x et y sont les coordonnées encodées par les deux segments précédents est un 1 ;

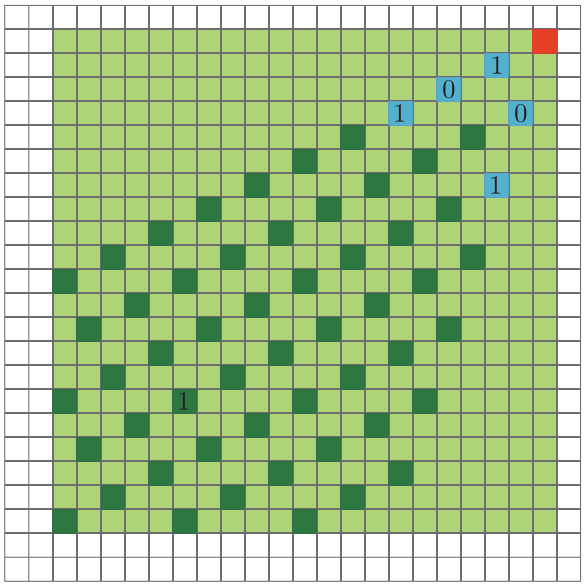

FIGURE 6.2 – Exemple d'un mot de L. En vert clair les cellules du mot, en foncé les cellules de la grille de calcul, et en bleu les cellules qui constituent le code des coordonnées.

L'automate que nous allons construire va vérifier ces divers points en parallèle via plusieurs couches de calcul.

Vérifier le premier point est assez facile. Chaque cellule ayant un 1 dans sa première composante va envoyer un signal à vitesse maximale vers l'origine avec une pente α. Si ce signal atteint un bord, il se déplace ensuite le long de ce bord à vitesse maximale jusqu'à l'origine. Si deux signaux de cette nature se rencontrent, ils envoient un signal de refus vers l'origine. Si l'origine reçoit un signal venant d'une telle cellule et aucun signal de refus, la première condition est validée. Comme on ne s'intéresse ici qu'à des entrées de proportion α, on peut vérifier que ces signaux arrivent forcément avant le temps réel. Pour cela deux cas se présentent à nous. On note (α_h, α_v) le vecteur maximal de pente α dans $\mathcal{V}'$. On utilise à nouveau $(v_h, 0)$ pour

noter le vecteur horizontal maximal de $\mathcal{V}'$ et $(0, v_v)$ le plus grand vecteur vertical. Sans perdre de généralité, on va supposer que la cellule $c = (x, y)$ qui envoie le signal est située au dessus de la droite de pente α qui part de l'origine. Ceci signifie que le signal va arriver sur le bord gauche de l'image. Si $v_v \geq \alpha_v$ alors le signal arrive à l'origine avant le temps $\lceil \frac{x}{\alpha_v} \rceil$ et donc avant $\lceil \frac{m}{\alpha_v} \rceil$ qui est le temps minimal pour que la cellule (n, m) puisse communiquer avec l'origine, et donc inférieur au temps réel. Sinon, on peut vérifier de la même manière que le signal arrive avant l'information venant de la cellule $(0, m)$.

Le deuxième point est aussi très facile à vérifier. La cellule marquée va, sur une deuxième couche de calcul, envoyer un signal qui va se dupliquer à chaque étape de calcul se propageant selon v_1 et v_2. Ainsi le signal va se propager sur toutes les cellules de la grille engendrée par $-v_1$ et $-v_2$ ancrée en c. Si c est dans l'ensemble D, alors ce signal va arriver à l'origine. Sinon, l'origine et c ne sont pas sur les mêmes grilles et le signal ne va jamais l'atteindre.

Pour vérifier le troisième point, chaque cellule k marquée sur la troisième couche va vérifier que les cellules $(k+v_1)$ et $(k+v_2)$ sont aussi marquées. Si l'une d'elles n'est pas marquée, la cellule k se marque elle même sur une troisième couche de calcul comme étant sur un bord (haut ou droit, dépendant de la cellule non marquée détectée). Si aucune n'est marquée et que la cellule k n'est pas marquée sur la première composante, elle envoie dans toutes les directions un signal de refus. À l'étape suivante, chaque cellule sur un bord vérifie que la suivante est sur le bord aussi. Chaque cellule k qui n'est pas sur le bord haut vérifie que la cellule $(k + v_1)$ n'est pas non plus sur un bord haut. De même pour les bords droits mais avec la cellule $(k+v_2)$. Si ces conditions ne sont pas respectées, la cellule envoie dans toutes les directions un signal de refus. Enfin chaque cellule k qui n'est pas marquée vérifie que les cellules $(k + v_1)$ et $(k + v_2)$ ne le sont pas non plus. Si l'origine ne reçoit aucun signal de refus sur cette couche de calcul, la grille était correcte.

Une fois que les cellules du bord se sont reconnues, il est facile de tester la quatrième condition. Chaque cellule marquée doit être sur un bord et la cellule qui la précède dans le code doit aussi être marquée.

Le cinquième point se vérifie à l'aide d'un signal qui part de la cellule (n, m) vers l'origine avec une pente α. Ce signal va arriver sur l'origine (ou à une distance bornée due à l'approximation de la pente α avec le voisinage $\mathcal{V}'$). Si ce n'est pas le cas il se transforme lui aussi en signal de refus. Si aucun signal de ce type n'arrive à l'origine avant le temps réel, le mot est aussi refusé.

Enfin, il faut reconnaître le langage à proprement parler. Pour cela, nous allons utiliser la même construction que dans la section précédente avec le voisinage de von Neumann. En effet, l'algorithme proposé est tout à fait réalisable en utilisant seulement les vecteurs $(1, 0)$ et $(0, 1)$ du voisinage. À la place de ces vecteurs nous allons utiliser les vecteurs v_1 et v_2 et ignorer simplement le reste du voisinage. Comme pour le cas du voisinage de von Neumann, chaque cellule est capable de détecter en une étape qu'elle est sur le bord. Il est donc facile de démarrer le compteur binaire et de recopier les codes qui sont eux aussi marqués dès le début. Il reste maintenant

à vérifier que cet algorithme nécessite moins de temps de calcul que le temps réel pour le voisinage $\mathcal{V}'$. Tout d'abord, on remarque que cet algorithme s'effectue en temps réel pour n'importe quel voisinage convexe N qui contient uniquement v_1 et v_2 comme sommets dans le quart nord est. Comme il n'y a aucun sommet de l'enveloppe convexe de $\mathcal{V}'$ entre v_1 et v_2, la forme de $\mathcal{V}'$ dans le cône en question est exactement la même que celle de N. Par conséquent, la cellule (n, m) qui est dans ce cône de direction est à même distance dans $\mathcal{V}'$ et dans N, or c'est la cellule la plus éloigné pour un N bien choisi. Ainsi le temps réel de $\mathcal{V}'$ est supérieur à celui de ce N, et donc supérieur au temps que nécessite l'algorithme pour s'exécuter correctement.

Nous avons donc prouvé qu'il est possible de reconnaître L en temps réel avec un automate cellulaire avec le voisinage $\mathcal{V}'$.

Le deuxième partie de cette preuve consiste à montrer que ce langage vérifie les conditions nécessaire à ce que la preuve de la section 5.1 fonctionne, c'est à dire que le code est bien dans un rectangle suffisamment petit en haut à droite, et que le nombre de coordonnées encodables est suffisamment grand. On ne va s'intéresser qu'aux cas où la cellule c (en rouge sur la figure 6.2) est proche du coin nord est du mot. En effet, dans les mots de proportion α, il existe toujours un point de D_O à moins de $\|v_1\| + \|v_2\|$ cases de la cellule (n, m). La taille des codes est de l'ordre de $log(n)$ pour les deux codes. Donc l'ensemble des cellules qui codent les coordonnées est inclus dans un carré $\|v_1\|(log(n) + 2) \times \|v_2\|((log(m) + 2)$. Enfin, il faut vérifier que les codes peuvent désigner assez de cellules différentes. Puisque v_1 et v_2 ne sont pas colinéaires, le nombre de cellules sur la grille de calcul est proportionnel à la taille du mot et donc est de l'ordre de nm. Nous pouvons donc appliquer le théorème pour justifier qu'il est impossible de reconnaître L avec le voisinage $\mathcal{V}$. □

Corollaire 6.2. *Les classes de temps réel de deux voisinages rectangles de forme différente sont incomparables.*

6.2 Equivalence des losanges

Le théorème 5.3 donne un statut particulier aux voisinages qui n'ont pas de sommet limitant. En effet ces sommets qui, à la base, expriment la possibilité pour l'information de se déplacer efficacement dans une direction donnée sont en fait des faiblesses. C'est leur présence au sein de ces voisinages qui nous permet de prouver que certains langages reconnaissables en temps linéaire ne le sont pas en temps réel. L'étape suivante est sans surprise de s'intéresser aux voisinages sans sommet limitant. La première étape de cette étude concerne les voisinages qui n'ont aucun sommet de leur enveloppe convexe de coordonnées strictement positives. Ces voisinages ont déjà été présentés dans ce manuscrit et sont appelés voisinages losanges à cause de la forme de leur enveloppe convexe. Tout comme les rectangles sont une généralisation du voisinage usuel de Moore, ces voisinages sont une généralisation du voisinage de von Neumann. Contrairement aux rectangles, dont on peut démontrer que les classes de temps réel sont deux à deux incomparables, on démontre ici que

les losanges définissent tous la même classe de complexité. Ce résultat a été présenté à Automata 2016 [Gra16].

Théorème 6.3. *Étant donnés deux voisinages losanges $\mathcal{V}$ et $\mathcal{V}'$, si L est reconnu en temps réel par un automate $\mathcal{A}$ avec le voisinage $\mathcal{V}$, alors il existe un automate $\mathcal{A}'$ avec le voisinage $\mathcal{V}'$ reconnaissant L en temps réel.*

Démonstration. Soient deux voisinages losanges $\mathcal{V}_{a,b}$ et $\mathcal{V}_{c,d}$ avec les notations de la définition 4.1. Soit L un langage reconnu en temps réel par un automate $\mathcal{A}$ avec le voisinage $\mathcal{V}_{a,b}$. Nous allons construire un automate avec le voisinage $\mathcal{V}_{c,d}$ qui reconnaît L en temps réel. Cette construction va se faire en trois étapes, décrites par les lemmes suivants :

Lemme 6.4. *Étant donnés trois entiers a, b et k, si L est reconnu en temps réel par un automate $\mathcal{A}$ avec le voisinage $\mathcal{V}_{a,b}$, alors il existe un automate $\mathcal{A}'$ avec le voisinage $\mathcal{V}^k_{a,b} = \mathcal{V}_{ka,kb}$ reconnaissant L en temps réel.*

Lemme 6.5. *Étant donnés trois entiers a, b et k, si L est reconnu en temps réel par un automate $\mathcal{A}$ avec le voisinage $\mathcal{V}_{ka,b}$, alors il existe un automate $\mathcal{A}'$ avec le voisinage $\mathcal{V}_{a,b}$ reconnaissant L en temps réel.*

Lemme 6.6. *Étant donnés trois entiers a, b et k, si L est reconnu en temps réel par un automate $\mathcal{A}$ avec le voisinage $\mathcal{V}_{a,kb}$, alors il existe un automate $\mathcal{A}'$ avec le voisinage $\mathcal{V}_{a,b}$ reconnaissant L en temps réel.*

Le lemme 6.4 nous donne un automate avec le voisinage $\mathcal{V}^{cd}_{a,b} = \mathcal{V}_{cda,cdb}$ qui reconnaît L en temps réel. Grâce ensuite au lemme 6.5 nous en obtenons un qui utilise le voisinage $\mathcal{V}_{c,cdb}$, et enfin grâce au lemme 6.6 nous avons l'automate $\mathcal{A}'$ voulu, qui utilise le voisinage $\mathcal{V}_{c,d}$ et qui reconnaît L en temps réel.

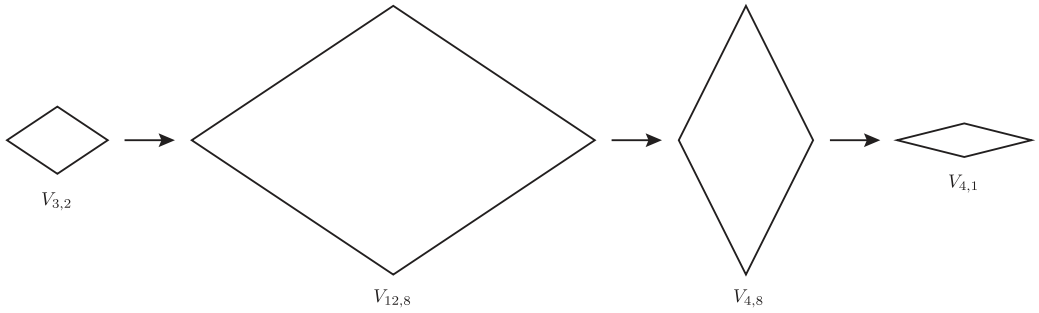

FIGURE 6.3 – Les différentes étapes pour passer du voisinage $\mathcal{V}_{3,2}$ au voisinage $\mathcal{V}_{4,1}$.

Pour aider la compréhension tout au long de la preuve, nous utiliserons la notation $\mathrm{TR}_{a,b}$ pour désigner la fonction de temps réel qui correspond au voisinage $\mathcal{V}_{a,b}$, quel que soit a et quel que soit b.

Nous pouvons maintenant prouver le premier lemme. Il suffit de remarquer que $\mathrm{TR}_{ka,kb} = \left\lceil \frac{\mathrm{TR}_{a,b}}{k} \right\rceil$. De plus, un automate $\mathcal{A}_k$ avec le voisinage $\mathcal{V}_{ka,kb}$ peut aisément

simuler k étapes d'un automate $\mathcal{A}$ avec le voisinage $\mathcal{V}_{a,b}$ à chaque étape de son calcul. Ainsi l'automate $\mathcal{A}_k$ aura effectué plus de $\mathrm{TR}_{a,b}$ étapes simulées de l'automate $\mathcal{A}$ en temps $\mathrm{TR}_{ka,kb}$.

Les deux autres lemmes sont très similaires, aussi nous ne prouverons dans ce manuscrit que le lemme 6.5, la preuve du lemme 6.6 étant symétrique. Soient trois entiers a, b, k, et un langage L reconnu en temps réel par l'automate $\mathcal{A}$ avec le voisinage $\mathcal{V}_{ka,b}$. Nous allons construire un automate $\mathcal{A}'$ avec le voisinage $\mathcal{V}_{a,b}$ qui va simuler $\mathcal{A}$ en temps réel grâce à une compression de l'entrée. Cette fois ci, nous n'allons compresser les entrées que dans une dimension ce qui va artificiellement « étirer » le voisinage dans la dimension voulue.

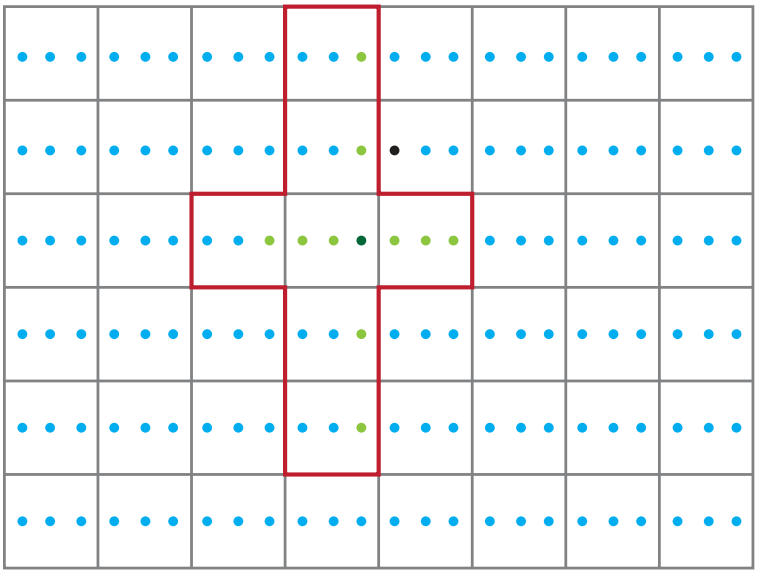

FIGURE 6.4 – En rouge le voisinage $\mathcal{V}$ d'une cellule dans $\mathcal{A}'$, en vert foncé une cellule c de $\mathcal{A}$, et en vert clair le voisinage $\mathcal{V}'$ de cette cellule dans $\mathcal{A}$. En noir une cellule voisine de c dans $\mathcal{A}$ mais inaccessible dans $\mathcal{A}'$.

Sur une entrée de taille $n \times m$, chaque ligne va indépendamment compresser son entrée d'un facteur k. Cette compression nécessite $\lceil \frac{n(k-1)}{ka} \rceil$ étapes de calcul de l'automate. Après cette compression chaque cellule dispose de plus d'informations qu'au départ pour faire évoluer les k états de l'automate $\mathcal{A}$ qu'elle contient. En effet, pour chaque « cellule » de $\mathcal{A}$ comprise dans une cellule de $\mathcal{A}'$, cette dernière voit les ka cellules à sa gauche et à sa droite ainsi que les b cellules au dessus et en dessous (voir figure 6.4). Cependant, comme illustré sur la figure 6.4, la cellule ne dispose pas d'assez d'informations pour simuler directement le comportement de l'automate $\mathcal{A}$. Notons N le voisinage constitué des cellules de $\mathcal{V}_{ka,b}$ dont une des coordonnées est nulle. Alors N a la même enveloppe convexe que $\mathcal{V}_{ka,b}$. De plus N est complet donc, grâce à un théorème de M. Delacourt et V. Poupet dans [DP07], on sait que

N peut reconnaître L en $\mathrm{TR}_{ka,b}+t$. Une fois la compression effectuée, l'automate $\mathcal{A}'$ peut donc reconnaître L en temps $\mathrm{TR}_{ka,b}+t$. Il nous faut maintenant nous intéresser de plus près à l'expression de ces temps réels. Nous avons :

$$\mathrm{TR}_{a,b}(n,m) = \left\lceil \frac{n}{a} + \frac{m}{b} \right\rceil \geq \frac{n}{a} + \frac{m}{b}$$

$$\mathrm{TR}_{ka,b}(n,m) = \left\lceil \frac{n}{ka} + \frac{m}{b} \right\rceil \leq \frac{n}{ka} + \frac{m}{b} + 1$$

Par notre lemme 3.5, nous savons que le calcul de l'automate $\mathcal{A}'$ sera terminé au plus en temps $\left\lceil \frac{n(k-1)}{ka} \right\rceil + \mathrm{TR}_{ka,b}+t \leq \frac{n}{a} + \frac{m}{b} + t + 2 \leq \mathrm{TR}_{ka,b}(n,m) + t + 2$.

Grâce maintenant au théorème 4.2 d'accélération constante, nous savons qu'il existe un automate $\mathcal{A}''$ avec le voisinage $\mathcal{V}_{ka,b}$ qui reconnaît L en temps réel.

□

6.3 La supériorité du voisinage de von Neumann

Dans ce manuscrit nous avons mis en évidence le fait que de nombreux voisinages ne disposaient pas de la même puissance de calcul en temps réel et en temps linéaire. Nous avons aussi montré qu'en temps réel ces voisinages offraient une puissance de calcul incomparable avec celle d'autres voisinages similaires. Cependant, le voisinage de von Neumann semble toujours résister à nos constructions. En effet, nous ne savons toujours pas si la puissance de calcul qu'il offre diffère en temps réel et en temps linéaire. S'il semble ne pas avoir de faiblesse, nous n'avons pas non plus réussi à montrer qu'il était supérieur aux autres voisinages. Le théorème précédent, concernant les voisinages losanges, ressemblant à celui de von Neumann, va nous permettre de montrer cette supériorité vis-à-vis d'une certaine classe de voisinages encore une fois définie à partir de la forme de l'enveloppe convexe :

Définition 6.7. *On dit qu'un voisinage $\mathcal{V}$ est* rond *s'il est complet, symétrique, qu'il possède un sommet sur chaque axe et que toute paire de sommets du quart nord est de l'enveloppe convexe (a,b) et (c,d) vérifie $c > a \Leftrightarrow b > d$.*

Moins formellement, il s'agit des voisinages symétriques dont le quart nord est de l'enveloppe convexe est composé uniquement de segments de pentes négatives sans segment horizontaux ni verticaux. C'est la première fois dans ce manuscrit que de tels voisinages sont présentés. En effet, ils ne correspondent à aucun voisinage usuel, ni même à une déformation simple de ceux-ci. En fait, ce sont les voisinages dont l'enveloppe convexe est comprise entre un losange et un rectangle, une fois une mise à l'échelle adéquate effectuée. Le voisinage de von Neumann correspond au cas particulier sans sommet positif et le voisinage de Moore est une limite inatteignable mais dont on peut se rapprocher arbitrairement avec des pentes de plus en plus proches de l'horizontale et de la verticale. Cette idée est illustrée sur la figure 6.5.

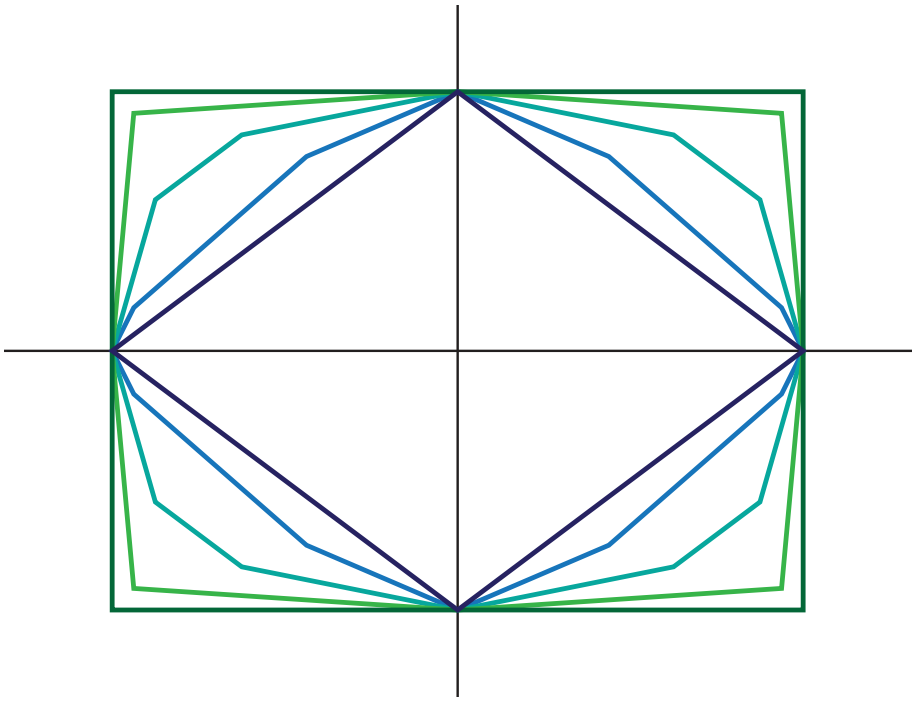

FIGURE 6.5 – Représentation de plusieurs enveloppes convexes de voisinages ronds, mises à l'échelle entre les voisinages losanges et rectangles correspondant.

Théorème 6.8. *Le voisinage de von Neumann permet de reconnaître strictement plus de langages en temps réel que n'importe quel voisinage rond.*

Démonstration. L'idée de cette preuve est de réussir à simuler n'importe quel voisinage rond $\mathcal{V}$ avec celui de von Neumann sans perdre de temps. Pour cela, nous allons utiliser le théorème 6.3 et simuler plusieurs voisinages qui sont équivalents à celui de von Neumann. En effet, on peut remarquer qu'à chaque segment du quart nord est de $\mathrm{CH}(\mathcal{V})$, on peut faire correspondre un losange qui coïncide avec ce segment et qui contient entièrement $\mathcal{V}$. Ce losange ne correspond pas parfaitement à une enveloppe convexe de voisinage car ses sommets ne sont pas forcément à coordonnées entières. Ces coordonnées sont, par contre, rationnelles. Il est donc possible, en considérant une puissance de $\mathcal{V}$, de s'assurer que tous ces losanges correspondent effectivement à des voisinages.

Considérons un voisinage rond $\mathcal{V}$. On note les sommets du quart nord est de son enveloppe convexe $s_1, s_2, ..., s_k$ en partant du plus à gauche vers le plus à droite. Deux sommets consécutifs $s_i = (x_i, y_i)$ et $s_{i+1} = (x_{i+1}, y_{i+1})$ définissent une enveloppe convexe symétrique dont les deux sommets dans le quart nord est sont $(0, y_i - x_i \frac{y_{i+1}-y_i}{x_{i+1}-x_i})$ et $(x_i - y_i \frac{x_{i+1}-x_i}{y_{i+1}-y_i}, 0)$. Puisque ces deux sommets ont des coordonnées rationnelles, et qu'il y a un nombre fixe $(k-1)$ de telles enveloppes, il est possible de considérer une puissance du voisinage $\mathcal{V}$ pour laquelle tous ces sommets ont des coordonnées entières. En effet, la même construction sur le voisinage $\mathcal{V}^p$ nous

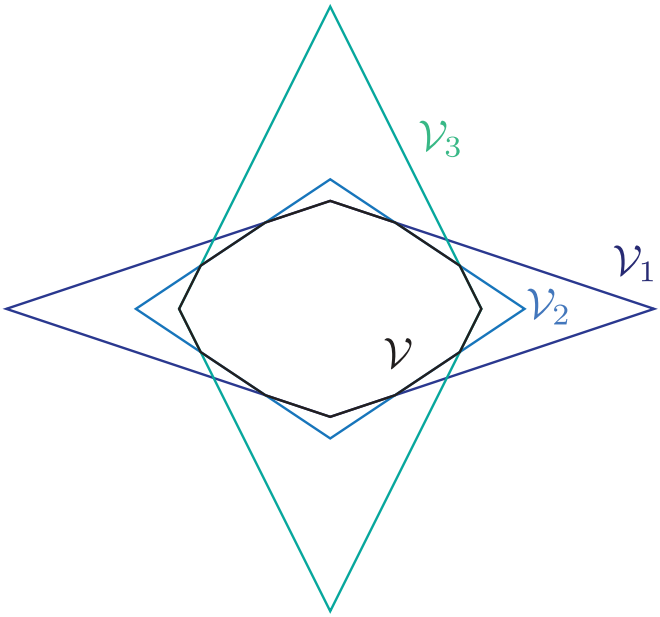

FIGURE 6.6 – Un voisinage rond et les voisinages losanges associés à chacun de ses segments (seules les enveloppes convexes sont représentées)

donne les mêmes enveloppes mises à la puissance p elles aussi.

De plus, on peut reconnaître tout langage reconnaissable avec $\mathcal{V}$ en temps réel, en temps réel avec $\mathcal{V}^p$. Simuler les automates utilisant le voisinage $\mathcal{V}^p$ suffit donc à conclure la preuve. Pour simplifier les notations, nous noterons désormais $\mathcal{V}$ le voisinage $\mathcal{V}^p$. On note aussi $\mathcal{V}_i$ le voisinage induit par les sommets s_i et s_{i+1} de $\mathcal{V}$.

Il ne reste maintenant plus que deux remarques à faire pour conclure la preuve. Tout d'abord on remarque aisément que tout voisinage $\mathcal{V}_i$ ainsi défini contient le voisinage $\mathcal{V}$. Par conséquent un automate avec le voisinage $\mathcal{V}_i$ peut simuler un automate avec $\mathcal{V}$ sans perdre de temps. Ensuite, un rapide calcul montre que lorsque la proportion de l'entrée est comprise entre les proportions de s_i et s_{i+1}, le temps réel est le même pour les voisinages $\mathcal{V}$ et $\mathcal{V}_i$. Cela est dû au fait que dans les voisinages rond, chaque point de l'enveloppe convexe est « limitant ». Étant donné un langage L, reconnu en temps réel avec le voisinage $\mathcal{V}$, il peut être reconnu en temps réel par un automate avec le voisinage $\mathcal{V}_i$ par simple simulation en se restreignant aux entrées de bonne proportion. Grâce au théorème d'équivalence précédemment démontré, il existe un automate $\mathcal{A}_i$ avec le voisinage de von Neumann qui est capable de reconnaître en temps réel ce même langage avec cette même restriction. Par définition d'un voisinage rond, on sait que s_1 est sur l'axe des ordonnées et que s_k est sur l'axe des abscisses. Par conséquent, tout mot d'entrée a une proportion comprise entre celles de deux s_i consécutifs. Le langage L est donc exactement l'union des langages restreints reconnus par les automates $\mathcal{A}_i$. En définissant $\mathcal{A}$ comme l'automate qui

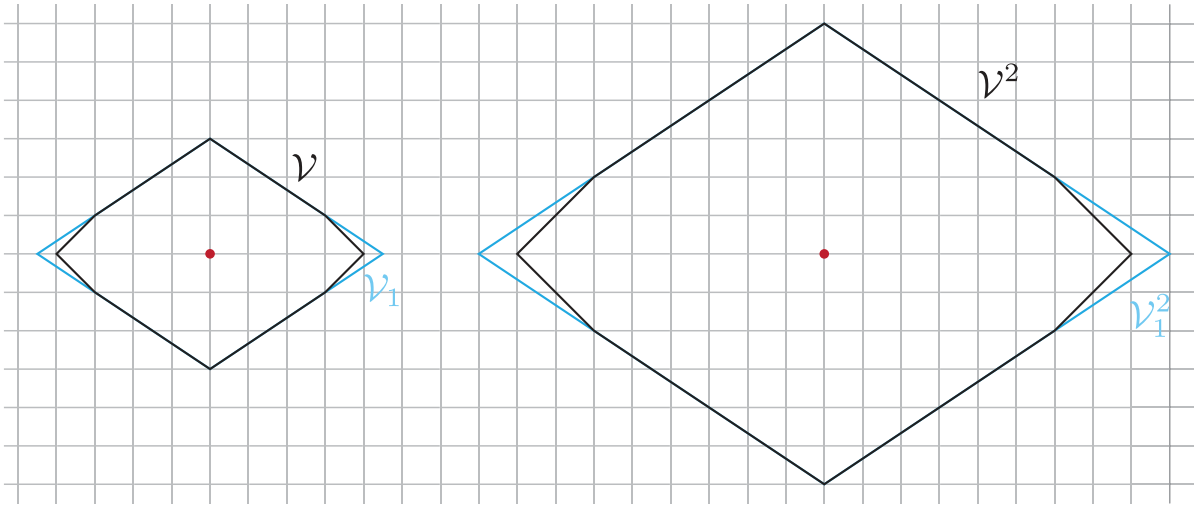

FIGURE 6.7 – Un voisinage pour lequel $\mathcal{V}_1$ n'est en fait pas un voisinage, le sommet sur l'axe des abscisses n'est pas un point de coordonnées entières. En passant au voisinage $\mathcal{V}^2$ on résout le problème.

simule en parallèle tous les automates $\mathcal{A}_i$ et qui accepte ou refuse le mot en fonction du premier résultat qu'il obtient, on obtient alors un automate avec le voisinage de von Neumann qui reconnaît le langage L en temps réel. □

Un des objectifs à long terme des recherches effectuées au cours cette thèse était la classification des voisinages bidimensionnels en fonction des langages qu'ils permettent de reconnaître en temps réel. Si cette classification n'est toujours pas complète, la compréhension que nous en avons a grandement avancé. En effet, nous sommes aujourd'hui en mesure d'exhiber un ensemble infini de voisinages dont les classes de temps réel sont incomparables deux à deux, ainsi qu'un ensemble infini de voisinages définissant la même classe. De plus, nous sommes en mesure de prouver la supériorité d'une classe de voisinage par rapport à une autre uniquement en fonction de la forme de leur enveloppe convexe, et notamment dans le quart de plan positif. Ces résultats permettent une classification déjà assez complète des voisinages symétriques. Les voisinages qui restent bloquants sont ceux dont l'enveloppe convexe a un sommet positif mais non limitant. On pense notamment au plus petit voisinage complet présenté au début de ce mémoire dont le seul sommet positif n'est pas limitant. Il serait aussi très intéressant d'être capable d'exhiber un voisinage qui serait capable de reconnaître un langage non reconnaissable avec le voisinage de von Neumann. En effet, tous les résultats démontrés dans ce manuscrit tendent à faire penser que ce voisinage est plus puissant que tous les autres. Trouver un tel voisinage nécessite l'exhibition d'un langage reconnaissable en temps linéaire mais pas en temps réel avec le voisinage de von Neumann, ce qui nous ramène à la question posée dans le chapitre précédent, qui est sans aucun doute la question principale à résoudre pour parfaire cette compréhension du temps réel en deux dimensions.

Chapitre 7

Temps sous réel

La plus grosse partie de ce mémoire est dédiée à l'étude du temps réel, comme l'indique le titre de cette thèse. Outre les intérêts déjà discutés de cette notion, un résultat connu motive aussi l'utilisation d'une telle mesure de temps : un langage en une dimension reconnu en moins de temps que le temps réel (pour au moins une taille n) est en fait reconnaissable en temps constant. En effet, un automate qui s'arrête avant d'avoir lu toute l'entrée sur tout mot de taille n va s'arrêter en exactement le même temps pour toutes les entrées de taille supérieure à n. Ce résultat n'est plus vrai en deux dimensions, mais nous allons présenter d'autres quantités de temps, plus petites que le temps réel, pour lesquelles un théorème similaire va tenir. Cependant, ces classes de temps sont très restreintes et par conséquent très instables. C'est cette instabilité qui nous motive à utiliser quand même le temps réel comme notion phare pour mesurer la complexité d'un langage.

Définition 7.1 (Temps sous réel)**.** *Étant donné un voisinage $\mathcal{V}$, on définit la fonction de temps sous-réel associé à $\mathcal{V}$ comme le temps minimal pour que l'origine voit un des bords opposés du mot. Formellement :*

$$\mathrm{TSR}_{\mathcal{V}}(n,m) = \min\{\ t \mid \exists (x,y) : x > n \ ou\ y > m \ et\ (x,y) \in \mathcal{V}^t\}$$

On note $\mathrm{SR}_{\mathcal{V}}$ la classe des langages reconnaissables en temps sous réel avec un automate utilisant le voisinage $\mathcal{V}$.

Le premier résultat que nous allons prouver est le suivant :

Proposition 7.2. *Étant donné un langage L reconnaissable avec un voisinage $\mathcal{V}$ en temps $f \neq \mathrm{TSR}_{\mathcal{V}}$, si pour tout couple (n,m), $f(n,m) < \mathrm{TSR}_{\mathcal{V}}(n,m)$, alors L est reconnaissable en temps constant.*

Démonstration. Considérons L un langage qui correspond aux hypothèses du théorème et $\mathcal{A}$ l'automate capable de le reconnaître en temps f. Comme f est bornée par le temps sous réel, mais différente, il existe une taille d'entrée (n_0, m_0) telle que $f(n_0, m_0) < \mathrm{TSR}_{\mathcal{V}}(n_0, m_0)$. Cela signifie que pour toute entrée w de taille (n_0, m_0),

l'automate $\mathcal{A}$ accepte ou refuse le mot avant que l'origine ne voit un bord du mot. L'évolution jusqu'à ce temps est donc exactement la même pour tout mot plus grand qui coïncide avec w sur le rectangle $n_0 \times m_0$. Ainsi, chaque mot w' de taille (n, m) avec $n \geq n_0$ et $m \geq m_0$ est en fait accepté ou refusé en temps $f(n_0, m_0)$. De plus, tous les mots de taille (n, m) avec $n > n_0$ et $m < m_0$ sont reconnus en temps $f(n, m)$, fonction bornée par le temps sous réel. Or, ce temps sous réel est lui même borné par le temps nécessaire pour que l'origine « voit » une cellule dans l'état dièse au dessus de l'entrée. Formellement, en notant $v_v = (x_v, y_v)$ le vecteur de $\mathcal{V}$ le plus haut dans le quart nord est, le temps sous réel sur une entrée de taille (n, m) est borné par $\lceil \frac{m}{y_v} \rceil$ et donc ici par $\lceil \frac{m_0}{y_v} \rceil$. De même, en notant $v_h = (x_h, y_h)$ le vecteur le plus à droite, le temps sous réel est borné par $\frac{n}{x_h}$. Ainsi, tout mot de taille (n, m) est accepté ou refusé par l'automate $\mathcal{A}$ en temps $\max(\left\lceil \frac{m_0}{y_v} \right\rceil, \left\lceil \frac{n_0}{x_h} \right\rceil, f(n_0, m_0))$ ce qui est indépendant de n et m.

□

Ce théorème justifie en quelque sorte que le temps sous réel est le temps minimal pour reconnaître des langages non triviaux. De ce fait, on pourrait vouloir l'étudier au même titre que le temps réel l'a été. Cependant, les prochains résultats montrent que ce temps sous réel ne dispose pas de certaines propriétés de stabilité qui justifient elles aussi l'étude de certaines classes de complexité. Tout d'abord cette classe n'est pas stable par ajout d'un temps constant de calcul :

Proposition 7.3. *Étant donné un voisinage $\mathcal{V}$ et deux entiers k et k' avec $k' > k$, il existe un langage L reconnaissable avec $\mathcal{V}$ en temps* $\mathrm{TSR}_{\mathcal{V}} + k'$ *mais pas en temps* $\mathrm{TSR}_{\mathcal{V}} + k$.

Démonstration. On note $x_{\mathcal{V}}$ l'abscisse du point le plus à droite dans le quart nord est du voisinage $\mathcal{V}$ et $y_{\mathcal{V}}$ l'ordonnée du point le plus haut. Ainsi, on remarque aisément que $\mathrm{TSR}_{\mathcal{V}}(n, m) = \min(\left\lceil \frac{n}{x_{\mathcal{V}}} \right\rceil, \left\lceil \frac{m}{y_{\mathcal{V}}} \right\rceil)$. On note aussi x_h l'abscisse du point le plus à droite sur l'axe des abscisses de l'enveloppe convexe de $\mathcal{V}$. Cette valeur est donc potentiellement un rationnel et non pas un nombre entier. On définit le langage L comme l'ensemble des mots w de taille (n, m) sur l'alphabet $\{0, 1\}$ qui vérifient :

— $\left\lceil \frac{n}{x_{\mathcal{V}}} \right\rceil > \left\lceil \frac{m}{y_{\mathcal{V}}} \right\rceil$

— $w(0, \lfloor (\mathrm{TSR}_{\mathcal{V}} + k')x_h \rfloor) = 1$

Ce langage est aisément reconnaissable en temps $\mathrm{TSR}_{\mathcal{V}} + k'$, il suffit de vérifier que le premier # vu par l'origine vient du haut, de garder en mémoire l'état de la case la plus à gauche que l'origine a vu sur sa propre ligne et de répondre k' étapes après le temps sous réel. La cellule la plus éloignée sur cette ligne de l'origine visible en temps $\mathrm{TSR}_{\mathcal{V}} + k$ a donc pour abscisse $\lfloor (\mathrm{TSR}_{\mathcal{V}} + k)x_h \rfloor$. Or, x_h est un rationnel, il est donc possible de trouver t tel que $(t + k')x_h$ soit entier. Dans ce cas, comme $k' > k$, $\lfloor (\mathrm{TSR}_{\mathcal{V}} + k')x_h \rfloor > \lfloor (\mathrm{TSR}_{\mathcal{V}} + k)x_h \rfloor$. Ainsi, lorsque l'entrée est de taille $(2(t + k')x_{\mathcal{V}}, ty_{\mathcal{V}} - 1)$, le temps sous réel est t et aucun automate ne peut en $(\mathrm{TSR}_{\mathcal{V}} + k)$ lire la cellule voulue. Par conséquent, un tel automate doit répondre pareillement

quelle que soit la valeur de cette cellule dans le mot d'entrée et donc ne peut pas reconnaître le langage L. □

On peut aussi montrer que cette classe de complexité n'est pas stable par multiplication par une constante du temps de calcul :

Proposition 7.4. *Étant donné un voisinage $\mathcal{V}$ et deux rationnels q et q' avec $q' > q$, il existe un langage L reconnaissable avec $\mathcal{V}$ en temps $(1+q')\,\mathrm{TSR}_{\mathcal{V}}$ mais pas en temps $(1+q)\,\mathrm{TSR}_{\mathcal{V}}$.*

Démonstration. La preuve se fait comme précédemment, le temps supplémentaire $(q'-q)\,\mathrm{TSR}_{\mathcal{V}}$ permettant de voir une plus grande proportion du mot d'entrée dès que le temps sous réel est assez grand. □

En plus de ces résultats qui ne dépendent pas du voisinage utilisé, nous pouvons montrer que le choix du voisinage influe drastiquement sur les capacités de reconnaissance en temps sous réel. En effet, les deux prochains théorèmes illustrent des différences entre des voisinages qui paraissent pourtant assez similaires.

Proposition 7.5. *Étant donnés un voisinage $\mathcal{V}$ et deux entiers $k' > k$, il existe un langage reconnu en temps sous réel par un automate avec le voisinage $\mathcal{V}^{k'}$ mais pas reconnaissable avec le voisinage $\mathcal{V}^k$. De plus, si k' n'est pas un multiple de k, il existe aussi un langage reconnu en temps sous réel par un automate avec le voisinage $\mathcal{V}^k$ mais pas reconnaissable avec le voisinage $\mathcal{V}^{k'}$.*

Démonstration. On note comme dans la preuve précédente $x_{\mathcal{V}}$ et $y_{\mathcal{V}}$ l'abscisse et l'ordonnée respectivement de la cellule la plus en haut et de la cellule la plus à droite de V. Ainsi on peut remarquer que :

$$\mathrm{TSR}_{\mathcal{V}^k}(n,m) = \min(\left\lceil \frac{n}{kx_{\mathcal{V}}} \right\rceil, \left\lceil \frac{m}{ky_{\mathcal{V}}} \right\rceil)$$

$$\mathrm{TSR}_{\mathcal{V}^{k'}}(n,m) = \min(\left\lceil \frac{n}{k'x_{\mathcal{V}}} \right\rceil, \left\lceil \frac{m}{k'y_{\mathcal{V}}} \right\rceil).$$

De plus, la cellule la plus au nord qui influence l'état de l'origine au temps t a pour ordonnée $tky_{\mathcal{V}}$ si l'on utilise $\mathcal{V}^k$ et $tk'y_{\mathcal{V}}$ si l'on utilise $\mathcal{V}^{k'}$. Considérons alors le langage composé des mots de l'alphabet $\{0,1\}$ qui contiennent au moins un 1 dans $\mathcal{V}^{\mathrm{TSR}_{\mathcal{V}^{k'}}\,k'}(0)$. Ce langage est très facilement reconnaissable en temps sous réel avec le voisinage $\mathcal{V}^{k'}$. Il suffit de propager les états 1 dans toutes les directions et d'accepter le mot si l'origine est dans l'état 1 au moment de terminer le calcul. Considérons maintenant un mot w de taille $(kk'x_{\mathcal{V}}+1, kk'y_{\mathcal{V}}+k'y_{\mathcal{V}}+1)$. Dans ce cas le temps sous réel avec le voisinage $\mathcal{V}^k$ est $k'+1$ et avec le voisinage $\mathcal{V}^{k'}$, $k+1$. Ainsi la cellule la plus au nord accessible lors d'un calcul en temps sous réel avec le voisinage $\mathcal{V}^k$ a pour ordonnée $kk'y_{\mathcal{V}} + ky_{\mathcal{V}} < kk'y_{\mathcal{V}} + k'y_{\mathcal{V}}$. Par conséquent, un automate avec le voisinage $\mathcal{V}^k$ ne dispose pas d'assez d'informations au temps sous réel pour décider de l'appartenance de w à L.

Supposons maintenant que k' n'est pas un multiple de k. En remplaçant $\mathcal{V}$ par $\mathcal{V}^{\text{pgcd}(k,k')}$, on peut supposer sans perdre de généralités que k et k' sont premiers entre eux et plus grands que 1. On considère le même langage que précédemment mais en s'intéressant désormais aux cellules qui sont visibles avec le voisinage $\mathcal{V}^k$ au lieu de $\mathcal{V}^{k'}$. Comme k et k' sont premiers entre eux, il existe deux entiers p et q tels que $pk' = qk+1$. Lorsque que le mot w en entrée est de taille $(pk'x_{\mathcal{V}}, pk'y_{\mathcal{V}}+ky_{\mathcal{V}}+1)$ le temps sous réel avec le voisinage $\mathcal{V}^k$ est $q+1$ alors que le temps sous réel avec le voisinage $\mathcal{V}^{k'}$ est p. La cellule la plus au nord accessible avec le voisinage $\mathcal{V}^{k'}$ en temps sous réel a donc pour ordonnée $pk'y_{\mathcal{V}} < qky_{\mathcal{V}} + ky_{\mathcal{V}}$,ce qui conclut à nouveau avec le même argument que dans la preuve précédente. □

Proposition 7.6. *Étant donnés deux voisinages $\mathcal{V}$ et $\mathcal{V}'$ ayant la même enveloppe convexe mais qui diffèrent dans le quart nord est. Alors $\mathcal{V} \subset \mathcal{V}' \Leftrightarrow \text{SR}_{\mathcal{V}} \subset \text{SR}_{\mathcal{V}'}$.*

Démonstration. Puisque $\mathcal{V}$ et $\mathcal{V}'$ ont tous les deux la même enveloppe convexe, leurs fonctions de temps sous réel associées sont aussi les mêmes. Dans cette preuve par souci de clarté nous noterons donc cette fonction de temps T.

Supposons que $\mathcal{V} \subset \mathcal{V}'$. Alors, il est possible d'effectuer une simulation d'un automate avec le voisinage $\mathcal{V}$ avec un automate dont le voisinage est $\mathcal{V}'$ sans perdre de temps. Comme ces deux voisinages disposent d'exactement autant de temps, tout ce qui est reconnaissable avec le voisinage $\mathcal{V}$ l'est aussi avec le voisinage $\mathcal{V}'$.

Supposons maintenant que $\mathcal{V}$ ne soit pas inclus dans $\mathcal{V}'$. Comme dans la preuve de la proposition précédente, on définit $1_{\mathcal{V}}$ le langage des mots qui ont au moins un 1 dans $\mathcal{V}^T(0)$. Alors, comme ces deux voisinages diffèrent dans le quart nord est, il existe une cellule de coordonnées positives (x,y) qui est dans $\mathcal{V}(0)$ mais pas dans $\mathcal{V}'(0)$. De plus, cette cellule n'est pas un sommet de l'enveloppe convexe de $\mathcal{V}$. Ceci signifie, toujours avec les notations des preuves précédentes, que $x < x_{\mathcal{V}}$ ou $y < y_{\mathcal{V}}$. Sans perdre de généralités nous pouvons supposer $x < x_{\mathcal{V}}$. Dans ce cas, lorsque l'entrée est de taille $(x_{\mathcal{V}}, y_{\mathcal{V}} + 1)$, le temps sous réel est de 1 et la cellule (x, y) qui correspond à une lettre du mot w peut contenir un 1 ou un 0. Comme cette cellule n'est pas dans le voisinage direct de l'origine pour $\mathcal{V}'$, aucun automate avec ce voisinage de peut reconnaître $1_{\mathcal{V}}$, et donc $\text{SR}_{\mathcal{V}}$ n'est pas inclus dans $\text{SR}_{\mathcal{V}'}$. □

Proposition 7.7. *Étant donné un voisinage $\mathcal{V}$, alors le voisinage $\mathcal{M} = \mathcal{V} \Cup \{(x, y) | 0 \leq x \leq x_{\mathcal{V}}; 0 \leq y \leq y_{\mathcal{V}}\}$ permet de reconnaître strictement plus de langages que le voisinage $\mathcal{V}$, sauf si ces deux voisinages sont identiques.*

Démonstration. Il suffit de se rendre compte que ces deux voisinages ont le même temps sous réel. Ainsi, s'il manque à $\mathcal{V}$ une cellule autre que $(x_{\mathcal{V}}, y_{\mathcal{V}})$, le même argument que dans la preuve précédente donne le résultat escompté. Dans le cas ou $\mathcal{V}$ contient toutes les cellules de $\mathcal{M}$ sauf la cellule $(x_{\mathcal{V}}, y_{\mathcal{V}})$, en prenant un mot de taille $(3x_{\mathcal{V}}, 3y_{\mathcal{V}})$, on peut vérifier qu'au moins une cellule de l'entrée ne sera pas dans $\mathcal{V}^3(0)$. □

Chapitre 8

Conclusion

Les résultats décrits dans ce mémoire permettent un approfondissement de la compréhension des capacités algorithmiques des automates cellulaire en temps réel et en temps linéaire. Plus particulièrement mes travaux précisent l'impact du choix du voisinage sur ces capacités algorithmiques. De toutes les classes étudiées au cours de ma thèse, la plus restreinte est celle du temps sous réel. Cette notion nouvelle de complexité en temps mène en fait à la définition de tout un ensemble de classes, dépendant du voisinage utilisé. Ces classes présentent une très forte instabilité comme expliqué dans le chapitre 7. En effet, deux voisinages différents définissent des classes différentes, et même incomparables dès lors que les deux voisinages ont des formes différentes. De plus ces classes ne sont pas stables, ni par multiplication du temps disponible, ni même par ajout d'un temps constant de calcul.

L'étude du temps réel mène à des résultats plus intéressants et constitue une partie bien plus importante du mémoire. Une fois de plus, cette notion de complexité en temps permet de définir de nombreuses classes de langages, dépendant du voisinage. Contrairement au temps sous réel, les travaux présentés ici permettent d'exhiber des voisinages qui permettent de reconnaître exactement les mêmes langages en temps réel. On prouve en section 6.2 que les voisinages dont l'enveloppe convexe est un losange sont un tel exemple de voisinages qui permettent tous de reconnaître les mêmes langages en temps réel. De plus, la classe de langages définie par ces voisinages présente plus de stabilité que celles associées au temps sous réel. En effet on montre dans la section 4.1 que l'ajout d'un temps constant de calcul ne change pas l'ensemble des langages reconnus. Ces voisinages se démarquent des autres par l'absence de sommet de leur enveloppe convexe de coordonnées strictement positives. La classification des voisinages avec de tels sommets apparaît plus complexe. Certains sommets de l'enveloppe convexe peuvent être limitants, c'est à dire qu'il peuvent correspondre à une voie de communication optimale entre l'origine et la cellule la plus éloignée de celle ci dans le mot d'entrée. On montre dans la section 6.1 que deux voisinages ayant des sommets limitants différents définissent des classes de langages incomparables. De plus, aucun de ces voisinage ne définit une classe plus grande que celle définie par les voisinages losanges. Un cas particulier de

ces voisinages, les voisinages ronds sont quand à eux strictement moins puissants, en temps réel, que les voisinages losanges. Ce résultat se base sur une décomposition d'un voisinage rond en un nombre fini de voisinages losanges qu'il est possible de simuler en parallèle. Le détail de cette preuve est présenté en section 6.3.

Tous ces voisinages offrent la même puissance de calcul en temps linéaire, la vitesse de calcul offert par chacun différant au plus d'une constante multiplicative. Un des résultats majeurs de cette thèse est le théorème d'accélération linéaire présenté en section 4.2. Ce théorème stipule que la constante multiplicative en question peut être ramenée à $(1 + \epsilon)$ pour tout ϵ positif. Plus précisément la classe des langages reconnus en temps linéaire en deux dimension est la même que celle des langages reconnus en temps $(1 + \epsilon)\,\mathrm{TR}$ avec n'importe quel voisinage. La difficulté principale était de trouver un moyen efficace d'effectuer une compression avec n'importe quel voisinage et quelle que soit le rapport entre la longueur et la largeur de l'entrée. La clé de la construction que je propose est une approximation de n'importe quel voisinage par un ensemble fini de voisinages de forme rectangulaires. La compression à l'aide de ces voisinages étant plus simple à effectuer, cette approximation permet une compression en temps quasi-optimal, ce qui est suffisant pour l'expression du théorème.

Une petite partie des résultats présentés dans ce mémoire porte sur la puissance de calcul offerte par un automate cellulaire unidimensionnel et par un automate tridimensionnel. Plus précisément je me suis intéressé aux capacités de reconnaissance d'un automate de dimension d en terme de langages de dimension $(d - 1)$. Dans la section 5.4 on prouve qu'il est possible de reconnaître strictement plus de langages 2D avec un automate 3D utilisant le voisinage de Moore qu'avec un automate 2D utilisant ce même voisinage. Ce résultat repose essentiellement sur la connaissance d'un langage peu complexe (reconnaissable en temps réel avec le voisinage de von Neumann) qui n'est pas reconnaissable en temps réel avec le voisinage de Moore en dimension deux. Il est possible de généraliser la construction de ce langage aux dimensions supérieures, ainsi que la construction qui permet de le reconnaître avec une dimension supplémentaire. On peut donc généraliser ce résultat et annoncer que les automates de dimension $(d+1)$ avec le voisinage de Moore reconnaissent plus de langages en dimension d que les automates en dimension d avec ce voisinage, lorsque d vaut au moins 2. Les limites du temps réel en une dimension sont moins bien comprises et l'absence d'exemple de langage simple mais non reconnaissable en temps réel rend compliquée la preuve de la différence entre reconnaissance en une dimension et en deux dimension. Les résultats de la section 5.3 permettent néanmoins de montrer que l'égalité des capacités de reconnaissance des deux modèles impliquerait une stabilité par concaténation de la classe du temps réel en une dimension.

Enfin parmi les multiples constructions présentées dans ce mémoire, certaines ont un intérêt indépendamment du théorème qu'elles servent à démontrer ici. La plus significative est sans aucun doute la construction de synchronisation passive présentée en section 3.2. Cette construction permet de construire un automate fonctionnant avec une entrée asynchrone qui simule le comportement d'un automate

avec une entrée synchrone sans perdre de temps. C'est-à-dire que l'on peut donner en entrée à un automate cellulaire des cellules qui s'activent une à une avec chacune une partie des informations de l'entrée. Ce lemme est principalement utilisé lors de transformations de l'entrée telles que des compressions, pour assurer ensuite que le calcul effectuée sur l'entrée compressée s'effectue correctement, comme s'il avait démarré exactement au moment où la compression était terminée. La construction présentée dans la section 3.3 concerne les automates cellulaires en dimension un. Elle permet de montrer que la donnée d'un marqueur sur une cellule située dans une certaine portion de l'entrée n'augmente pas la puissance de calcul des automates cellulaires en temps réel. Ce lemme peut être utilisé pour reconnaître des langages en supposant que l'on dispose d'un indice du type « cette cellule est dans le deuxième quart du mot ».

Les travaux de cette thèse permettent de répondre à quelques questions précédemment ouvertes, mais de nombreuses interrogations demeurent. Aussi, nous terminons notre conclusion par un exposé de problèmes ouverts qui constituent des pistes pour des réflexions ultérieures.

Tout d'abord, afin de compléter la classification des voisinages en deux dimensions, il serait intéressant de montrer la conjecture suivante :

Conjecture 8.1. *Les voisinages sans sommets limitants offrent plus de puissance de calcul en temps réel que ceux avec au moins un sommet limitant.*

La relation entre la reconnaissance en temps réel et la reconnaissance en temps linéaire est connue pour un certain ensemble de voisinages, mais reste inconnue en général, et notamment pour le voisinage de von Neumann. La conjecture suivante est à mon avis la première étape de la généralisation à tous les voisinages, même si elle reste sensiblement plus difficile à prouver que la conjecture précédente.

Conjecture 8.2. *Il existe un langage reconnaissable en temps linéaire mais pas en temps réel avec le voisinage de von Neumann en dimension deux.*

Cette question a l'air fortement reliée à son équivalent en dimension un, ouverte de longue date et probablement la question la plus compliquée posée dans cette conclusion.

Question ouverte 8.3. *Existe-t-il un langage reconnaissable en temps linéaire mais pas en temps réel en dimension un ?*

Des pistes plus prometteuses pour des résultats à court terme concerne les deux résultats d'accélération de ce manuscrit. Bien que la généralisation complète du théorème d'accélération constante semble assez complexe, l'étendre à de nouvelles classes de voisinages bien choisies semble accessible.

Conjecture 8.4. *On peut étendre le résultat d'accélération constante à d'autres classes de voisinages en dimension deux.*

Le théorème d'accélération linéaire est déjà complet mais comporte une très légère perte de temps lors de la compression qui empêche d'établir exactement le même théorème qu'en dimension un. Je pense que l'on peut montrer que la compression ne peut pas être faite en temps optimal sans connaître à l'avance le rapport entre la longueur et la largeur de l'entrée.

Conjecture 8.5. *Il n'est pas possible avec un voisinage quelconque $\mathcal{V}$ de compresser une entrée par un facteur k en temps $\frac{(k-1)\,\mathrm{TR}_{\mathcal{V}}}{k}$ sans connaître à l'avance de restriction sur le rapport entre la longueur et la largeur de l'entrée.*

La classification des voisinages en fonction des langages qu'ils permettent de reconnaître en temps réel est l'objectif à long terme de mes travaux. On peut aussi penser à une version duale de cet objectif, à savoir la classification des langages en fonction des voisinages qui permettent de les reconnaître en temps réel. On peut en effet assez facilement trouver des langages reconnaissables avec n'importe quel voisinage ou avec aucun. On a aussi vu des exemples de langages qui étaient reconnaissables en temps réel mais pas avec n'importe quel voisinage. Cette distinction permet un raffinement de la notion de complexité d'un langage. En effet parmi tous les langages qu'il est possible de reconnaître en utilisant seulement le temps de lecture de l'entrée il en existe qui nécessite quand même des calculs d'une complexité trop importante pour pouvoir être effectués avec certains voisinages. Ces langages sont donc plus complexes que certains autres bien qu'ils soient aussi reconnaissable en temps réel. Inversement, parmi les langages reconnaissable seulement en temps linéaire avec certains voisinages, il en existe qui sont aussi reconnaissable en temps réel pour peu que l'on choisisse un voisinage adéquat. Étudier les relations entre les voisinages permet de mieux comprendre les possibilités de raffinement que peut offrir une telle classification.

Bibliographie

[AC97] J. Albert and K. Čulik II. A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1 :1–16, 1987.

[BC07] Mike Bailey and Steve Cunningham. A hands-on environment for teaching gpu programming. *SIGCSE Bull.*, 39(1) :254–258, March 2007.

[Bey69] W.T. Beyer. *Recognition of topological invariants by iterative arrays*. Massachusetts Institute of Technology, Project MAC, 1969.

[CC84] Christian Choffrut and Karel Culik II. On real-time cellular automata and trellis automata. *Acta Inf.*, 21 :393–407, 1984.

[Col69] Stephen N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machines. *IEEE Trans. Computers*, 18(4) :349–365, 1969.

[DP07] Martin Delacourt and Victor Poupet. Real time language recognition on 2d cellular automata : Dealing with non-convex neighborhoods. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 298–309. Springer, 2007.

[DM02] Marianne Delorme and Jacques Mazoyer. Reconnaissance parallele des langages rationnels sur automates cellulaires plans. *Theor. Comput. Sci.*, 281(1-2) :251–289, 2002.

[Den88] B. Denby. Neural networks and cellular automata in experimental high energy physics. *Computer Physics Communications*, 49(3) :429 – 448, 1988.

[EE93] G.Bard Ermentrout and Leah Edelstein-Keshet. Cellular automata approaches to biological modeling. *Journal of Theoretical Biology*, 160(1) :97 – 133, 1993.

[Fis65] Patrick C. Fischer. Generation of primes by a one-dimensional real-time iterative array. *J. ACM*, 12(3) :388–394, 1965.

[FNI17] Toru Fujita, Koji Nakano, and Yasuaki Ito. Fast simulation of conway's game of life using bitwise parallel bulk computation on a gpu. *International Journal of Foundations of Computer Science*.

[Hed69] G. A. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Mathematical systems theory*, 3(4) :320–375, 1969.

[Hen61] F.C. Hennie. *Iterative Arrays of Logical Circuits*. MIT Press Classics. MIT Press, 1961.

[Hen65] F.C. Hennie. One-tape, off-line turing machine computations. *Information and Control*, 8(6) :553 – 578, 1965.

[IJ88] Oscar H. Ibarra and Tao Jiang. Relating the power of cellular arrays to their closure properties. *Theor. Comput. Sci.*, 57 :225–238, 1988.

[IKM85] Oscar H. Ibarra, Sam M. Kim, and Shlomo Moran. Sequential machine characterizations of trellis and cellular automata and applications. *SIAM J. Comput.*, 14(2) :426–447, 1985.

[KMH01] T. Kobori, T. Maruyama, and T. Hoshino. A cellular automata system with fpga. In *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, pages 120–129, March 2001.

[KM10] Martin Kutrib and Andreas Malcher. Transductions computed by iterative arrays. In Jarkko Kari, editor, *Second Symposium on Cellular Automata "Journeacute ;es Automates Cellulaires", JAC 2010, Turku, Finland, December 15-17, 2010. Proceedings*, pages 156–167. Turku Center for Computer Science, 2010.

[Maz86] Jacques Mazoyer. A six states minimal time solution to the firing squad synchronization problem. *Publications du Département de mathématiques (Lyon)*, (1A) :1–92, 1986.

[Maz96] Jacques Mazoyer. On optimal solutions to the firing squad synchronization problem. *Theor. Comput. Sci.*, 168(2) :367–404, 1996.

[MR92] Jacques Mazoyer and Nicolas Reimen. A linear speed-up theorem for cellular automata. *Theor. Comput. Sci.*, 101(1) :59–98, 1992.

[Oll02] Nicolas Ollinger. *Automates cellulaires : structures*. PhD thesis, École Normale Supérieure de Lyon, 2002.

[PA08] Kalyan S. Perumalla and Brandon G. Aaby. Data parallel execution challenges and runtime performance of agent simulations on gpus. In *Proceedings of the 2008 Spring Simulation Multiconference*, SpringSim '08, pages 116–123, San Diego, CA, USA, 2008. Society for Computer Simulation International.

[PG99] Reid Porter and Neil Bergmann. *Evolving FPGA Based Cellular Automata*, pages 114–121. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[Pou05] Victor Poupet. Cellular automata : Real-time equivalence between one-dimensional neighborhoods. In *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, pages 133–144, 2005.

[Pou06] Victor Poupet. *Automates cellulaires : temps réel et voisinages*. PhD thesis, École Normale Supérieure de Lyon, 2006.

[Rap98] IVAN RAPAPORT. *Ordre induit sur les automates cellulaires par l'operation de regroupement*. PhD thesis, 1998. Thèse de doctorat dirigée par Mazoyer, Jacques Sciences appliquées École normale supérieure (Lyon) 1998.

[Ric08] Gaétan Richard. *Systèmes de particules et collisions discr'etes dans les automates cellulaires*. PhD thesis, Université de provence Aix-Marseille, 2008.

[RNSL96] M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour. Two lane traffic simulations using cellular automata. *Physica A : Statistical Mechanics and its Applications*, 231(4) :534 – 550, 1996.

[Rok99] Zsuzsanna Róka. Simulations between cellular automata on Cayley graphs. *Theoretical Computer Science*, 225(1-2) :81–111, 1999.

[Ros67] Arnold L. Rosenberg. Real-time definable languages. *J. ACM*, 14(4) :645–662, October 1967.

[Smi71] Alvy R. Smith III. Simple computation-universal cellular spaces. *J. ACM*, 18(3) :339–353, 1971.

[Smi71a] Alvy R. Smith III. Two-dimensional formal languages and pattern recognition by cellular automata. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (Swat 1971)*, SWAT '71, pages 144–152, Washington, DC, USA, 1971. IEEE Computer Society.

[Smi72] Alvy R. Smith III. Real-time language recognition by one-dimensional cellular automata. *Journal of the Assoc. Comput. Mach.*, 6 :233–253, 1972.

[Ter99] Véronique Terrier. Two-dimensional cellular automata recognizer. *Theor. Comput. Sci.*, 218(2) :325–346, 1999.

[Ter04] Véronique Terrier. Two-dimensional cellular automata and their neighborhoods. *Theor. Comput. Sci.*, 312(2-3) :203–222, 2004.

[Tof84] Tommaso Toffoli. Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D : Nonlinear Phenomena*, 10(1) :117 – 127, 1984.

[vN66] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, IL, USA, 1966.

[Wag86] K. Wagner and G. Wechsung. *Computational Complexity*. Springer Netherlands, 1986.

[Whi07] S. Hoya White, A. Martín del Rey, and G. Rodríguez Sánchez. Modeling epidemics using cellular automata. *Applied Mathematics and Computation*, 186(1) :193 – 202, 2007.

Publications en rapport avec la thèse

[GP15a] Anaël Grandjean and Victor Poupet. Comparing 1d and 2d real time on cellular automata. In *STACS*, volume 30 of *LIPIcs*, pages 367–378. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[GP16] Anaël Grandjean and Victor Poupet. A linear acceleration theorem for 2d cellular automata on all complete neighborhoods. In *ICALP*, volume 55 of *LIPIcs*, pages 115 :1–115 :12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[Gra16] Anaël Grandjean. Constant acceleration theorem for extended von neumann neighbourhoods. In *Automata*, volume 9664 of *Lecture Notes in Computer Science*, pages 149–158. Springer, 2016.

Autres publications

[DGG14] Bruno Durand, Guilhem Gamard, and Anaël Grandjean. Aperiodic tilings and entropy. In *Developments in Language Theory*, volume 8633 of *Lecture Notes in Computer Science*, pages 166–177. Springer, 2014.

[GLU12] Anaël Grandjean, Johannes Langguth, and Bora Uçar. On optimal and balanced sparse matrix partitioning problems. In *CLUSTER*, pages 257–265. IEEE Computer Society, 2012.

[GP15b] Anaël Grandjean and Victor Poupet. L-convex polyominoes are recognizable in real time by 2d cellular automata. In *Automata*, volume 9099 of *Lecture Notes in Computer Science*, pages 127–140. Springer, 2015.

[GRT12] Anaël Grandjean, Gaétan Richard, and Véronique Terrier. Linear functional classes over cellular automata. In *AUTOMATA & JAC*, volume 90 of *EPTCS*, pages 177–193, 2012.

[GU14] Anaël Grandjean and Bora Uçar. On partitioning two dimensional finite difference meshes for distributed memory parallel computers. In *PDP*, pages 9–16. IEEE Computer Society, 2014.

Abstract

Cellular automata were introduced in the 50s by J. von Neumann and S. Ulam and are now studied as a model of parallel computation. My studies focus on the impact of the definition of neighborhood on the algorithmic abilities of the whole automata, that is the way cells are linked together. This impact is well understood for one dimensionnal cellular automata, therefore this manuscript mainly discuss two dimensional properties. In order to witness some differences we are interested in small complexity classes like real time and linear time. We first expend some well known properties of those classes to automata with more general definitions of neighborhood. We are then able to show that some neighborhoods are equivalent whereas some define incomparable language classes.

Keywords : Cellular Automata, Langage Recognition, Real Time, Complexity

Résumé

Les automates cellulaires sont un modèle de calcul massivement parallèle apparu dans les années 50. Il existe plusieurs manières de définir des automates cellulaires, en faisant varier la dimension par exemple ou en faisant varier les possibilité de communication entre les différentes cellules de calcul. En effet, chaque cellule ne peut communiquer instantanément qu'avec un nombre fini d'autres cellules appelées ses voisines. Mes travaux s'intéressent principalement à l'impact du choix des voisines sur les capacités algorithmiques de ce modèle. Cet impact étant bien compris en une dimension, mes travaux portent majoritairement sur les automates cellulaires bidimensionnels. J'ai tout d'abord généralisé des propriétés classiques de certaines classes de complexité au plus de voisinages possibles. On arrive notamment à un théorème d'accélération linéaire valable pour tous les voisinages. J'ai ensuite étudié les différences entre ces classes de complexités en fonction du voisinage choisi. Ces travaux ont permis d'exhiber des voisinages définissant des classes incomparables, ainsi que des ensembles de voisinages définissant exactement les mêmes classes de complexité. Enfin, je présente aussi des travaux sur les différence de puissance de calcul entre les automates de dimensions différentes.

Mots clefs : Automate cellulaire, Reconnaissance de langages, Temps réel, Complexité