N° d'ordre : 390
N° attribué par la bibliothèque : 06ENSL0390

École Normale Supérieure de Lyon
Laboratoire de l'Informatique du Parallélisme

THÈSE

présentée et soutenue publiquement le 8 décembre 2006 par

Victor Poupet

pour l'obtention du grade de

Docteur de l'École Normale Supérieure de Lyon
spécialité : Informatique

au titre de l'École doctorale de mathématiques et d'informatique fondamentale de Lyon

Automates cellulaires : temps réel et voisinages

Directeurs de thèse :	Marianne Delorme
	Jacques Mazoyer
Après avis de :	Étienne Grandjean
	Kenichi Morita

Devant la commission d'examen formée de :

Marianne Delorme	Membre
Bruno Durand	Membre
Étienne Grandjean	Membre/Rapporteur
Martin Kutrib	Membre
Jacques Mazoyer	Membre
Kenichi Morita	Membre/Rapporteur
Véronique Terrier	Invitée

À Pierre.

Table des matières

Remerciements

Il est pénible de réaliser qu'après avoir finalement réussi à écrire tout le contenu scientifique de ce manuscrit de thèse, il m'est si difficile d'écrire la page de remerciements. Non pas bien entendu que je n'aie personne à remercier (cela aurait au contraire bien allégé ma tâche) mais que tellement de personnes aient contribué au bon déroulement de ma thèse. J'aurais voulu tant que possible éviter de devoir achever cette section par le traditionnel « et aussi tous ceux que j'oublie », mais il sera malheureusement impossible de dresser une liste suffisamment exhaustive pour être satisfaisante.

Bien qu'il ne me soit pas possible de détailler à quel point chacune des personnes nommées ici m'a été d'une aide précieuse (pour des sombres histoires de limitation de place) je tiens à signaler que mes années de thèse ainsi que mes années en tant qu'élève à l'École normale supérieure de Lyon ont été parmi les plus intéressantes et agréables, tout simplement parce que j'ai eu la chance de les passer dans un cadre aussi sympathique, stimulant et épanouissant (là je parle des personnes qui sont remerciées par la suite... pas des arbres, des oiseaux ou des fleurs).

Prenant le risque de passer pour un conformiste, je remercie en premier lieu mes deux directeurs de thèse, Marianne Delorme et Jacques Mazoyer, non seulement pour l'encadrement de la thèse proprement dite, mais également pour m'avoir donné goût à l'informatique théorique, d'abord en tant que professeurs depuis mon arrivée à l'ENS puis en tant que chercheurs pendant trois ans. Je suis très fier d'avoir été leur élève et j'espère qu'un jour je pourrai à mon tour transmettre à mes propres élèves tout ce qu'ils m'ont apporté.

Une grande partie de la réussite de ma thèse revient également aux autres thésards du LIP (ou ex-thésards pour certains) et tout particulièrement à Emmanuel Jeandel, Vincent Nesme, Stéphane Le Roux, Guillaume Theyssier, Sylvain Périfel et Guillaume Melquiond. Je suis persuadé qu'en ce qui concerne la recherche scientifique, pouvoir mélanger aussi facilement la détente et le travail est un merveilleux stimulant et je ne doute pas un instant que les discussions cinématographiques, les parties de jeux vidéos en réseau ou encore les grilles de mots-fléchés aient fortement contribué à la mise en place des idées qui sont dans cette thèse (bon, j'avoue que je peux le dire maintenant, mais qu'avant de commencer la rédaction je ne me serais pas prononcé aussi nettement).

Même si pour un thésard, les autres thésards représentent une très grande proportion de la communauté dans laquelle il travaille, il ne faut pas oublier les autres membres du laboratoire. Les conversations avec Natacha Portier et Éric Thierry notamment me manqueront beaucoup lorsque je quitterai le LIP (je me sentirai bien seul si dans mon prochain laboratoire personne ne vient me chercher à 11h30...).

Je dois bien sûr également remercier les différentes personnes extérieures au laboratoire avec qui j'ai eu l'occasion de travailler. En particulier Véronique Terrier pour de nombreuses conversations et idées que j'ai ignoblement volées pour les mettre dans cette thèse (avec permission quand même), mais aussi Bruno Durand, Nicolas Ollinger et Kenichi Morita qui m'ont donné l'occasion de réfléchir à des problèmes qui n'étaient pas ceux de ma thèse (ce qui est toujours agréable).

Je remercie aussi mes parents et mes trois sœurs d'avoir écouté (ou feint d'écouter de manière suffisamment crédible) mes monologues scientifiques et de m'avoir donné un point de vue non scientifique appréciable (et difficile à trouver dans un labo d'informatique).

Pour l'inspiration qu'ils m'ont apportée, je remercie enfin spécialement Ludwig, Johannes, Edgar, Charles, John et Danny ainsi qu'Ernest Sproutch et tous ceux que j'oublie...

Introduction

Turing universalité et automates cellulaires

Les automates cellulaires sont un puissant modèle de calcul introduits au début des années 50 par le mathématicien américain (d'origine hongroise) John Von Neumann qui s'intéressait alors à l'auto-reproduction des systèmes artificiels [18].

Depuis leur création ils ont été étudiés dans divers domaines comme par exemple la physique ou la biologie où ils permettent de modéliser et de simuler divers phénomènes qui ne peuvent pas être analysés directement. Ils ont aussi été regardés en tant que systèmes dynamiques discrets car, bien que leur définition soit très simple, ils sont capables de produire des comportements complexes difficiles à prévoir [1] et fournissent ainsi un outil théorique pour l'étude des systèmes complexes discrets.

En 1969 Alvy Ray Smith, alors étudiant en thèse à l'université de Stanford, montre que les automates cellulaires peuvent être vus comme des objets calculant des fonctions. Il prouve ensuite que ce modèle de calcul est aussi puissant que les machines de Turing et établit les fondements de la théorie de la complexité sur automates cellulaires [16].

Les automates cellulaires deviennent alors, par leur simplicité de description et leur uniformité, un modèle privilégié du calcul massivement parallèle (par opposition aux modèles usuels tels que les machines de Turing ou les machines RAM qui sont fondamentalement séquentiels).

De nombreux travaux sont ensuite réalisés pour mieux comprendre la notion de complexité sur ce nouveau modèle (équivalences efficaces avec les autres modèles existants [10], théorèmes d'accélérations [13], résultats d'indécidabilité [16, 5], etc.).

Les automates cellulaires se prêtent également très bien au passage en dimensions supérieures à 1 (les premiers automates cellulaires introduits par John Von Neumann étaient bidimensionnels et le célèbre « jeu de la vie » inventé par le mathématicien John Conway également) ce qui ouvre de nombreuses perspectives dans des domaines aussi divers que le traitement d'images, l'organisation d'un calcul sur de grands réseaux réguliers de machines ou encore la représentation de systèmes dynamiques tridimensionnels qui n'avaient pas pu être explorées à l'aide des modèles de calcul précédents.

La règle de transition locale et le voisinage « minimal »

Tout comme dans le cas des machines de Turing, et pour que l'automate soit descriptible de manière finie, on impose que les automates cellulaires aient un comportement local : bien qu'étant constitués d'une infinité de cellules identiques qui sont toutes capables d'évoluer (à la différence de la machine de Turing qui ne possède qu'un nombre fini de têtes capables de travailler, le reste du ruban étant alors inerte en l'absence d'une tête de travail) l'évolution d'une cellule donnée ne peut être affectée que par les états d'un certain nombre fini de cellules qui se trouvent dans une zone limitée autour d'elle. L'ensemble des cellules qui peuvent affecter en un temps le comportement d'une cellule donnée est appelé le voisinage de cette dernière, et le voisinage est identique (à translation près) pour chaque cellule. L'évolution d'une cellule en un temps ne dépend alors que de son propre état et de celui des cellules se trouvant dans son voisinage, ses *voisines*.

Puisque toutes les cellules sont supposées identiques, le fonctionnement de l'automate peut être entièrement décrit par les états de ses cellules au temps initial et la donnée d'une « fonction de transition locale » qui décrit le fonctionnement d'une cellule en fonction des états de ses voisines.

Bien qu'il existe également d'autres définitions équivalentes des automates cellulaires, certaines ne nécessitant pas le concept de fonction de transition locale comme par exemple la définition due à Gustav A. Hedlund des automates cellulaires comme étant l'ensemble des fonctions continues de l'ensemble des mots bi-infinis dans lui-même qui commutent avec le décalage [9], la fonction de transition locale est considérée comme étant le cœur d'un automate cellulaire en ce sens qu'elle le caractérise entièrement de manière finie.

Vue comme une « description élémentaire » d'un automate cellulaire, il est naturel de vouloir réduire cette fonction à sa représentation essentielle, en supprimant toute information redondante ou superflue. C'est alors que la notion de « voisinage minimal » intervient.

Jusqu'ici nous avons vu que le voisinage était un ensemble fini de cellules qui pouvaient avoir une influence sur le comportement d'une cellule donnée en un temps. Si l'on note V un voisinage et Q un ensemble d'états, la fonction de transition locale de l'automate est de taille

$$|Q|^{|Q|^{|V|}}$$

qui dépend donc grandement du cardinal de V. Si l'on considère un voisinage V' plus grand que V (au sens de l'inclusion), il est possible de définir le même automate (en ce qui concerne son évolution globale à partir de toute configuration) sur ce nouveau voisinage mais la fonction locale est alors bien plus grande. On peut alors s'intéresser au plus petit voisinage sur lequel il est possible de définir l'automate cellulaire, qui correspond aux cellules qui d'une part peuvent influencer l'état d'une cellule en un temps mais de plus ont une véritable influence c'est-à-dire que la fonction de transition locale n'est pas constante lorsque l'on fixe toutes les autres cellules du voisinage.

Cette idée de voisinage minimal nous permet de voir qu'à un automate cellulaire donné correspond exactement un unique voisinage sur lequel il doit être

naturellement considéré, et l'on peut facilement justifier que tout voisinage est voisinage minimal d'un automate donné.

Cependant l'analyse systématique des automates cellulaires en tant que machines de calcul, probablement très fortement influencée par son équivalent dans le cas des machines de Turing, n'attache que très peu d'intérêt à la notion de voisinage d'un automate cellulaire (qui n'a pas véritablement d'équivalent dans le cas Turing) et la plupart des travaux effectués se limitent bien souvent à quelques voisinages particuliers, ou des familles bien spécifiques.

L'importance du voisinage

Le voisinage est pourtant une caractéristique fondamentale d'un automate cellulaire. Outre l'aspect « minimaliste » qui permet d'obtenir la plus petite fonction de transition locale, on sait par exemple qu'il est possible d'effectuer une même tâche (calcul d'une fonction par exemple) avec exponentiellement moins d'états si l'on s'autorise un voisinage linéairement plus grand, de même que l'on peut construire un automate nécessitant linéairement moins de temps en augmentant linéairement le voisinage. Le voisinage intervient donc significativement dans toute notion de complexité, et dans les idées de compromis (« *trade-off* » en anglais) permettant de privilégier une grandeur au détriment d'une autre (les *trade-off* font ici intervenir le temps, l'espace, le nombre d'états et le voisinage).

Lorsque l'on s'intéresse à des notions fines de complexité (en nombre d'états, en temps ou en espace par exemple) il est donc indispensable de tenir compte du voisinage. Certains voisinages se sont imposés naturellement comme étant « canoniques » et ont ainsi reçu une attention particulière. En dimension 1 le voisinage $\{-1, 0, 1\}$ correspondant aux « plus proches voisins » est sans conteste le plus couramment utilisé.

En dimension 2 et au-delà la situation est plus complexe car il existe deux voisinages qui se disputent la place (très convoitée) de « voisinage le plus naturel » : le voisinage de Von Neumann correspondant en dimension 2 aux 4 plus proches voisins d'une cellule (en forme de croix) et le voisinage de Moore des 8 plus proches voisins (en forme de carré).

Il est facile de justifier que ces deux voisinages sont « linéairement » équivalents en ce sens que tout calcul effectué par un automate cellulaire fonctionnant sur l'un d'entre eux peut être également effectué par un automate cellulaire fonctionnant sur l'autre voisinage avec un ralentissement au plus linéaire. Toutefois on peut montrer que si l'on considère des classes de complexité inférieure au temps linéaire les capacités algorithmiques de ces deux voisinages sont différentes : de nombreuses techniques s'appliquent très bien sur l'un d'entre eux mais pas sur l'autre et si l'on considère la classe des langages reconnus en temps réel, où le « temps réel » est le plus petit temps tel que toutes les lettres du mot donné en entrée aient pu atteindre l'origine ce qui correspond informellement au plus petit temps « raisonnable » pour qu'un automate puisse reconnaître un mot, on peut montrer que les langages reconnus en temps réel par des automates cellulaires travaillant sur le voisinage de Von Neumann ne sont pas tous reconnus en temps réel sur le voisinage de Moore [17].

Organisation du manuscrit

Dans cette thèse nous chercherons donc à déterminer l'impact du choix du voisinage sur les capacités algorithmiques d'une classe d'automates cellulaires. L'équivalence linéaire des voisinages nous pousse à nous intéresser aux classes de complexité inférieure au temps linéaire, et tout particulièrement au temps réel.

Le document sera structuré de la manière suivante :

Après avoir défini les notions qui nous serviront au cours de cette étude, nous rappellerons les principales techniques et les résultats existants concernant la reconnaissance de langages sur automates cellulaires.

Puis dans un premier temps nous nous concentrerons sur les automates cellulaires en dimension 1. En généralisant un résultat d'accélération par une constante connu dans le cas du voisinage standard $\{-1, 0, 1\}$ nous montrerons qu'il est possible de prouver une très forte équivalence entre les différents voisinages en dimension 1. En effet, tout voisinage en dimension 1 assez complet pour permettre la reconnaissance de langages est équivalent à l'un des deux voisinages bien connus $\{-1, 0, 1\}$ (le voisinage standard) ou $\{0, 1\}$ (le voisinage « one-way ») au sens du temps réel c'est-à-dire qu'il permet de reconnaître les mêmes langages en temps réel. Ce théorème d'équivalence nous permet alors de clore la question posée initialement dans le cas de la dimension 1 (il est connu que les deux voisinages $\{-1, 0, 1\}$ et $\{0, 1\}$ ne sont pas équivalents au sens du temps réel).

Nous verrons alors comment il est possible de généraliser ce résultat aux dimensions supérieures. Il est déjà connu que les voisinages en dimension 2 ne sont pas tous équivalents (le voisinage de Von Neumann et le voisinage de Moore ne permettent pas de reconnaître les mêmes langages en temps réel par exemple) et donc il n'est pas possible d'obtenir une équivalence aussi forte qu'en dimension 1. Cependant nous pouvons montrer que tout langage reconnu en temps réel par un automate cellulaire fonctionnant sur l'enveloppe convexe d'un voisinage V est également reconnu en temps réel par un automate cellulaire fonctionnant sur le voisinage V. Bien que de moins grande portée que le théorème obtenu en dimension 1, ce résultat nous permet alors de ne plus considérer que des voisinages convexes en dimension 2 et au-delà lorsque l'on s'intéressera à des cas de reconnaissance en temps réel ce qui simplifiera grandement notre étude.

Tout en restant dans le cadre des automates cellulaires en dimension 2, nous nous intéresserons ensuite aux théorèmes d'accélération par une constante, c'est-à-dire les résultats indiquant que, selon un voisinage V, tout langage reconnu en temps réel plus une constante est également reconnu en temps réel. Ces théorèmes sont bien connus dans le cas des machines de Turing et des automates cellulaires unidimensionnels fonctionnant sur le voisinage usuel et les théorèmes précédemment évoqués permettent de les étendre à tout voisinage unidimensionnel. Nous verrons qu'en dimension 2 l'accélération constante est facile à réaliser sur le voisinage de Moore en utilisant la même méthode qu'en dimension 1 mais qu'elle est plus compliquée sur le voisinage de Von Neumann. En utilisant des techniques similaires à celles utilisées pour montrer des théorèmes d'accélération linéaire nous montrerons que tout langage reconnu en temps réel plus une constante sur le voisinage de Von Neumann peut également être reconnu en temps réel plus un. Nous étendrons ensuite cette technique à de nombreux voisinages sur lesquels on peut obtenir une telle accélération constante

« partielle » (ne permettant pas de se ramener exactement au temps réel) et l'étude de ce problème nous conduira tout naturellement à la recherche de théorèmes d'accélérations linéaires.

Ici encore, le choix du voisinage semble avoir un grand impact. En effet, les théorèmes d'accélération sont faciles à montrer en dimension 1 (la preuve est semblable à ce qu'elle est dans le cas Turing) et en dimension 2 sur les voisinages de Moore et de Von Neumann. Toutefois, les méthodes usuelles ne sont pas applicables sur certains voisinages en dimension 2. En effet, même en se restreignant à des voisinages convexes à l'aide du théorème annoncé précédemment nous verrons que la symétrie joue un rôle déterminant. Plus surprenant encore, même dans le cas de voisinages symétriques nous verrons que lorsque les voisinages deviennent suffisamment grands pour que leur enveloppe convexe devienne un polygone ayant suffisamment de côtés l'accélération linéaire n'est plus possible selon la méthode usuelle.

Sans toutefois réussir à montrer que de tels théorèmes sont impossibles sur ces voisinages nous donnerons tout de même quelques idées permettant de mettre en évidence l'impossibilité d'effectuer des opérations simples telles que le déplacement d'une information depuis une cellule quelconque vers l'origine qui ne posent aucune difficulté lorsque l'on travaille sur les deux voisinages habituels de Von Neumann et Moore.

Nous continuerons ensuite notre étude en comparant les possibilités algorithmiques de voisinages de dimensions différentes. Un rapide argument de dénombrement permet de justifier qu'il n'est pas possible pour un automate cellulaire en dimension d de simuler linéairement le comportement d'un automate cellulaire en dimension $(d + 1)$. Nous montrerons alors qu'il est possible pour un automate cellulaire fonctionnant en dimension 2 d'effectuer le calcul d'un automate cellulaire fonctionnant en dimension 3 avec un ralentissement polynomial de degré 4.

Dans le dernier chapitre de ce document nous reviendrons sur les théorèmes d'équivalence au sens du temps réel et linéaire obtenus en dimension 1 pour nous pencher sur une célèbre question ouverte de complexité sur automates cellulaires : les langages reconnus en temps linéaires par des automates cellulaires de dimension 1 sont ils également reconnus en temps réel ?

Nous nous sommes penchés sur cette question, initialement posée par Alvy Ray Smith dans son article de 1969, car elle représente une sorte de réciproque de notre théorème d'équivalence des voisinages au sens du temps réel. En effet, il est facile de voir que l'équivalence au sens du temps réel découlerait immédiatement de l'équivalence linéaire (bien plus naturelle et élémentaire) si les langages reconnus en temps réel et en temps linéaire coïncidaient sur les voisinages en dimension 1.

Réciproquement, nous nous sommes demandés si l'équivalence au sens du temps réel (que nous avons pu obtenir par des moyens différents) permettrait de répondre à la question de Smith. Par ailleurs, l'équivalence des voisinages au sens du temps réel permet de justifier que cette question est indépendante du voisinage choisi et qu'elle est donc intrinsèque aux automates cellulaires en dimension 1 dans leur ensemble.

En cherchant à montrer que les deux classes étaient distinctes, nous nous sommes intéressés à une question plus large visant à déterminer si tout langage reconnu en espace linéaire était reconnu en temps réel. Nous montrerons alors qu'en cas de réponse affirmative tout langage reconnu en espace f pour toute

fonction f « constructible » serait reconnu en temps f. Enfin nous explorerons les conséquences d'une telle propriété dans le cas des machines de Turing et les liens entre le calcul séquentiel et parallèle.

Nous conclurons finalement en revenant sur l'ensemble du travail effectué et en développant les problématiques mises en évidence et les perspectives pour des travaux ultérieurs.

Chapitre 1

Préliminaires

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

Lewis Carroll – *Jabberwocky*

1.1 Automates cellulaires

Un automate cellulaire peut être vu comme une infinité de petites machines identiques : les cellules (parallélisme massif), réparties sur un graphe régulier (espace discret). Les cellules n'ont qu'une mémoire finie, c'est-à-dire qu'elles ne peuvent être que dans un nombre fini d'états possibles et qu'elles ne portent pas d'autre information que leur état. À chaque étape de temps (temps discret et synchrone), chaque cellule change son état courant en fonction de son état et de celui de ses voisines uniquement (localité).

Formellement, on a la définition suivante :

Définition 1.1.1 (Automate cellulaire)
Un automate cellulaire *est un quadruplet* $\mathcal{A} = (d, \mathcal{Q}, \delta, V)$ *où*

- $d \in \mathbb{N}$ *est la* dimension *de l'automate (le graphe sur lequel se trouvent les cellules est alors* $\mathbb{Z}^d$*) ;*
- $\mathcal{Q}$ *est un ensemble fini dont les éléments sont appelés* états *de l'automate ;*
- $V \subseteq Z^d$ *est un ensemble fini de vecteurs appelé* voisinage*. On considérera toujours que* 0 *appartient à* V *;*
- $\delta : \mathcal{Q}^V \to Q$ *est la fonction de transition locale de l'automate.*

Étant donné un automate cellulaire $\mathcal{A} = (d, \mathcal{Q}, \delta, V)$, on appelle configuration de $\mathcal{A}$ toute application $\mathfrak{C}$ de $\mathbb{Z}^d$ dans $\mathcal{Q}$. Selon la description informelle que l'on a donnée précédemment, pour tout $c \in \mathbb{Z}^d$, $\mathfrak{C}(c)$ est l'état dans lequel se trouve

la machine qui se trouve à la position c sur le graphe $\mathbb{Z}^d$ dans la configuration $\mathfrak{C}$. Le voisinage d'une cellule c est alors

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

C'est l'ensemble des cellules dont la cellule c peut voir l'état avant de changer le sien. On peut ainsi définir la restriction locale de $\mathfrak{C}$ au voisinage de c par

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

qui représente très précisément ce que la cellule c voit dans chacune des directions du voisinage. Ceci nous amène tout naturellement à la définition suivante d'évolution de l'automate.

Définition 1.1.2
Soit $\mathcal{A} = (d, \mathcal{Q}, \delta, V)$ un automate cellulaire. Soit $\mathfrak{C} : \mathbb{Z}^d \to \mathcal{Q}$ une configuration de $\mathcal{A}$, on appelle successeur *par $\mathcal{A}$ de $\mathfrak{C}$ la configuration définie par*

$$\mathfrak{C}' : \left\{ \begin{array}{ccc} \mathbb{Z}^d & \to & \mathcal{Q} \\ c & \mapsto & \delta(V_{\mathfrak{C}}(c)) \end{array} \right.$$

La fonction qui à une configuration $\mathfrak{C}$ fait correspondre son successeur par $\mathcal{A}$ est entièrement définie par l'automate (principalement le voisinage et la fonction de transition locale). Elle est appelée *fonction de transition globale* de l'automate $\mathcal{A}$.

Les automates cellulaires étant fréquemment étudiés en tant que systèmes dynamiques, il est courant de confondre la fonction de transition globale avec l'automate lui-même. Par abus de notation, nous adopterons donc cette convention et l'on notera $\mathcal{A}(\mathfrak{C})$ le successeur de la configuration $\mathfrak{C}$.

Étant donnée une configuration $\mathfrak{C}$, la suite $(\mathcal{A}^i(\mathfrak{C}))_{i\in\mathbb{N}}$ définie par

$$\left\{ \begin{array}{rcl} \mathcal{A}^0(\mathfrak{C}) & = & \mathfrak{C} \\ \forall i \in \mathbb{N}, \quad \mathcal{A}^{i+1}(\mathfrak{C}) & = & \mathcal{A}(\mathcal{A}^i(\mathfrak{C})) \end{array} \right.$$

est entièrement fixée. On l'appelle l'*évolution* de $\mathcal{A}$ à partir de la configuration initiale $\mathfrak{C}$. On dira alors que l'automate cellulaire $\mathcal{A}$ est dans la configuration $\mathcal{A}^i(\mathfrak{C})$ au temps i.

Remarque. La définition de la fonction δ comme fonction de $\mathcal{Q}^V$ dans $\mathcal{Q}$ étant fastidieuse on supposera souvent que le voisinage V est ordonné. Si l'on a alors $V = (v_1, v_2, \ldots, v_s)$, on dira que δ est une fonction de $\mathcal{Q}^s$ dans $\mathcal{Q}$ et l'on autorisera la notation $\delta(q_1, \ldots, q_s) = q$.

1.2 Voisinages

En dimension d, soit V un voisinage ($V \subseteq \mathbb{Z}^d$). On utilisera les notations suivantes :

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

Remarque. Les notations précédentes ne sont techniquement pas cohérentes puisque l'on choisit de noter multiplicativement la somme de deux ensembles ($V^{k+1} = V + V^k$). Cependant cette convention prête moins à confusion car il est courant de considérer que pour un ensemble $E \subseteq \mathbb{Z}^d$, $k.E = \{k.e \mid e \in \mathbb{Z}^d\}$. Par ailleurs la notation que l'on a choisie ici est justifiée par le fait que l'on voit souvent le voisinage comme une application de $\mathbb{Z}^d$ dans les parties de $\mathbb{Z}^d$ (comme par exemple dans la notation $V(c)$), voire même des parties de $\mathbb{Z}^d$ dans lui-même (notation $V(E)$).

1.2.1 Enveloppes convexes

Lorsque nous étudierons les propriétés de différents voisinages du point de vue des possibilités de reconnaissance de langages, nous verrons que la vitesse maximale à laquelle une information peut se propager sur un automate cellulaire dans une direction donnée est très importante. Si l'on ne s'intéresse qu'au temps de parcours sur des distances très grandes (vitesse asymptotique dans une direction donnée) cette vitesse dépend uniquement de la forme de l'enveloppe convexe du voisinage considéré.

On a les définitions suivantes :

Définition 1.2.1 (Enveloppe convexe continue)
Étant donné un voisinage $V = \{v_1, \ldots, v_p\}$ en dimension d, on appelle enveloppe convexe continue *de V l'ensemble de $\mathbb{R}^d$ défini par*

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

Cette définition correspond à la définition usuelle de l'enveloppe convexe (*convex hull* en anglais, d'où la notation) d'un ensemble fini de $\mathbb{R}^d$.

Définition 1.2.2 (Enveloppe convexe discrète)
Étant donné un voisinage V en dimension d, on appelle enveloppe convexe discrète *de V l'ensemble*

$$\mathrm{CH}(V) = \mathrm{CHC}(V) \cap \mathbb{Z}^d$$

Nous verrons que la forme de l'enveloppe convexe discrète d'un voisinage V a une grande incidence sur les propriétés de V. Par la suite, nous dirons simplement que $\mathrm{CH}(V) \subseteq \mathbb{Z}^d$ est l'enveloppe convexe de V.

Remarque. La notion d'enveloppe convexe d'un voisinage n'a que très peu d'intérêt en dimension 1 puisqu'elles sont toutes réduites à un segment :

$$\mathrm{CH}(V) = [\![\min V, \max V]\!]$$

La proposition suivante est un résultat bien connu d'algèbre linéaire :

Proposition 1.2.1
En dimension $d \in \mathbb{N}$, l'enveloppe convexe continue d'un ensemble fini $E \subseteq \mathbb{R}^d$ est un polyèdre convexe dont les sommets appartiennent à E. Soit $\{s_1, \ldots, s_p\} \subseteq E$ l'ensemble des sommets de $\mathrm{CHC}(E)$.

Si tous les points de E sont de coordonnées rationnelles, pour tout point $x \in \mathrm{CHC}(E)$ de coordonnées rationnelles, il existe des coefficients $(\lambda_1, \ldots, \lambda_p)$ rationnels dans $[0,1]$ tels que

$$x = \sum_{i=1}^{p} \lambda_i . s_i \quad et \quad \sum_{i=1}^{p} \lambda_i = 1$$

Cela implique notamment que pour tout voisinage V en dimension d, les sommets de $\mathrm{CHC}(V)$ appartiennent à V et que pour tout vecteur x de coordonnées rationnelles dans $\mathrm{CHC}(V)$ il existe un entier k tel que $k.x$ s'écrive comme une somme de k sommets de $\mathrm{CHC}(V)$:

Ceci nous servira par la suite puisque pour tout vecteur $x \in \mathbb{Q}^d \cap \mathrm{CHC}(V)$ il est donc possible de déplacer une information selon une direction qui correspond asymptotiquement au vecteur $(-x)$, au sens où en k générations il est possible de déplacer une information selon le vecteur $k(-x)$ (soulignons ici le fait que l'information se déplace dans le sens inverse du voisinage puisque si une cellule c peut voir l'état de sa voisine $(c+v)$ alors l'information portée par $(c+v)$ peut se déplacer jusqu'à c en un temps).

1.2.2 Complétude

Définition 1.2.3 (Complétude)
Soit V un voisinage en dimension d. On dira que V est

- semi-complet *si* $\mathbb{N}^d \subseteq V^\infty$;
- complet *si* $V^\infty = \mathbb{Z}^d$;
- semi-incomplet *s'il est semi-complet mais pas complet.*

On a la caractérisation suivante :

Proposition 1.2.2
Un voisinage V en dimension d est complet si et seulement si le $\mathbb{Z}$-module engendré par V est $\mathbb{Z}^d$ tout entier et que l'enveloppe convexe continue de V contient un voisinage de l'origine.

Preuve : Cette caractérisation est présentée dans [3]. Nous en donnerons ici une preuve différente.

Montrons cette proposition par double implication.

Conditions nécessaires. Puisque V^∞ est inclus dans le $\mathbb{Z}$-module engendré par V, il est nécessaire que ce dernier contienne $\mathbb{Z}^d$ tout entier.

Pour ce qui est de l'enveloppe convexe de V, si l'on suppose qu'elle ne contient aucun voisinage de l'origine, alors c'est que l'une de ses faces (l'enveloppe convexe d'un ensemble fini est un polyèdre) rencontre l'origine puisque l'origine appartient à V et donc à son enveloppe convexe. L'hyperplan sur lequel se trouve cette face passe par l'origine et sépare $\mathbb{R}^d$ en deux demi-espaces, l'enveloppe convexe de V se trouvant alors entièrement dans l'un des deux demi-espaces. Cela signifie qu'aucune combinaison linéaire à coefficients dans $[0,1]$ et de somme 1 des vecteurs de V n'est dans l'autre demi-espace. On en déduit donc qu'aucune combinaison linéaire des vecteurs de

V à coefficients dans $\mathbb{R}^+$ ne se trouve dans le second demi-espace, V ne peut donc pas être complet et la seconde condition est donc bien nécessaire.

Conditions suffisantes. Afin de simplifier les notations la preuve sera effectuée en dimension 2 mais elle est tout à fait similaire en dimension d quelconque. Soit $V = \{(0,0), v_1 = (x_1, y_1), \ldots, v_p = (x_p, y_p)\}$. On suppose que V vérifie les deux conditions énoncées précédemment.

Montrons dans un premier temps qu'il existe une cellule dans V^∞ dont les deux coordonnées sont strictement positives.

Puisque l'enveloppe convexe des vecteurs de V contient un voisinage de l'origine et par densité de $\mathbb{Q}$ dans $\mathbb{R}$ il existe un vecteur

$$u \in \mathrm{CHC}(V) \cap (\mathbb{Q}^{+*})^2$$

Ce vecteur s'écrit donc sous la forme

$$u = \sum_{i \in [\![1,p]\!]} \alpha_i v_i$$

avec $\sum_i \alpha_i = 1$ et $\forall i, \alpha_i \in \mathbb{Q}^+$ (proposition 1.2.1). Ainsi, en multipliant tous les α_i par le PPCM de leurs dénominateurs on obtient un vecteur $u_1 \in (\mathbb{N}^*)^2$ qui s'écrit comme combinaison linéaire à coefficients dans $\mathbb{N}$ des éléments de V. On montre de même l'existence de vecteurs $u_2 \in (\mathbb{N}^*) \times (-\mathbb{N}^*)$, $u_3 \in (-\mathbb{N}^*) \times (\mathbb{N}^*)$ et $u_4 \in (-\mathbb{N}^*)^2$ appartenant à V^∞ (voir figure 1.1).

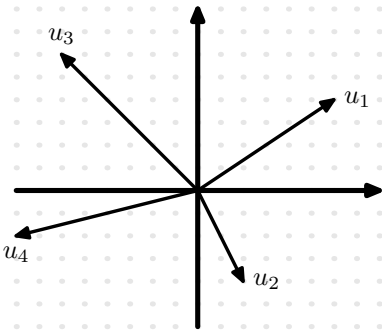

FIGURE 1.1 – Les quarts de plan et les vecteurs u_i.

Il reste maintenant à montrer que toute cellule de $\mathbb{Z}^2$ s'écrit comme somme finie d'éléments de V. Puisque V engendre $\mathbb{Z}^2$ (en tant que $\mathbb{Z}$-module), on sait qu'il existe des coefficients $(\lambda_i)_{i \in [\![1,p]\!]} \in \mathbb{Z}^p$ tels que

$$\sum_{i=1}^{p} \lambda_i v_i = (1,0)$$

et donc en séparant les coefficients positifs des négatifs, on peut écrire

$$\sum_{i=1}^{p} \beta_i v_i = (1,0) + \sum_{i=1}^{p} \alpha_i v_i = (1,0) + A$$

(où $\forall i, \alpha_i \geq 0$ et $\beta_i \geq 0$). Cela signifie qu'il existe deux cellules adjacentes ($A = \sum \alpha_i v_i$ et $A + (1,0)$) qui sont toutes deux dans V^∞ (figure 1.2). On

va maintenant montrer qu'il existe deux cellules adjacentes X et $X + (1, 0)$ placées sur l'axe des abscisses appartenant à V^∞. On suppose que A n'est pas sur l'axe des abscisses (sinon on prend $X = A$) et que A a une ordonnée positive y_A (le cas d'une ordonnée négative est similaire par symétrie et l'on utilisera alors $u_1$au lieu de u_2 dans la suite). On considère maintenant le vecteur u_2 dont on a montré l'existence précédemment. Soit y_{u_2} son ordonnée (on a $y_{u_2} < 0$ par définition de u_2). Prenons alors

$$X = (-y_{u_2}).A + y_A.u_2$$

L'ordonnée de X est $(-y_{u_2}).y_A + y_A.y_{u_2} = 0$ et donc X est bien sur l'axe des abscisses. De plus X appartient à V^∞ puisqu'il s'écrit comme somme finie d'éléments de V^∞ (tous les coefficients devant les vecteurs sont finis et positifs). Enfin on a

$$X + (1, 0) = [A + (1, 0)] + (-y_{u_2} - 1).A + y_A.u_2$$

et il s'écrit donc comme somme finie d'éléments de V^∞ (voir figure 1.3).

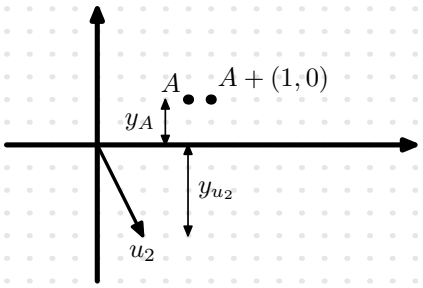

FIGURE 1.2 – Les cellules A et $A + (1, 0)$.

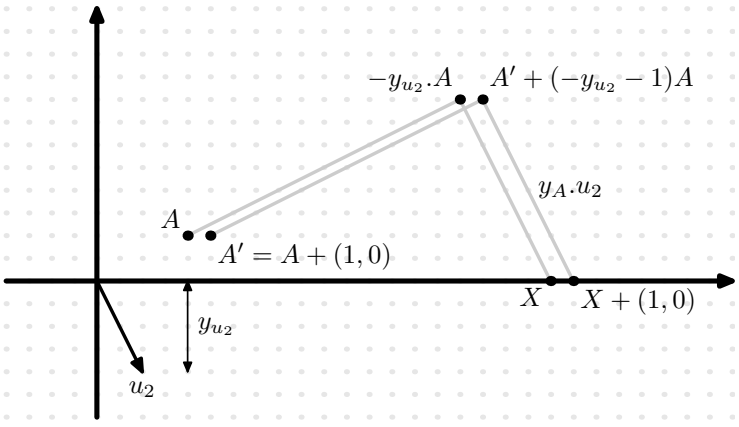

FIGURE 1.3 – Construction de X et $X + (1, 0)$.

Notons $X = (x, 0)$ et raisonnons maintenant sur l'axe des abscisses. Pour alléger les notations nous noterons n la cellule $(n, 0)$. Nous avons montré qu'il existe un entier x tel que x et $(x + 1)$ appartiennent tous deux à V^∞.

Ainsi, $2x = x + x$, $2x + 1 = x + (x + 1)$ et $(2x + 2) = (x + 1) + (x + 1)$ appartiennent tous trois à V et de manière générale, pour tout entier naturel k et tout $i \in [\![0, k]\!]$ on a

$$kx + i = (k - i)x + i(x + 1) \in V^\infty$$

ce qui signifie que le segment $[\![kx, k(x + 1)]\!]$ est inclus dans V^∞. Pour $k \geq (x - 1)$, les segments $[\![kx, kx + k]\!]$ et $[\![(k + 1)x, (k + 1)(x + 1)]\!]$ se rejoignent, et donc tout le demi-axe $[\![x(x - 1), \infty[\![$ est inclus dans V^∞.

On montre de même qu'il existe un entier $x' \in (-\mathbb{N})$ tel que le demi-axe $]\!] - \infty, x'(x' - 1)]\!]$ soit inclus dans V^∞. On a alors

$$[\![x(x-1), \infty[\![\quad + \quad]\!] - \infty, x'(x' - 1)]\!] \quad = \quad \mathbb{Z} \quad \subseteq \quad V^\infty$$

On a donc montré que V^∞ contient tout l'axe des abscisses. On montre de manière similaire que l'axe des ordonnées est inclus dans V^∞ et donc que $\mathbb{Z}^2 \subseteq V^\infty$. Le voisinage V est donc complet. □

Remarque. Les voisinages semi-complets sont ceux sur lesquels un automate cellulaire peut « raisonnablement » reconnaître un langage puisque l'origine peut recevoir de l'information de toute lettre du mot en entrée (dans le cas des voisinages qui ne sont pas semi-complets, il existe des positions telles que les lettres du mot en entrée sur ces positions ne peuvent pas influencer la réponse donnée par la cellule origine).

Remarque. Les voisinages semi-incomplets correspondent à la notion d'automate cellulaire *à sens unique* ou *one-way*. En effet, nous venons de montrer que ces voisinages n'avaient aucun élément négatif ce qui implique que les informations ne peuvent pas se déplacer vers la droite. Ils sont cependant semi-complets ce qui signifie qu'une cellule peut transmettre son information à n'importe quelle cellule qui se trouve sur sa gauche.

1.2.3 Quelques exemples de voisinages

Parmi tous les voisinages possibles en toute dimension, certains ont été beaucoup plus étudiés que d'autres. Remarquons en particulier les voisinages suivants :

Le voisinage standard en dimension 1

Parfois appelé voisinage « usuel » ou « classique », il est défini par

$$V_{\text{std}} = \{-1, 0, 1\}$$

C'est de loin le voisinage le plus étudié car c'est le plus petit voisinage permettant de transmettre des informations dans les deux directions. Il arrive parfois que l'on parle de voisinage standard de rayon r pour un entier $r \geq 1$. Il s'agit alors du voisinage

$$V_{\text{std}(r)} = [\![-r, r]\!]$$

ce qui permet d'augmenter la vitesse de déplacement de l'information puisque chaque cellule voit alors r fois plus loin que dans le cas du voisinage de rayon 1. Tout automate cellulaire en dimension 1 peut être vu comme un automate cellulaire fonctionnant sur le voisinage standard de rayon r pour un certain r.

Le voisinage *one-way* en dimension 1

Introduit pour la première fois par C. R. Dyer dans [7] puis par K. Čulik et C. Choffrut dans [2], il est défini par

$$V_{1w} = \{0, 1\}$$

ce voisinage est le plus petit voisinage permettant de définir correctement la reconnaissance de langages (c'est également le plus petit voisinage capable d'effectuer un calcul de quelque sorte que ce soit puisqu'un voisinage n'ayant qu'une cellule ne permet pas de « combiner » d'informations). Ce voisinage a été très étudié car il permet de modéliser des machines de calcul à une dimension dans lesquelles l'information ne se déplace que dans une seule direction (d'où le nom). En effet, si la cellule c peut voir l'état de la cellule $(c + 1)$ et donc utiliser cette information pour modifier son propre état, le comportement de la cellule $(c + 1)$ ne peut pas être affecté par l'état de sa voisine de gauche (donc l'information se propage vers la gauche, c'est-à-dire de $(c + 1)$ vers c).

Il existe de nombreux résultats et de nombreuses questions ouvertes (voir [10, 7]) visant à déterminer l'impact de cette restriction sur la capacité de calcul des machines concernées par rapport aux automates fonctionnant sur des voisinages *two-ways* tels que le voisinage standard. Ainsi, on ignore aujourd'hui si la classe des langages reconnus en temps linéaire par des automates fonctionnant sur le voisinage *one-way* est égale à la classe des langages reconnus en temps linéaire par des automates fonctionnant sur le voisinage usuel.

Le voisinage de Von Neumann

S'il est vrai que le voisinage standard en dimension 1 s'impose de manière évidente comme le plus naturel des voisinages, il n'en va pas de même en dimension 2 et au-delà. Ainsi, le voisinage de Von Neumann en dimension d défini par

$$V_{\mathrm{VN}} = \{(x_1, \ldots, x_d) \mid \sum_i |x_i| \leq 1\}$$

correspond à la boule unité pour la norme 1. En dimension 2 (cas que nous considérerons le plus fréquemment) le voisinage de Von Neumann d'une cellule est donc l'ensemble de la cellule et de ses 4 voisines les plus proches : en haut, à droite, en bas et à gauche.

Le voisinage de Von Neumann est petit, symétrique (selon chacun des deux axes et par symétrie centrale). C'est l'un des deux voisinages les plus étudiés en dimension 2.

Le voisinage de Moore

Le voisinage de Moore est le deuxième voisinage couramment considéré en dimension 2, et probablement le plus utilisé pour les dimensions supérieures à 2. Il est défini par

$$V_{\mathrm{M}} = \{(x_1, \ldots, x_d) \mid \max_i(|x_i|) \leq 1\}$$

et correspond à la boule de rayon 1 pour la norme infinie. En dimension 2 le voisinage de Moore d'une cellule est donc constitué de la cellule elle-même ainsi

que de ses 8 plus proches voisines, les quatre incluses dans le voisinage de Von Neumann ainsi que les 4 « diagonales ».
Remarque. En dimension 1, le voisinage de Moore et le voisinage de Von Neumann coïncident tous deux avec le voisinage standard.
Remarque. Chacun des deux voisinages précédents peut être restreint à $\mathbb{N}^d$ pour obtenir une version *one-way*. Par exemple en dimension 2, le voisinage de Von Neumann one-way est l'ensemble $\{(0,0),(1,0),(0,1)\}$ et c'est un voisinage selon lequel les informations peuvent se déplacer selon les abscisses et les ordonnées décroissantes mais pas dans le sens inverse (voir [15]).

Le voisinage minimal en dimension 2

Beaucoup moins étudié que les voisinages de Moore et de Von Neumann, le voisinage minimal présente cependant quelques intérêts théoriques. Il est défini par

$$V_{\min} = \{(0,0),(1,0),(0,1),(-1,-1)\}$$

C'est un plus petit voisinage bidimensionnel tel que toute cellule du plan puisse faire parvenir de l'information à l'origine, ce qui implique notamment qu'il n'est pas *one-way*. Le manque de symétrie le rend cependant plus complexe à manipuler que les voisinages définis précédemment, mais l'étude des automates cellulaires fonctionnant sur de tels voisinages permet de comprendre les véritables implications de la symétrie.

1.3 Reconnaissance de langages

Dans toute cette section, Σ désigne un ensemble fini que nous appellerons *alphabet*.

Définition 1.3.1 (Mots unidimensionnels)
Pour tout entier naturel n, on définit l'ensemble des mots unidimensionnels de longueur n sur Σ par

$$\Sigma^n = \Sigma^{[\![0,n-1]\!]}$$

L'ensemble des mots unidimensionnels sur Σ est alors

$$\Sigma^* = \bigcup_{i\in\mathbb{N}} \Sigma^i$$

Cette définition correspond à la notion usuelle de mot sur un alphabet fini, mais nous choisissons ici d'en donner une caractérisation « fonctionnelle » au lieu de la construction inductive habituelle car elle est plus facile à généraliser aux dimensions supérieures.

Définition 1.3.2 (Mots bidimensionnels)
Pour tout couple d'entiers $(n,m) \in \mathbb{N}$, on définit l'ensemble des mots bidimensionnels *(ou* images*) de taille (n,m) sur Σ par*

$$\Sigma^{n\times m} = \Sigma^{[\![0,n-1]\!]\times[\![0,m-1]\!]}$$

L'ensemble des mots bidimensionnels sur Σ est alors

$$\Sigma^{**} = \bigcup_{i,j\in\mathbb{N}} \Sigma^{i\times j}$$

Cette définition correspond bien à l'idée naturelle que l'on se fait d'une image discrète rectangulaire. Si une image est de taille (n, m) on dira qu'elle est de longueur n et de hauteur m.

En pratique, dans tout le document, nous ne nous intéresserons qu'à des mots de dimension 1 ou 2. Cependant la plupart des résultats que nous obtiendrons sur les mots de dimension 2 sont facilement généralisables aux mots de dimensions supérieures (dans ce cas le résultat sera donné en dimension quelconque mais la preuve sera faite en dimension 2 afin d'alléger les notations). Dans un souci de rigueur, il est donc nécessaire de définir les mots en dimension quelconque :

Définition 1.3.3 (Mots de dimension d)
Soit $d \geq 1$ un entier. Pour tout $(n_1, n_2, \ldots, n_d) \in \mathbb{N}^d$, on définit l'ensemble des mots de taille $(n_1, n_2, \ldots, n_d)$ sur Σ par

$$\Sigma^{n_1 \times \ldots \times n_d} = \Sigma^{[\![0, n_1 - 1]\!] \times \ldots \times [\![0, n_d - 1]\!]}$$

L'ensemble des mots de dimension d sur Σ est alors

$$\Sigma^{*^d} = \bigcup_{i_1, \ldots, i_d \in \mathbb{N}} \Sigma^{i_1 \times \ldots \times i_d}$$

Pour toute dimension d un langage est un ensemble de mots. Ainsi un langage unidimensionnel est une partie de Σ^* tandis qu'un langage bidimensionnel est une partie de Σ^{**}.

Afin de reconnaître des langages à l'aide d'automates cellulaires il est nécessaire d'enrichir légèrement la définition :

Définition 1.3.4 (Automate de reconnaissance)
Un automate cellulaire de reconnaissance sur l'alphabet Σ est un nonuplet

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, \mathcal{Q}_{\text{acc}}, q_a, q_r)$$

où :

- *d, $\mathcal{Q}$, δ et V ont la même signification que pour un automate cellulaire classique ;*
- *$\Sigma \subseteq \mathcal{Q}$ est un sous-ensemble d'états qui sera utilisé pour encoder le mot dans la configuration initiale de l'automate ;*
- *$B \in \mathcal{Q}$ est un état particulier n'appartenant pas à Σ appelé état* quiescent *ou état* blanc *de l'automate. Il vérifie $\delta(B, B, \ldots, B) = B$;*
- *$\mathcal{Q}_{\text{acc}} \subseteq \mathcal{Q}$ est un sous-ensemble d'états appelé ensemble d'*états acceptants *;*
- *$q_a \in \mathcal{Q}_{\text{acc}}$ et $q_r \in \mathcal{Q} \setminus \mathcal{Q}_{\text{acc}}$ sont deux états persistants c'est-à-dire que si une cellule est dans l'un de ces états au temps t elle y reste au temps $(t + 1)$ quels que soient les états de ses voisines.*

Soit w un mot de taille $(n_1, \ldots, n_d)$ en dimension d et

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, \mathcal{Q}_{\text{acc}}, q_a, q_r)$$

un automate cellulaire de reconnaissance sur Σ, on appelle configuration initiale associée à w la configuration

$$\mathfrak{C}_w : \begin{cases} \mathbb{Z}^d & \to & \mathcal{Q} \\ (x_1, \ldots, x_d) & \mapsto & w(x_1, \ldots, x_d) & \text{si } \forall i, 0 \leq x_i < n_i \\ (x_1, \ldots, x_d) & \mapsto & B & \text{sinon} \end{cases}$$

En d'autres termes, $\mathfrak{C}_w$ est simplement la configuration qui coïncide avec le mot w sur toutes les cases où celui-ci est défini et contient l'état blanc B partout ailleurs.

On peut alors définir l'acceptation et le refus d'un mot par un automate cellulaire de reconnaissance :

Définition 1.3.5 (Acceptation d'un mot)
On dira qu'un automate cellulaire de reconnaissance

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, \mathcal{Q}_{\text{acc}}, q_a, q_r)$$

reconnaît (ou accepte) le mot $w \in \Sigma^{*^d}$ en temps $t \in \mathbb{N}$ et espace

$$(s_1, \ldots, s_d) \in \mathbb{N}^d$$

si

$$\begin{cases} \mathcal{A}^t(\mathfrak{C}_w)(0) & \in & \mathcal{Q}_{\text{acc}} \\ \mathcal{A}^{t+1}(\mathfrak{C}_w)(0) & = & q_a \end{cases}$$

et

$$\forall t' \in [\![0, t]\!], \forall (c_1, \ldots, c_d) \notin [\![0, s_1 - 1]\!] \times \ldots \times [\![0, s_d - 1]\!],$$
$$\mathcal{A}^{t'}(\mathfrak{C}_w)(c_1, \ldots, c_d) = B$$

Définition 1.3.6 (Refus d'un mot)
On dira qu'un automate cellulaire de reconnaissance

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, \mathcal{Q}_{\text{acc}}, q_a, q_r)$$

refuse le mot $w \in \Sigma^{*^d}$ en temps $t \in \mathbb{N}$ et espace

$$(s_1, \ldots, s_d) \in \mathbb{N}^d$$

si

$$\begin{cases} \mathcal{A}^t(\mathfrak{C}_w)(0) & \notin & \mathcal{Q}_{\text{acc}} \\ \mathcal{A}^{t+1}(\mathfrak{C}_w)(0) & = & q_r \end{cases}$$

et

$$\forall t' \in [\![0, t]\!], \forall (c_1, \ldots, c_d) \notin [\![0, s_1 - 1]\!] \times \ldots \times [\![0, s_d - 1]\!],$$
$$\mathcal{A}^{t'}(\mathfrak{C}_w)(c_1, \ldots, c_d) = B$$

La définition formelle précédente signifie que l'on « écrit » le mot sur l'automate au temps 0 (c'est la configuration initiale $\mathfrak{C}_w$) puis que l'automate évolue selon la règle de transition globale pendant un certain temps t après lequel l'automate est donc dans la configuration $\mathcal{A}^t(\mathfrak{C}_w)$.

On dit alors que l'automate a reconnu le mot w si la cellule origine $0 = (0, 0, \ldots, 0)$ est dans un état d'acceptation $q \in \mathcal{Q}_{\text{acc}}$ et qu'elle passera au temps suivant dans l'état q_a dans lequel elle restera indéfiniment par la suite.

La définition de refus est similaire puisque l'on demande que l'origine ne soit pas dans un état acceptant au temps t et qu'elle passe au temps $(t+1)$ dans l'état persistant q_r.

La complexité en espace correspond aux cellules qui participent au calcul, c'est-à-dire celles qui prennent un état autre que B au cours de la reconnaissance. On remarque qu'avec une telle définition il n'est pas possible de reconnaître un mot de taille $(n_1, \ldots, n_d)$ en espace $(s_1, \ldots, s_d)$ si l'on a $s_i < n_i$ pour un certain i. Nous ne nous intéresserons cependant que peu à la complexité en espace et cette définition sera donc bien suffisante.

Remarque. Il peut sembler inutilement compliqué d'avoir à la fois un ensemble d'états $\mathcal{Q}_{\text{acc}}$ et les deux états q_a et q_r. Toutefois nous verrons par la suite (section 1.6) que cette définition sera la plus facile à manipuler car elle est à la fois adaptée à la définition du temps réel (qui sera la classe de complexité que nous étudierons le plus) et aux classes de plus grande complexité (temps linéaire, polynomial, etc.). Nous justifierons donc cette définition après avoir introduit les classes de faible complexité.

On peut alors définir le langage reconnu par un automate cellulaire :

Définition 1.3.7
On dira qu'un automate cellulaire de reconnaissance

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, \mathcal{Q}_{\text{acc}}, q_a, q_r)$$

reconnaît le langage $L \subseteq \Sigma^{*^d}$ *en temps* $T : \mathbb{N}^d \to \mathbb{N}$ *et en espace* $S : \mathbb{N}^d \to \mathbb{N}^d$ *si* $\mathcal{A}$ *accepte tout mot* $w \in L$ *de taille* $(n_1, \ldots, n_d)$ *en temps* $T(n_1, \ldots, n_d)$ *et espace* $S(n_1, \ldots, n_d)$ *et refuse tout mot* $w \in \Sigma^{*^d} \setminus L$ *de taille* $(n_1, \ldots, n_d)$ *en temps* $T(n_1, \ldots, n_d)$ *et espace* $S(n_1, \ldots, n_d)$.

Cette définition correspond à la notion naturelle de reconnaissance (similaire à ce qu'elle est pour les autres modèles de calcul). La fonction de temps $T : \mathbb{N}^d \to \mathbb{N}$ permet de représenter la façon dont varie le temps mis par l'automate à reconnaître si un mot appartient ou non à L en fonction de la taille du mot en entrée.

Dans la suite du document, nous ne différencierons plus les automates cellulaires de reconnaissance des automates cellulaires simples. En effet, lorsque l'on dira d'un automate qu'il reconnaît un langage ou lorsque nous en construirons un il sera sous-entendu que l'alphabet Σ est inclus dans les états de l'automate ainsi que l'état particulier B (pour que la configuration initiale ait un sens). Enfin nous ne spécifierons pas l'ensemble d'états acceptants dans la définition de l'automate afin d'alléger les notations. Lorsque nous décrirons un automate qui reconnaît un langage le choix des états acceptants sera en général évident.

Avant de définir les classes de complexité en temps qui nous intéresseront par la suite (les classes de très basse complexité et principalement la classe *temps réel*) il est nécessaire de revenir sur le fonctionnement général d'un automate cellulaire.

1.4 Concepts élémentaires

Cette section concerne les automates cellulaires en général, principalement vus comme des modèles de calcul. Les techniques et idées exposées ici ne sont pas

restreintes à la reconnaissance de langages, mais elles seront très utilisées tout au long de ce document.

1.4.1 Diagramme espace-temps

Étant donnés un automate cellulaire $\mathcal{A} = \{1, \mathcal{Q}, \delta, V\}$ en dimension 1 et une configuration $\mathfrak{C} : \mathbb{Z} \to \mathcal{Q}$.

On peut représenter graphiquement la configuration $\mathfrak{C}$ par une ligne infinie de carrés, chaque carré (correspondant à une cellule de $\mathbb{Z}$) étant colorié en fonction de son état. Si par exemple on choisit un automate à deux états ($\mathcal{Q} = \{0, 1\}$) et que l'on représente l'état 1 par du noir et l'état 0 par du blanc, la configuration

$$\mathfrak{C}_0 : \begin{cases} \mathbb{Z} & \to & \{0,1\} & \\ c & \mapsto & 1 & \text{si } \exists k, c = \pm 2^k \\ c & \mapsto & 0 & \text{sinon} \end{cases}$$

peut être représentée par la figure 1.4 (bien évidement, on ne représente qu'un segment de la configuration).

FIGURE 1.4 – Représentation de la configuration $\mathfrak{C}_0$ des puissances de 2.

Nous avons vu par ailleurs que l'évolution d'un automate était entièrement déterminée par la configuration initiale et la règle de transition (locale ou globale). Supposons par exemple que l'on utilise le voisinage $V = \{-1, 0, 1\}$ (canoniquement ordonné en $(-1, 0, 1)$), c'est-à-dire qu'une cellule calcule son nouvel état en fonction de son propre état courant et de ceux de sa voisine de droite et de sa voisine de gauche, et que l'on définisse la règle locale de notre automate $\mathcal{A}$ pour qu'elle corresponde à l'opérateur XOR, c'est-à-dire

$\forall q_1, q_2, q_3 \in \{0, 1\}$,

$$\begin{array}{rcll} \delta(q_1, q_2, q_3) & = & 0 & \text{si 0 ou 2 des états } q_1,\ q_2 \text{ et } q_3 \text{ sont 1} \\ & = & 1 & \text{sinon} \end{array}$$

La configuration suivante est alors obtenue naturellement en appliquant la règle locale à chaque cellule. On obtient la configuration représentée sur la figure 1.5.

FIGURE 1.5 – La configuration successeur de $\mathfrak{C}_0$ dans le cas de l'automate XOR.

Si l'on continue à calculer les configurations suivantes, il est possible d'en représenter une séquence en les disposant l'une au-dessus de l'autre comme illustré sur la figure 1.6. On utilise la convention usuelle d'orientation du temps vers le haut (bien que contraire à l'habitude naturelle d'écrire de haut en bas), c'est-à-dire que la configuration au temps 0 est dessinée en bas, puis immédiatement dessus la configuration au temps 1 et ainsi de suite.

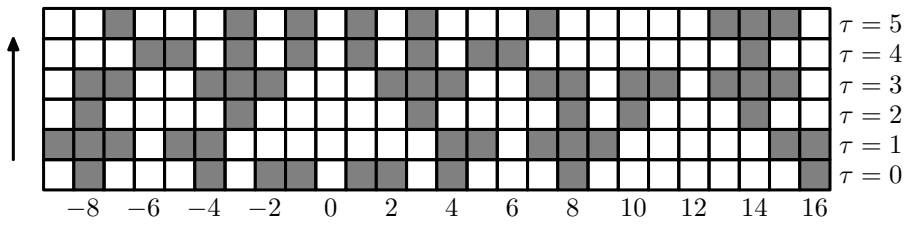

FIGURE 1.6 – Un diagramme espace-temps.

Les états successifs d'une même cellule sont ainsi représentés par une colonne (par exemple, on voit sur la figure 1.6 que la cellule 2 passe successivement par les états 1, 0, 0, 1, 0 et 0).

Idéalement, le diagramme espace-temps est une image infinie (tant en largeur qu'en hauteur) ce qui permet de représenter la totalité de l'évolution de l'automate. En pratique nous ne représenterons bien évidemment qu'une portion limitée dans l'espace et le temps.

Enfin, notons qu'il est également possible de réaliser des diagrammes espace-temps pour les automates en dimension 2 mais le diagramme est alors en dimension 3 (on superpose des configurations qui sont alors des plans selon une nouvelle dimension) et donc bien moins facile à manipuler. L'objet peut également être défini pour les dimensions supérieures à 2 mais la représentation n'est alors plus possible, et l'intérêt en est donc très diminué.

1.4.2 Vitesse maximale de propagation et cônes d'influence

Considérons un automate cellulaire $\mathcal{A} = (d, \mathcal{Q}, \delta, V)$. Si l'on se place sur la cellule $c \in \mathbb{Z}^d$ au temps t, la définition de la règle de transition de l'automate indique que l'état suivant de c (au temps $t+1$) dépend uniquement des états courants des cellules dans l'ensemble

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

et ce pour toute cellule c.

Au temps $t+1$ la situation est très similaire : l'état suivant de la cellule c (donc au temps $t+2$) dépend des états courants (au temps $t+1$) des cellules de $V(c)$. Or l'état au temps $t+1$ d'une cellule c' de $V(c)$ dépendait uniquement des états au temps t des cellules dans $V(c')$.

Ainsi, l'état de la cellule c au temps $(t+2)$ dépend uniquement de l'état au temps $(t+1)$ des cellules de

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

ces états eux-mêmes ne dépendant que des états présents au temps t sur les cellules de

$$\begin{aligned} \bigcup_{c' \in V(c)} V(c') &= \bigcup_{c' \in V(c)} \{c' + v \mid v \in V\} \\ &= \{c + v + v' \mid v, v' \in V\} \quad = \quad V^2(c) \end{aligned}$$

Par une induction facile on montre donc que pour tout $k \in \mathbb{N}$ l'état de la cellule c au temps $(t+k)$ ne dépend que de l'état au temps t des cellules dans

$$V^k(c) = \{c + v_1 + v_2 + \ldots + v_k \mid v_1, \ldots, v_k \in V\}$$

En dimension 1 ce phénomène se représente très bien à l'aide des diagrammes espace-temps décrits précédemment. En effet, si l'on considère par exemple un automate cellulaire dont le voisinage est $V = \{-1, 0, 1, 2\}$, la figure 1.7 illustre les états qui peuvent avoir une influence sur l'état de la cellule 0 au temps $t+3$ à chaque étape de temps (ceux qui ne sont pas hachurés).

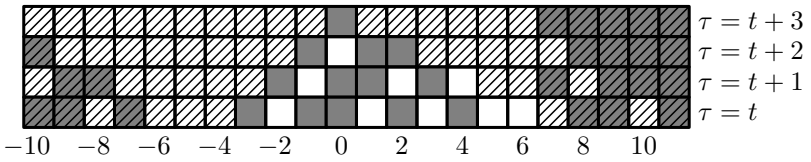

FIGURE 1.7 – Le cône des états qui ont une influence sur la cellule 0 au temps $t+3$.

La disposition de ces cellules ne dépend que du voisinage de l'automate, et l'on peut donc affirmer sans même connaître la règle de transition de l'automate $\mathcal{A}$ que la connaissance des états des cellules de -3 à 6 au temps t est suffisante pour déterminer avec exactitude l'état de la cellule 0 au temps $(t+3)$ (pour déterminer effectivement cet état il faut alors connaître la règle δ).

Sur un diagramme espace-temps ces cellules apparaissent disposées sur un cône (cela inclut aussi les diagrammes espace-temps en dimensions supérieures à 1). Ce cône a une certaine pente selon chaque direction (en dimension 1 il n'y a que deux directions, mais en dimension 2 et au-delà il y en a plus). Cette pente représente l'inverse de la vitesse maximale à laquelle une information peut se déplacer à travers l'espace. En effet, si l'on regarde l'exemple de la figure 1.7, la pente est de 1 sur la gauche du cône tandis qu'elle est de 1/2 sur la droite. Or nous avons vu que pour qu'une cellule puisse influencer l'origine en k étapes, il faut qu'elle soit à moins de k cellules sur sa gauche ou à moins de $2k$ cellules sur sa droite, ce qui peut être vu inversement en considérant que pour qu'une information contenue sur une cellule puisse atteindre l'origine en k étapes il faut qu'elle ait moins de k cellules à parcourir vers la droite ou moins de $2k$ cellules à parcourir vers la gauche.

En dimension 1 cette vitesse maximale est exactement le plus grand vecteur du voisinage dans chacune des directions (remarquons une fois encore que l'information se déplace dans le sens contraire du voisinage, c'est-à-dire que si V contient un très grand vecteur positif toute cellule peut voir loin sur sa droite et donc l'information se déplace rapidement vers la gauche).

En dimensions supérieures la vitesse maximale de déplacement de l'information dans une direction donnée dépend de l'enveloppe convexe du voisinage.

Par la suite, nous nous intéresserons beaucoup à la reconnaissance de langages et puisque la réponse du calcul est donnée par la cellule origine on s'intéressera au temps qu'il faut pour que l'état d'une cellule donnée puisse affecter l'état de l'origine. On pose donc la définition suivante :

Définition 1.4.1 (Rang)
En dimension d, étant donné un voisinage V. On définit la fonction de rang *associée à V par*

$$\mathrm{rg} : \begin{cases} \mathbb{Z}^d & \to & \mathbb{N} \cup \{\infty\} \\ c & \mapsto & \infty \quad si\ c \notin V^\infty \\ c & \mapsto & \min\{t \in \mathbb{N} \mid c \in V^t\} \end{cases}$$

Ainsi le rang de l'origine est 0, et de manière générale si le rang d'une cellule c est r, il faut attendre r générations de l'automate pour que l'état porté par la cellule c puisse avoir une incidence sur l'état de l'origine.

1.4.3 Produit cartésien et construction par couches

La définition d'un automate cellulaire exige que le nombre d'états soit fini. Ceci est nécessaire si l'on veut pouvoir définir extensivement toutes les transitions possibles de manière finie. C'est également une idée plus « réaliste » qui considère qu'une cellule de l'automate ne peut avoir qu'une mémoire finie, et c'est en augmentant le nombre de cellules porteuses d'information que l'on peut augmenter la taille de l'information globale portée par l'automate (exactement comme on le fait sur le ruban d'une machine de Turing). Cependant ce nombre d'état peut être arbitrairement grand.

Il peut être intéressant de chercher des automates ayant très peu d'états et il existe de nombreuses questions ouvertes de la forme : « quel est le nombre minimal d'états nécessaires pour qu'un automate cellulaire ait la propriété P ». Cependant nous négligerons totalement cet aspect dans ce document et nous nous contenterons de vérifier que les automates que l'on construit n'ont bien qu'un nombre fini d'états.

Sans limitation sur le nombre d'états, il est alors possible de « combiner » plusieurs automates en un seul, ce qui nous permettra de séparer les tâches à effectuer avant de produire un unique automate pour les réaliser toutes[1].

Supposons par exemple que l'on ait deux automates de même dimension et de même voisinage $\mathcal{A}_1 = (d, \mathcal{Q}_1, \delta_1, V)$ et $\mathcal{A}_2 = (d, \mathcal{Q}_2, \delta_2, V)$, et l'on suppose de plus que les ensembles $\mathcal{Q}_1$ et $\mathcal{Q}_2$ sont disjoints, ce que l'on peut toujours obtenir quitte à renommer des états. On peut alors définir un « automate produit » $\mathcal{A}_{1\times 2} = (d, \mathcal{Q}_1 \times \mathcal{Q}_2, \delta_{1\times 2}, V)$ où la règle locale $\delta_{1\times 2}$ est définie par :
$\forall q_1, \ldots, q_s \in \mathcal{Q}_1, \forall q'_1, \ldots q'_s \in \mathcal{Q}_2$,

$$\delta_{1\times 2}((q_1, q'_1), \ldots, (q_s, q'_s)) = (\delta_1(q_1, \ldots, q_s), \delta_2(q'_1, \ldots, q'_s))$$

(c'est la définition naturelle du produit, similaire au produit de deux machines de Turing ou au produit de deux automates finis).

Cependant cet automate produit tel quel n'a que très peu d'intérêt en termes de possibilités de calcul (puisqu'au fond il ne fait que calculer exactement ce que faisaient les deux automates de départ). On peut alors modifier la fonction $\delta_{1\times 2}$ pour que chacun des deux automates puisse tenir compte des états de l'autre qui se trouvent sur les cellules.

Ainsi, il est par exemple possible de maintenir le fonctionnement de $\mathcal{A}_1$ sans qu'il soit affecté par le fonctionnement de $\mathcal{A}_2$, tout en modifiant les règles sur

[1] ... et dans les ténèbres les lier

la seconde composante de telle sorte que l'automate $\mathcal{A}_2$ ait accès aux informations du calcul de $\mathcal{A}_1$ (l'automate que l'on obtient n'est alors plus exactement l'automate produit au sens usuel du terme, mais l'ensemble de ses états est vu comme le produit des ensembles des deux automates initiaux).

Par la suite nous utiliserons fréquemment cette technique de construction « par couches ». On expliquera dans un premier temps le fonctionnement d'un automate (la couche inférieure) puis on en décrira un second (de même dimension et fonctionnant sur le même voisinage) qui pourra alors utiliser, lors de son fonctionnement les états présents sur la couche inférieure (ce nouvel automate sera alors la couche supérieure). On pourra ainsi ajouter indéfiniment de nouvelles couches, chacune ayant accès aux informations calculées par les précédentes.

Il arrivera également que l'on construise plusieurs couches interdépendantes, c'est-à-dire que chacune sera décrite séparément mais tout en utilisant quelques informations portées par l'autre couche. Une telle description est avantageuse par rapport à une description de l'automate en une seule couche si les deux « sous-automates » que l'on décrit n'interagissent que peu l'un avec l'autre (seules les cellules dans des états bien spécifiques tiennent compte de ce qui se passe sur l'autre couche).

Il faut enfin voir que le nombre d'états de l'automate produit est le produit des nombres d'états des deux automates de départ. Ajouter une nouvelle couche à un automate signifie donc multiplier le nombre d'états qu'il avait initialement par le nombre d'états nécessaires à la nouvelle couche. En utilisant de telles techniques il est courant d'obtenir des automates avec un très grand nombre d'états. Cependant, comme il a été dit précédemment, nous ne nous soucierons pas de cette croissance du nombre d'états du moment que l'on peut garantir que l'on conserve un ensemble fini (ce qui est toujours le cas si chacune des couches peut-être réalisée à l'aide d'un nombre fini d'états).

1.4.4 Stockage

Le stockage est une technique très similaire au produit cartésien, mais dont l'utilité est différente. Cela consiste tout simplement à donner plusieurs états à une seule cellule (ou plus exactement plusieurs états à toutes les cellules). Supposons par exemple que l'on ait un automate $\mathcal{A} = (d, \mathcal{Q}, \delta, V)$. Par définition de l'automate, des cellules et de leurs états, à chaque instant chacune des cellules de $\mathcal{A}$ n'est que dans un unique état de $\mathcal{Q}$.

Si l'on considère maintenant un automate dont l'ensemble d'états est $\mathcal{Q} \times \mathcal{Q}$, cela signifie alors que chaque cellule de notre nouvel automate est dans deux états de $\mathcal{Q}$ à la fois à chaque instant. Puisque ces deux états sont sur la même cellule il n'est pas nécessaire de faire évoluer chacun de ces états séparément. Au contraire, on peut alors donner une règle de transition locale (sur le même voisinage V que l'automate initial $\mathcal{A}$) qui dépende finalement de « deux fois plus » d'états qu'avant puisque sur chaque cellule du voisinage on voit maintenant deux états au lieu d'un.

Cette technique consistant à placer plusieurs états d'un certain automate sur une même cellule est appelée stockage. Lorsque l'on part d'un automate de référence bien précis on aura tendance à dire que l'automate que l'on construit a « plusieurs états sur chaque cellule ». Ceci est bien évidemment faux au sens strict du terme puisque le nouvel automate a en réalité beaucoup plus d'états que

l'automate de départ, chaque état pouvant être interprété comme une famille de sous-états. Ainsi chaque cellule du nouvel automate a bien un unique état, mais celui-ci est interprété comme de nombreux états de l'automate de départ.

1.4.5 Marquage et sous-états

Lorsque l'on décrit un automate effectuant des tâches complexes, il est souvent nécessaire qu'au cours du calcul certaines cellules aient à jouer un rôle particulier. Afin de distinguer ces cellules, on aimerait pouvoir les « marquer » de telle sorte qu'elles puissent fonctionner différemment lorsque c'est nécessaire, sans pour autant devoir redéfinir entièrement leur comportement (l'idée étant que ces cellules se comporteront comme toutes les autres dans la plupart des circonstances, mais à certains moments précis ou au contact de certains états particuliers elles devront agir différemment).

Pour cela, on utilise une méthode très simple de produit cartésien en ajoutant une couche supplémentaire. Cette nouvelle couche peut alors soit contenir un état « neutre » q_n soit contenir un état particulier q_s (éventuellement plusieurs états particuliers si l'on veut pouvoir marquer la cellule de différentes manières). Chacun de ces états est « inerte » en ce sens qu'il n'a pas de raison de changer au cours du temps, et qu'il n'affecte pas le comportement de ses voisins (sur la même couche). Les informations contenues sur cette couche de marquage ne sont donc observées et modifiées que par les autres couches de l'automate (celles qui font effectivement le calcul).

Comme nous l'avons vu à propos du groupage, nous dirons souvent qu'une cellule qui a été marquée par l'état q_s est dans l'état q_s. En réalité c'est un abus de langage puisque cela signifie simplement que la cellule est dans un état que l'on considère comme un produit de sous-états et dont l'une des composantes est q_s. La différence est importante puisque si la cellule était vraiment dans l'état q_s (et uniquement cet état) elle ne pourrait pas porter d'autre information et donc elle ne continuerait pas à participer au calcul. Or lorsque l'on marque une cellule on ne veut pas que ce marquage empêche la cellule de fonctionner. De la même manière, il est possible de marquer une même cellule à l'aide de plusieurs états à la fois, simplement en effectuant chaque marquage sur une nouvelle couche. Ainsi, ce n'est pas parce qu'on dit qu'une cellule est dans l'état q_s qu'elle ne peut pas être également dans l'état q'_s. Les états q_s et q'_s doivent alors être considérés comme des *sous-états* et non des états à part entière, un véritable état étant un groupement de sous-états.

Afin de s'assurer que l'on n'utilise qu'un nombre fini d'états, il suffit de vérifier que l'on n'utilise qu'un nombre fini de marquages différents (donc on ne crée qu'un nombre fini de couches supplémentaires).

1.4.6 Restriction de l'espace de travail

Comme dans le cas des machines de Turing, il est possible sur des automates cellulaires de restreindre l'espace de travail tout en conservant la même capacité de calcul. On montre ainsi que tout langage qui peut être reconnu par un automate cellulaire unidimensionnel fonctionnant sur le voisinage standard en temps t peut être reconnu par un automate cellulaire fonctionnant sur le même voisinage en temps t également tel que seules les cellules de $\mathbb{N}$ participent au calcul (toutes les cellules négatives restent dans l'état B à tout instant).

Un tel résultat est obtenu en « repliant » l'espace au niveau du point qui délimite l'espace de calcul autorisé (ici on replie autour de l'origine) comme illustré sur la figure 1.8. Chaque cellule de $\mathbb{N}$ reçoit alors deux états au lieu d'un (l'ensemble des états de l'automate augmente donc de $\mathcal{Q}$ à $\mathcal{Q} \times \mathcal{Q}$). Il est facile de voir que l'opération de pliage n'éloigne jamais les cellules les unes des autres. Ainsi, si le voisinage V est convexe et symétrique, tous les états qu'une cellule doit connaître pour calculer les états successeurs des états qu'elle a reçu par pliage se trouvent dans son voisinage et le calcul peut être réalisé (en fonction de leur position par rapport à l'origine elles savent où chercher les informations).

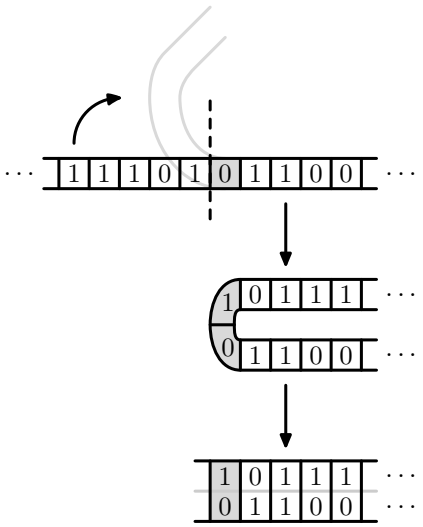

FIGURE 1.8 – Méthode de pliage de l'espace.

En dimensions supérieures on peut effectuer la même transformation qui correspond alors à une symétrie selon un hyperplan (symétrie axiale en dimension 2 par exemple). Pour pouvoir réaliser cette opération sans perturber le fonctionnement de l'automate il faut que le voisinage soit convexe et symétrique selon l'hyperplan de même orientation et passant par l'origine.

Il est possible d'appliquer plusieurs fois cette transformation selon des axes différents. En dimension 2 on peut ainsi restreindre le calcul d'un automate reconnaissant un langage au quart de plan positif $\mathbb{N}^2$ en repliant successivement le long de chacun des axes.

1.4.7 Signaux

Lors de la construction d'un automate cellulaire il est très souvent pratique de pouvoir faire en sorte que des cellules éloignées l'une de l'autre puissent facilement s'échanger de l'information. Une telle interaction peut être réalisée à l'aide de *signaux*.

Considérons l'automate cellulaire de dimension 1 à deux états ($\mathcal{Q} = \{0, 1\}$) fonctionnant sur le voisinage usuel $V = \{-1, 0, 1\}$ dont la règle locale est décrite par

$$\delta(q_1, q_2, q_3) = q_1$$

ce qui signifie que chaque cellule prend au temps suivant l'état de sa voisine de gauche. Cet automate est couramment appelé *shift droit* (ou *décalage droit*) puisque la règle globale est une simple translation de la configuration.

Pourtant, si l'on regarde une configuration où toutes les cellules sont dans l'état 0 à l'exception de l'origine qui est dans l'état 1 et que l'on observe l'évolution de l'automate à partir de cette configuration, on observe un « fond uniforme » de cellules dans l'état 0 qui ne changent pas d'état et un état 1 qui semble se déplacer vers la droite. Dans le cas général, on ne peut pas décrire un automate cellulaire en disant que ce sont des états qui se déplacent parce que les états présents au temps t ne se retrouvent pas nécessairement au temps $(t+1)$, mais dans ce cas particulier on a l'impression que l'état 1 qui se trouve sur la cellule $(c+1)$ au temps suivant est « le même » que celui qui se trouvait sur la cellule c juste avant. Cette interprétation est exactement la même que celle qui nous fait voir une vague fluide se déplaçant autour d'un stade lorsque des supporters se lèvent et s'assoient de manière correctement synchronisée lors d'une *ola*[2].

L'automate décrit précédemment permet donc de faire apparaître un signal (l'état 1) se déplaçant à vitesse maximale (une cellule par unité de temps sur le voisinage $\{-1,0,1\}$) vers la droite. Il est possible en utilisant plus d'états de produire des signaux aux comportement plus complexes. On peut par exemple ralentir le signal en utilisant des états intermédiaires. Ainsi, si l'on utilise 3 états ($\mathcal{Q}=\{0,1,2\}$) on peut définir la règle locale par

$$\begin{aligned}\delta(0,1,0) &= 2\\ \delta(2,0,0) &= 1\\ \delta(0,0,0) &= 0\end{aligned}$$

(et en complétant les transitions manquantes correctement) de telle sorte que ce qui était auparavant un état 1 qui se déplaçait soit maintenant un état alternativement 1 et 2 qui se déplace une étape sur deux. Cet autre signal (maintenant le signal est composé de deux états, mais il se déplace toujours sur un fond apparemment immobile de cellules dans l'état 0) se déplace donc à vitesse 1/2 (le comportement de ce signal est représenté par un diagramme espace-temps sur la figure 1.9). On peut ainsi produire des signaux se déplaçant à n'importe quelle vitesse rationnelle tant que celle-ci ne dépasse pas la vitesse maximale autorisée par le voisinage.

En dimensions supérieures à 1, on pourra construire des signaux se déplaçant dans toute direction rationnelle (un vecteur directeur de la droite a des coordonnées entières) pour peu que le voisinage soit suffisamment complet (nous reviendrons sur ce point).

On peut également définir des signaux beaucoup plus complexes ayant des trajectoires non rectilignes ou se déplaçant à vitesse variable mais nous ne les utiliserons pas dans ce document.

Voyons maintenant comment utiliser ces signaux. Supposons que l'on soit en train de construire un automate afin d'effectuer un calcul (ou une opération quelconque). Si l'on veut transmettre un message d'une cellule c_1 vers une

[2]Remarquons la troublante ressemblance entre le comportement des supporters sportifs lors d'un tel événement et les cellules d'un automate cellulaire. Une étude statistique estime que la vague se déplace à la vitesse de 22 sièges par seconde. Si l'on considère qu'un spectateur peut être influencé par ses voisins sur un rayon de 5 places, on obtient un automate cellulaire naturel fonctionnant au rythme d'environ 4 générations par seconde.

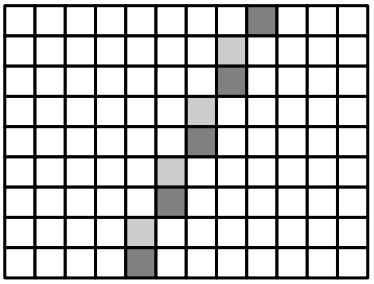

FIGURE 1.9 – Un signal se déplaçant vers la droite à vitesse 1/2 (l'état 1 est en gris foncé, l'état 2 en gris clair).

cellule c_2 et que l'on sait dans quelle direction se trouve c_2 par rapport à c_1, on ajoute une couche supplémentaire à l'automate comme il a été décrit précédemment. La nouvelle couche contiendra l'automate correspondant au signal que l'on veut envoyer (vitesse et direction). Par défaut cette couche est vide (c'est-à-dire qu'elle contient l'état que l'on considère comme le fond homogène du signal) mais lorsque la cellule c_1 veut envoyer un signal, elle ajoute l'état correspondant au signal sur la couche correspondante. Le signal va alors se propager selon sa propre règle de transition (décrite précédemment) en se déplaçant donc en direction de c_2. Il faut par ailleurs que la cellule cible (ici c_2) attende un signal de son côté (il n'est pas possible d'envoyer un signal à n'importe quelle cellule du plan) ce qui signifie qu'elle est distinguée des autres (par exemple par un marquage) et qu'elle sait donc qu'elle attend le signal. Lorsque celui-ci arrive, elle va donc pouvoir modifier son comportement et éventuellement détruire ou modifier le signal.

On utilise donc généralement une couche de marquage lorsque l'on veut envoyer des signaux et seules les cellules ayant été marquées sont affectées par le signal et sont capables de le modifier.

Il est également possible de faire porter de l'information aux signaux ainsi transmis. On dira par exemple que la cellule c_1 envoie un signal contenant une lettre d'un alphabet Σ à la cellule c_2. En réalité on utilise alors un signal différent pour chaque lettre possible, mais chacun de ces signaux se propage de la même manière (ce sont donc en quelque sorte différents éléments d'une même famille de signaux). Ainsi la cellule c_2 peut savoir quelle lettre a été envoyée par la cellule c_1 en fonction du signal qui arrive. On peut dans ce cas placer tous ces signaux sur la même couche de l'automate, ce qui permet d'utiliser un peu moins d'états.

Si l'on utilise une nouvelle couche par signal employé, il n'est pas difficile de faire en sorte que différents signaux se croisent ou se superposent. L'ajout d'un signal correspond à un produit cartésien, le nombre d'états total de l'automate est donc multiplié par le nombre d'états nécessaires à la description du signal (en général cela demande peu d'états, mais on ajoute parfois de nombreux signaux). Toutefois, il est évident que tant que l'on n'ajoute qu'un nombre fini de signaux, chacun ne portant qu'une quantité finie d'information (donc un nombre fini d'éléments dans chaque famille) le nombre total d'états de l'automate reste fini.

Il est parfois pratique de faire en sorte que le signal « laisse une trace » c'est-à-dire qu'il reste un état résiduel (différent du fond) sur toute cellule par

laquelle il est passé au cours de son déplacement. Cette idée n'a que peu d'intérêt en dimension 1 puisque les signaux ne risquent pas de se croiser sans se voir, mais en dimensions supérieures il arrive que l'on veuille faire en sorte que deux signaux se croisent et s'ils sont alors tous deux représentés par un point il arrive qu'ils ne se rencontrent jamais. En les représentant par un segment (la totalité de leur trajectoire passée) il est possible de s'assurer que les signaux se rencontrent et ainsi de déterminer le point exact de l'intersection des trajectoires, ce qui est souvent utilisé pour construire des points ayant des propriétés géométriques par rapport aux états déjà présents (construire le centre d'un rectangle par exemple).

Lorsque l'on veut modifier la vitesse ou la direction d'un signal il faut alors en réalité supprimer le signal existant et en émettre un autre qui se propagera comme on le désire. Souvent nous confondrons ces deux signaux en considérant que c'est un unique signal qui a changé de direction (ou de vitesse). Il arrivera également parfois qu'un signal modifie l'information qu'il porte au cours de sa course. Ici encore, on réalise cette manœuvre en supprimant le premier signal et en en produisant un autre portant une information différente mais par commodité et afin d'alléger les descriptions nous dirons que l'information du signal a changé.

1.4.8 Ondes et lignes de front

En dimensions supérieures à 1 on pourra utiliser un autre type de signaux appelés *ondes*. Les ondes sont très similaires aux signaux que l'on a déjà décrits mais au lieu de se déplacer dans une unique direction elles se propagent selon plusieurs vecteurs, ce qui leur permet d'atteindre une bien plus grande portion de l'espace (cette notion n'a de sens qu'en dimensions strictement supérieures à 1).

Considérons par exemple l'automate en dimension 2 à 3 états $\{0, 1, 2\}$ fonctionnant sur le voisinage de Von Neumann $V_{\text{VN}} = \{(0, 0), \pm(0, 1), \pm(1, 0)\}$ dont la règle de transition est représentée sur la figure 1.10

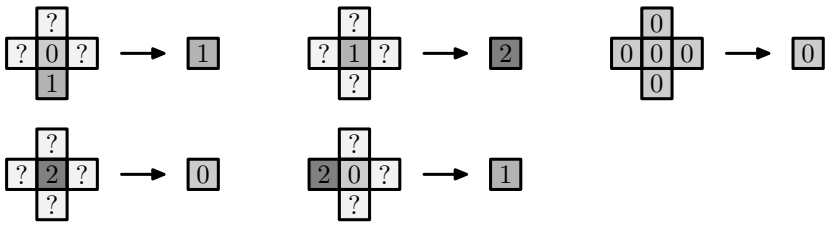

FIGURE 1.10 – Une règle de transition correspondant à une onde.

Cette figure est composée de 7 parties, chacune d'elle représentant une portion de la règle de transition. Par exemple, la partie en haut à gauche signifie que lorsqu'une cellule est dans l'état 0 et que sa voisine de dessous est dans l'état 1, quel que soit l'état de ses autres voisines (représentés ici par des points d'interrogation) elle passe dans l'état 1 à l'étape suivante. On voit en haut à droite de la figure que l'état 0 est quiescent (lorsque toutes les cellules sont dans l'état 0, elles y restent), ce qui va nous permettre de considérer l'état 0 comme un « fond » tel que nous l'avons vu pour les signaux.

La figure 1.11 montre l'évolution de cet automate à partir de la configuration où l'origine est dans l'état 1 tandis que toutes les autres cellules sont dans l'état 0 (toutes les cellules blanches sont dans l'état 0 et l'angle inférieur gauche de l'organe est marqué à chaque temps pour permettre au lecteur de se repérer).

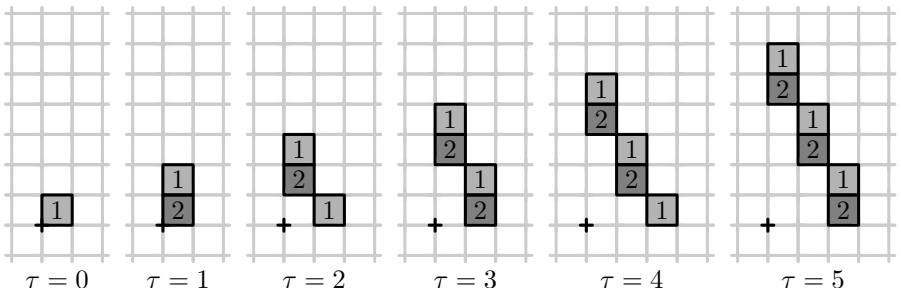

FIGURE 1.11 – L'évolution de l'automate décrit par la figure 1.10.

On voit sur cette figure que les états 1 et 2 (qui représentent l'onde) se déplacent en formant un segment s'éloignant de l'origine vers le haut et la droite. Ce segment est de plus en plus grand au fur et à mesure que le signal s'éloigne. Les extrémités du segment se déplacent le long des axes, l'un à vitesse 1/2 (horizontalement) et l'autre à vitesse 1. L'évolution de l'onde au cours du temps peut être représentée de manière simplifiée comme illustré sur la figure 1.12 où l'on représente les segments discrets sur lesquels se trouvent les états 1 et 2 par de vrais segments de droite (continus).

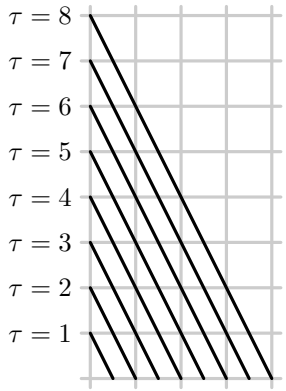

FIGURE 1.12 – Représentation continue de la progression de l'onde (lignes de front).

La position des états représentant une onde (ici les états 1 et 2) à chaque génération sera appelé *front d'onde*. En général lorsque l'on utilisera une onde on se contentera de décrire ses fronts d'onde. Nous ne construirons que des ondes dont les fronts successifs sont des segments dont les extrémités se déplacent en ligne droite à vitesse constante (il est toutefois possible de faire des ondes beaucoup plus complexes, comme par exemple des cercles discrets se propageant à vitesse 1 sur le voisinage de Moore [6]). Nous laisserons au lecteur le soin de se convaincre que si l'on peut construire deux signaux s_1 et s_2 partant d'un même

point, on peut construire l'onde dont les extrémités des fronts sont exactement les positions des signaux s_1 et s_2 au cours du temps (ici l'onde que l'on a décrite correspond aux signaux se déplaçant respectivement vers le haut à vitesse 1 et vers la droite à vitesse 1/2).

En pratique, les ondes sont utilisées pour transmettre une information depuis une cellule vers un groupe de cellules (parfois une seule) dont on ne connaît pas la direction exacte (ou parce qu'il y en a trop pour envoyer des signaux individuels). Les ondes sont donc utilisées, tout comme les signaux, en ajoutant une couche supplémentaire. En règle générale, les cellules ne modifient pas les ondes (donc les ondes qui sont émises ne sont jamais arrêtées ou déviées).

Notons enfin qu'une onde se propage toujours dans l'intérieur d'un cône formé par deux demi-droites (dans l'angle inférieur à 180°). Pour propager une information dans toutes les directions, on pourra utiliser plusieurs ondes dont l'union couvre la totalité du plan. Il arrivera alors que l'on considère ces différentes ondes comme une seule, par abus de langage, si leur fonction et l'information qu'elles portent sont identiques.

1.4.9 Diagramme espace-temps continu

Lorsque nous décrirons le fonctionnement d'un automate cellulaire, il sera parfois préférable de décrire l'évolution de manière « générale » sur toute une famille de configurations initiales possibles (par exemple quelle que soit la taille d'un mot en entrée, dans le cas d'un automate cellulaire de reconnaissance), ou bien tout simplement d'illustrer le fonctionnement de manière indépendante de l'échelle (lorsque l'on veut montrer les interactions entre signaux par exemple).

Dans ce cas, on utilisera une simplification des diagrammes espace-temps vus précédemment consistant à rendre l'espace et le temps continus. Un exemple d'une telle transformation est illustré sur la figure 1.13. Sur la partie gauche de la figure on a représenté un diagramme espace-temps classique (on rappelle que le temps s'écoule vers le haut), pour une configuration initiale correspondant à un mot de longueur 9. Cet automate émet deux signaux partant de chaque extrémité du mot en entrée et se dirigeant l'un vers l'autre à la vitesse d'une cellule par unité de temps (ce qui est la vitesse maximale si l'automate fonctionne sur le voisinage usuel). Lorsque ces deux signaux se rencontrent, ils disparaissent et produisent un nouveau signal qui se déplace vers la gauche à vitesse 1/2. Lorsque ce dernier signal atteint l'origine il disparaît.

La partie centrale de la figure 1.13 représente le fonctionnement du même automate à partir d'une configuration initiale correspondant à un mot plus long (18 cellules). La description du comportement de l'automate que l'on a faite précédemment s'applique tout aussi bien ici (on reconnaît les mêmes signaux). De manière générale, les signaux auront toujours un comportement similaire indépendamment de la longueur du mot en entrée.

Finalement la partie de droite de la figure 1.13 représente la version simplifiée et continue des deux diagrammes précédents. Sur cette dernière représentation les signaux sont représentés par des segments de droite (car nos signaux se propagent à vitesse constante) dont la pente est l'inverse de la vitesse du signal. La simplicité d'une telle représentation est évidente et permet de représenter un comportement beaucoup plus global (de plus on ne représente que les signaux ou états qui nous intéressent afin de rendre le diagramme le plus clair possible).

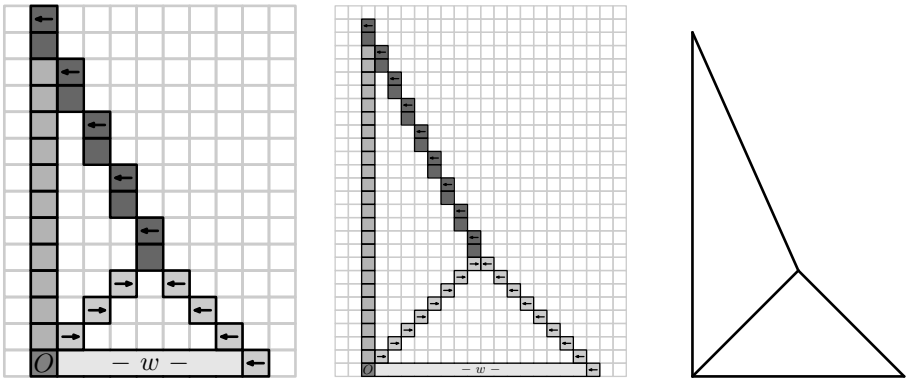

FIGURE 1.13 – La représentation continue (à droite) d'un diagramme espace-temps classique (à gauche).

On peut voir le diagramme continu comme une sorte de limite des diagrammes discrets lorsque la taille des motifs élémentaires (ici le mot en entrée) devient infiniment grande.

Il faut toutefois être prudent lorsque l'on utilise des diagrammes espace-temps continus car il n'est pas toujours possible de passer d'un diagramme espace-temps continu à un véritable automate cellulaire discret (cela peut parfois demander un nombre infini d'états).

1.4.10 Coordonnées rationnelles

Le caractère discret d'un automate cellulaire se retrouve à la fois sur l'espace et le temps, c'est-à-dire qu'un état donné se trouve sur une certaine cellule dont les coordonnées sont entières à un temps représenté par un entier naturel.

Toutefois, il est parfois pratique de considérer des coordonnées tant spatiales que temporelles qui ne sont pas des entiers mais des fractions. Il n'est pas vraiment possible de représenter toutes les positions rationnelles (du moins pas sans perdre de temps), mais pour un entier k fixé à l'avance on peut construire un automate qui travaille sur des positions et des temps de la forme i/k pour tout i. En effet, une méthode simple consiste à utiliser un groupage en considérant que chaque cellule contient en réalité k sous-cellules. Tant que l'on veut rester sur les positions entières on n'utilise qu'une unique case mais si l'on veut pouvoir gérer des positions fractionnaires on peut utiliser les positions prévues à cet effet.

Une autre méthode (souvent moins coûteuse en nombre d'états) consiste à multiplier par k tous les états que l'on voudra représenter comme occupant une position fractionnaire. Si l'on regarde par exemple l'automate dont le fonctionnement est sommairement illustré sur la figure 1.13, il commence par émettre deux signaux se dirigeant l'un vers l'autre à vitesse 1. Si l'on décrit le fonctionnement de l'automate comme il est illustré sur la partie droite de la figure (version continue du diagramme), on dira que si le mot en entrée est de longueur l, les signaux apparaissent respectivement sur les cellules 0 et $l-1$ au temps 0, qu'ils se dirigent l'un vers l'autre à vitesse 1 et qu'ils se rencontrent donc sur la cellule $(l-1)/2$ au temps $(l-1)/2$. Au moment de la collision, un nouveau signal apparaît qui se déplace à vitesse $1/2$ vers l'origine. Il atteint donc l'origine au

temps $3(l-1)/2$.

Lorsque l'on veut décrire un véritable automate qui réalise cette construction, on ne peut pas placer de cellules aux coordonnées $(l-1)/2$ lorsque l est pair. Cependant, il est clair que les coordonnées spatio-temporelles fractionnaires que l'on voudrait utiliser ici sont toutes des demi-entiers (on est donc dans le cas où $k = 2$ selon ce que l'on a dit précédemment). Si l'on dédouble le troisième signal (celui se déplaçant à vitesse $1/2$ vers la gauche) de telle sorte qu'une version porte l'information $(+0)$ tandis que l'autre porte l'information $(+\frac{1}{2})$, on peut encoder au moment de la collision entre les deux signaux de départ si celle-ci a eu lieu en un point de coordonnées entières (on fait partir le signal $(+0)$ ce qui signifie que la position du signal discret est exactement la position du signal idéal continu) ou non (on fait alors partir le signal $(+\frac{1}{2})$ qui signifie que le signal discret est décalé de $\frac{1}{2}$ par rapport au signal idéal). Ceci nous permet finalement, lorsque le dernier signal atteint l'origine, de savoir à quel moment le signal idéal devrait l'atteindre (dans le cas illustré, si le signal apparaît une demi-case plus loin il doit toucher l'origine un temps plus tard puisqu'il se déplace à vitesse $1/2$). Cette technique nous permet donc de réaliser des constructions géométriques à l'aide de signaux sans commettre d'erreurs d'arrondis lors des calculs intermédiaires, même si ultimement le signal ne peut atteindre une cellule donnée qu'à un temps entier.

Ainsi, lorsque l'on dira que l'on est capable de marquer l'origine au temps $l/2$ pour toute entrée de longueur l cela signifiera en réalité que l'origine sera marquée au temps $\lceil l/2 \rceil$ mais que si l'on combine trois fois la construction l'origine sera marquée au temps $\lceil 3l/2 \rceil$ et non pas $3\lceil l/2 \rceil$.

1.4.11 Équivalence linéaire des voisinages complets

Étant donnés deux voisinages complets V et V' en dimension d quelconque il existe une constante $k \in \mathbb{N}$ telle que tout langage reconnu par un automate cellulaire fonctionnant sur le voisinage V' en temps $T : \mathbb{N} \to \mathbb{N}$ est reconnu en temps $k.T$ par un automate cellulaire fonctionnant sur le voisinage V. Ce résultat bien connu (voir par exemple [3]) est obtenu en considérant un entier k tel que $V' \subseteq V^k$.

Il est alors possible, étant donné un automate $\mathcal{A}$ fonctionnant sur le voisinage V' de construire un automate $\mathcal{A}'$ fonctionnant sur V qui simule une génération de $\mathcal{A}$ en k générations : pendant les $(k-1)$ premières générations chaque cellule stocke toute l'information qu'elle peut obtenir en regardant ses voisines. À la k-ième génération, chaque cellule peut donc voir, en observant les informations portées par ses voisines, tous les états qui se trouvent autour d'elle sur un voisinage V^k qui contient V' et elle peut donc appliquer la fonction de transition de l'automate $\mathcal{A}$ à son état.

En répétant alors le procédé il est possible d'effectuer en temps $k.T$ tout calcul que l'automate $\mathcal{A}$ peut effectuer en temps T.

1.4.12 Simulations de machines de Turing par des automates cellulaires

Étant donnée une machine de Turing à un ruban et une tête il est très facile de décrire un automate cellulaire en dimension 1 fonctionnant sur le voisinage standard qui simule directement son fonctionnement (voir [16]). Si la machine

de Turing a pour états $\mathcal{Q}$ et pour alphabet sur son ruban Σ, l'automate que l'on construit a pour états l'ensemble $(\mathcal{Q} \cup \{-\}) \times \Sigma$ où « $-$ » est un nouvel état n'appartenant pas à $\mathcal{Q}$. À toute configuration de la machine de Turing on associe alors une configuration de l'automate cellulaire de telle sorte que chaque cellule de l'automate soit dans l'état (q, a) où $a \in \Sigma$ est la lettre qui se trouve sur le ruban de la machine sur la case correspondant à la cellule de l'automate (l'analogie entre le ruban bi-infini de la machine et l'ensemble $\mathbb{Z}$ des cellules de l'automate est immédiate) et $q \in (\mathcal{Q} \cup \{-\})$ est $-$ pour toutes les cellules sauf celle qui correspond à la position de la tête qui est dans le même état que la tête de la machine.

Il est alors immédiat que si l'automate est initialement dans la configuration qui correspond à la configuration initiale de la machine de Turing il peut à chaque étape appliquer la règle de la machine de Turing dans le voisinage de la cellule qui représente la tête de travail (toutes les cellules qui sont loin de la tête de travail conservent leur état) et ainsi suivre l'évolution de la machine de Turing en temps réel.

Inversement, une machine de Turing peut simuler le fonctionnement d'un automate cellulaire sur une configuration finie. Il suffit pour cela d'écrire les états de l'automate sur le ruban de la machine puis de déplacer la tête jusqu'au bord gauche de la configuration et de parcourir toute la configuration en appliquant la règle locale de l'automate. Lorsque la tête atteint l'autre extrémité de la configuration elle a bien calculé une génération de l'automate et peut alors parcourir de nouveau la configuration (dans l'autre sens cette fois-ci) en appliquant de nouveau la fonction de transition de l'automate à chaque cellule qu'elle rencontre.

Dans chacune des deux simulations décrites ci-dessus, l'objet construit utilise exactement le même espace que l'objet initial. Cela signifie que pour toute fonction $s : \mathbb{N} \to \mathbb{N}$, tout langage reconnu en espace s par une machine de Turing à un ruban est reconnu en espace s par un automate cellulaire de dimension 1 et réciproquement.

Par contre, si un automate cellulaire est capable de simuler le fonctionnement d'une machine de Turing sans perte de temps, la machine de Turing simulant un automate telle que nous l'avons décrite doit parcourir toute la configuration pour calculer une génération de l'automate. Ainsi, pour toutes fonctions s et t de $\mathbb{N}$ dans $\mathbb{N}$, tout langage reconnu par une machine de Turing à un ruban en temps t et espace s est reconnu par un automate cellulaire en temps t et espace s également, tandis que tout langage reconnu par un automate cellulaire en dimension 1 en temps t et espace s est reconnu par une machine de Turing en espace s et en temps $t \times s$.

1.5 Les classes de faible complexité

Maintenant que l'on a défini la reconnaissance de langages par des automates cellulaires nous pouvons définir, similairement à ce qui se fait dans le cas des machines de Turing, les classes de complexité que l'on étudiera par la suite.

1.5.1 Temps réel

Nous avons vu qu'il existait une notion de « vitesse maximale » de propagation de l'information lors d'un calcul d'automate cellulaire. Plus précisément pour chaque cellule de $\mathbb{Z}^d$ il existe un temps minimal avant lequel l'état de l'origine ne peut pas être affecté par la lettre qui se trouvait initialement sur cette cellule (appelé rang de la cellule).

Il semble peu raisonnable de chercher à reconnaître des langages sans laisser le temps à l'automate d'avoir pu « lire » la totalité du mot en entrée, c'est pourquoi on définit la notion de *temps réel*. Étant donné un voisinage V en dimension d, le temps réel correspondant à des mots de taille $(n_1, \ldots, n_d)$ est le temps minimal pour que toutes les lettres du mot en entrée aient pu affecter l'état de l'origine (selon le principe de vitesse maximale de propagation).

On peut voir le temps réel comme le plus petit temps en lequel un automate cellulaire peut reconnaître un langage simple (mais non trivial) tel que le langage des mots sur l'alphabet $\{a, b\}$ ne contenant qu'un unique a.

Formellement, on a la définition suivante :

Définition 1.5.1 (Temps réel)
En dimension d, étant donné un voisinage V, on définit la fonction temps réel *associée au voisinage V par*

$$\mathrm{TR}_V : \begin{cases} \mathbb{N}^d & \to & \mathbb{N} \\ (n_1, \ldots, n_d) & \mapsto & \max\{\mathrm{rg}(c) \mid c \in [\![0, n_1 - 1]\!] \times \ldots \times [\![0, n_d - 1]\!]\} \end{cases}$$

On dira alors qu'un langage L est reconnu en temps réel sur le voisinage V s'il existe un automate cellulaire fonctionnant sur ce voisinage qui reconnaisse L en temps TR_V. Étant donné un entier k, on dira que L est reconnu en temps réel plus k si L est reconnu en temps

$$\mathrm{TR}_V + k : (n_1, \ldots, n_d) \mapsto \mathrm{TR}_V(n_1, \ldots, n_d) + k$$

Quelques exemples

En dimension 1, sur le voisinage standard V_{std}, le rang de la cellule $c \in \mathbb{Z}$ est $|c|$. Il faut donc attendre $(n-1)$ étapes pour que la lettre la plus à droite d'un mot de taille n (cette lettre se trouvant sur la cellule $(n-1)$) puisse affecter l'état de l'origine. La fonction temps réel pour ce voisinage est donc

$$\mathrm{TR}_{V_{\mathrm{std}}} : n \mapsto n - 1$$

De manière générale, sur le voisinage $[\![-x_n, x_p]\!]$, avec $x_n, x_p \in \mathbb{N}$, la fonction temps réel ne dépend pas de x_n et vaut :

$$\mathrm{TR}_{[\![-x_n, x_p]\!]} : n \mapsto \lceil (n-1)/x_p \rceil$$

En dimension 2, sur le voisinage de Von Neumann, la cellule (x, y) est de rang $|x| + |y|$. La lettre placée sur la cellule $(n-1, m-1)$ d'une configuration initiale correspondant à un mot de taille (n, m) est donc la dernière à pouvoir atteindre l'origine, qu'elle ne peut atteindre qu'à partir du temps $(n + m - 2)$, ce qui nous donne la fonction :

$$\mathrm{TR}_{V_{\mathrm{VN}}} : (x, y) \mapsto x + y - 2$$

En dimension quelconque, le temps réel associé au voisinage de Von Neumann est

$$\mathrm{TR}_{V_{\mathrm{VN}}} : (x_1, \ldots, x_d) \mapsto x_1 + \ldots + x_d - d$$

ce qui nous permet de retrouver le cas du voisinage usuel en dimension 1.

Enfin, dans le cas du voisinage de Moore en dimension 2, la cellule (x, y) est de rang $\max(x-1, y-1)$. Sur un mot en entrée de taille (n, m) avec $n > m$, toutes les lettres se trouvant initialement sur la dernière colonne $\{(n-1, x) \mid 0 \leq x \leq m-1\}$ sont donc toutes les plus éloignées de l'origine (le fait qu'ici il existe plusieurs « lettres les plus éloignées » est une différence assez importante entre le voisinage de Moore et celui de Von Neumann). Dans le cas où $m > n$ ce sont les lettres sur la dernière ligne $\{(x, m-1) \mid 0 \leq x \leq n-1\}$ qui le sont et dans le cas particulier où $n = m$ toutes les lettres se trouvant sur le bord extérieur du mot (l'union de la dernière colonne et la dernière ligne) sont les plus éloignées. La fonction temps réel est donc

$$\mathrm{TR}_{V_{\mathrm{M}}} : (x, y) \mapsto \max(x-1, y-1)$$

et en dimension quelconque

$$\mathrm{TR}_{V_{\mathrm{M}}} : (x_1, \ldots, x_d) \mapsto \max\{x_1 - 1, \ldots, x_d - 1\}$$

Les voisinages non complets

Si l'on considère un voisinage non complet V en dimension d, c'est-à-dire tel qu'il existe une cellule $c \in \mathbb{Z}^d$ qui n'appartienne à V^k pour aucun k, la définition de temps réel que l'on a donnée n'a aucun sens (dès que le mot en entrée est assez grand le temps réel est infini).

Si l'on veut effectuer de la reconnaissance de langages sur de tels voisinages il faut alors légèrement modifier la définition du temps réel pour ne considérer que les cellules qui appartiennent à V^∞. On définit donc la fonction de temps réel par
TR_V :

$$\begin{cases} \mathbb{N}^d & \to & \mathbb{N} \\ (n_1, \ldots, n_d) & \mapsto & \max\{\mathrm{rg}(c) \mid c \in \left(\llbracket 0, n_1 - 1 \rrbracket \times \ldots \times \llbracket 0, n_d - 1 \rrbracket\right) \cap V^\infty\} \end{cases}$$

Cette définition coïncide avec la définition précédente sur les voisinages complets puisqu'on a alors $V^\infty = \mathbb{Z}^d$.

1.5.2 Temps linéaire

On dira d'un langage L de dimension d sur un alphabet Σ qu'il est reconnu par un automate $\mathcal{A}$ en *temps linéaire* s'il existe un entier p tel que L est reconnu par $\mathcal{A}$ en temps

$$p.\mathrm{TR}_V : (n_1, \ldots, n_d) \mapsto p.\mathrm{TR}_V(n_1, \ldots, n_d)$$

Pour la plupart des voisinages usuels (les voisinages standard et one-way en dimension 1 et les voisinages de Moore et Von Neumann en dimensions 2 et au-delà) il existe des théorèmes d'accélération linéaire permettant d'affirmer que tout langage reconnu en temps linéaire est également reconnu en temps $\lceil (1 +$

$\varepsilon).\mathrm{TR}_V\rceil$ pour tout ε rationnel strictement positif. Ces théorèmes d'accélération et leurs généralisations à de plus nombreux voisinages seront détaillés dans le chapitre 4.

L'une des grandes questions ouvertes concernant la complexité de reconnaissance de langages sur des automates cellulaires est de déterminer si tout langage reconnu en temps linéaire est également reconnu en temps réel (cette question est ouverte même si l'on ne considère que les automates cellulaires en dimension 1 fonctionnant sur le voisinage usuel).

1.5.3 Espace linéaire

La classe des langages reconnus en *espace linéaire* est définie de manière semblable à la classe des langages reconnus en temps linéaire. On dira ainsi d'un langage L de dimension d sur un alphabet Σ qu'il est reconnu par un automate $\mathcal{A}$ en *espace linéaire* s'il existe un entier p tel que L est reconnu par $\mathcal{A}$ en espace

$$p(\mathrm{Id}-1) : \left\{ \begin{array}{rcl} \mathbb{N}^d & \to & \mathbb{N}^d \\ (n_1,\ldots,n_d) & \mapsto & (p(n_1-1),\ldots,p(n_d-1)) \end{array} \right.$$

Sur les voisinages usuels tels que le voisinage standard en dimension 1 et les voisinages de Moore et Von Neumann en dimension quelconque, on peut appliquer la technique consistant à « plier l'espace » pour restreindre la surface sur laquelle le calcul a lieu (voir la sous-section 1.4.6).

On peut ainsi montrer que sur ces voisinages (et de manière générale tout voisinage présentant suffisamment de symétries pour appliquer la technique de repli de l'espace) tout langage reconnu en espace linéaire est reconnu en espace

$$\mathrm{Id}-1 : \left\{ \begin{array}{rcl} \mathbb{N}^d & \to & \mathbb{N}^d \\ (n_1,\ldots,n_d) & \mapsto & (n_1-1,\ldots,n_d-1) \end{array} \right.$$

c'est-à-dire que l'on peut reconnaître ce langage sur un automate cellulaire pour lequel toute cellule dans l'état B y reste quels que soient les états de ses voisines.

Remarquons enfin que sur les voisinages one-way en dimension quelconque (les voisinages tels que V^∞ soit inclus dans un demi-espace) tous les langages reconnus sont reconnus en espace linéaire.

1.6 À propos de la reconnaissance de langages

Maintenant que nous avons défini le temps réel, revenons sur la définition de la reconnaissance d'un langage en temps T par un automate cellulaire afin de justifier le choix que nous avons fait.

Les définitions 1.3.5, 1.3.6 et 1.3.7 peuvent paraître étranges car elles semblent être redondantes. En réalité les définitions énoncées proviennent de deux idées différentes de reconnaissance (les deux idées sont utilisées dans la littérature) chacune ayant des avantages et des inconvénients en ce qui concerne l'étude que nous nous apprêtons à réaliser.

Dans cette section nous allons présenter ces deux définitions de reconnaissance puis expliquer pour chacune d'elle quels sont les avantages et les inconvénients. Nous montrerons ensuite que les définitions 1.3.5, 1.3.6 et 1.3.7 que l'on a choisi d'utiliser dans ce texte permettent de résoudre les problèmes de chacune.

1.6.1 La définition « forte »

La première définition correspond aux idées que l'on utilise usuellement en complexité (y compris dans le cas des machines de Turing). On a les définitions suivantes :

Définition 1.6.1 (Automate de reconnaissance (version forte))
Un automate cellulaire de reconnaissance sur l'alphabet Σ est un octuplet

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, q_a, q_r)$$

où :

- *d, $\mathcal{Q}$, δ et V ont la même signification que pour un automate cellulaire classique ;*
- *$\Sigma \subseteq \mathcal{Q}$ est un sous-ensemble d'états qui sera utilisé pour encoder le mot dans la configuration initiale de l'automate ;*
- *$B \in \mathcal{Q}$ est un état particulier n'appartenant pas à Σ appelé état* quiescent *ou état* blanc *de l'automate. Il vérifie $\delta(B, B, \ldots, B) = B$;*
- *q_a et q_r sont deux états distincts persistants de $\mathcal{Q}$.*

Définition 1.6.2 (Reconnaissance d'un mot (version forte))
On dira qu'un automate cellulaire de reconnaissance

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, \mathcal{Q}_{\text{acc}}, q_a, q_r)$$

accepte (resp. refuse) le mot $w \in {\Sigma^*}^d$ en temps $t \in \mathbb{N}$ si

$$\mathcal{A}^t(\mathfrak{C}_w)(0) = q_a \text{ (resp. } q_r)$$

Définition 1.6.3 (Reconnaissance de langage (version forte))
On dira qu'un automate cellulaire de reconnaissance

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, q_a, q_r)$$

reconnaît le langage $L \subseteq {\Sigma^*}^d$ en temps $T : \mathbb{N}^d \to \mathbb{N}$ si $\mathcal{A}$ accepte tout mot $w \in L$ de taille $(n_1, \ldots, n_d)$ en temps $T(n_1, \ldots, n_d)$ et refuse tout mot $w \in {\Sigma^*}^d \setminus L$ de taille $(n_1, \ldots, n_d)$ en temps $T(n_1, \ldots, n_d)$.

Ces définitions présentent de nombreux intérêts. On sait par exemple qu'en ce sens seuls des langages récursifs sont reconnus et que réciproquement tout langage récursif est reconnu en temps T pour une certaine fonction T.

On sait également que si un langage est reconnu en temps T alors il est reconnu en temps T' pour toute fonction $T' \geq T$ ce qui permet de définir proprement des classes de complexité telles que P, L ou EXP comme étant l'ensemble des langages reconnus en un temps inférieur à certaines fonctions.

Toutefois cette définition est mal adaptée à la notion de temps réel. En effet, puisque les états q_a et q_r sont persistants, l'automate ne peut entrer dans un de ces états (et donc accepter ou refuser le mot en entrée) que s'il est sûr d'avoir reçu toutes les informations susceptibles d'avoir une influence sur l'appartenance du

mot au langage ce qui nécessite, pour quasiment tous les langages non triviaux, que l'automate sache qu'il a bien reçu toutes les lettres du mot.

En dimension 1 sur le voisinage standard $\{-1, 0, 1\}$, la dernière lettre d'un mot de taille n peut influencer l'origine à partir du temps $(n-1)$. Cependant, au temps $(n-1)$ l'origine n'a aucun moyen de savoir si le mot en entrée est de longueur exactement n ou s'il y avait d'autres lettres qui n'ont pas encore eu le temps de faire parvenir leur information. Pour cette raison, même dans le cas d'un langage très simple comme par exemple le langage L des mots sur l'alphabet $\{a, b\}$ contenant exactement un a l'origine ne peut pas dire avec certitude si le mot appartient à L en seulement $(n-1)$ étapes (si par exemple il n'y a eu aucun a jusqu'ici, il se peut que le mot soit plus long et contienne un a plus loin). Si l'on veut donc que l'automate reconnaisse le langage L dans le sens que nous venons de définir il faut donc attendre au moins n générations pour un mot de taille n, ce qui représente un temps de plus que le temps minimum pour que chaque cellule ait pu transmettre ses informations à l'origine.

On peut alors penser que cette définition entraîne simplement une augmentation de la fonction temps réel de 1. Ceci est vrai en dimension 1 pour tout voisinage, mais pas en dimension 2 et au-delà. Il existe ainsi des voisinages sur lesquels on peut reconnaître des langages en temps exactement $\mathrm{TR}_V(n, m)$ pour certains couples (n, m).

Le cas du voisinage de Von Neumann

Définissons un automate cellulaire $\mathcal{A}$ de dimension 2 à 3 états $\mathcal{Q} = \{B, \circ, +\}$ fonctionnant sur le voisinage de Von Neumann. Le fonctionnement de cet automate peut être décrit de la manière suivante :

- On ne s'intéresse qu'à l'évolution de $\mathcal{A}$ à partir d'une configuration où toutes les cellules de $\mathbb{Z}^2$ sont dans l'état B sauf les cellules dans le rectangle $[\![0, n-1]\!] \times [\![0, m-1]\!]$ pour des entiers m et n, qui elles sont dans l'état $\circ$ (ces configurations correspondent à des images sur l'alphabet à une lettre $\{\circ\}$) ;
- À chaque étape, les états $\circ$ et les états $+$ se déplacent vers le bas sauf s'il y a un état B en dessous d'eux, auquel cas ils essaient de se déplacer vers la gauche. S'il y a également un état B sur leur gauche l'état disparaît ;
- Lorsque deux états de $\{\circ, +\}$ se rencontrent sur une cellule, ils fusionnent en un état $+$.

Les règles détaillées de cet automate sont illustrées sur la figure 1.14 (les cases ⊙+ représentent une cellule dans l'état $\circ$ ou l'état $+$), un exemple d'évolution de l'automate est illustré sur la figure 1.15.

On s'intéresse ici au temps minimum qu'il faut pour que toutes les lettres d'un mot en entrée puissent atteindre l'origine et que de plus l'origine puisse savoir le plus tôt possible que toutes les lettres sont arrivées. Il faut donc interpréter l'état $\circ$ de l'automate $\mathcal{A}$ comme une unique lettre du mot en entrée, l'état $+$ comme une superposition de plusieurs lettres (on ne peut pas compter le nombre exact de lettres car cela demanderait un nombre infini d'états, mais nous n'aurons pas besoin du nombre précis) et l'état B comme une absence de lettres (donc similaire à l'état B usuel sur un automate de reconnaissance). Les

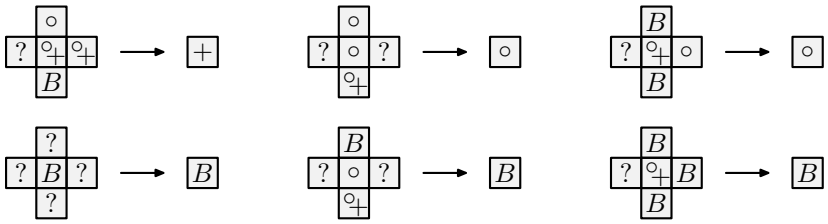

FIGURE 1.14 – Les règles de l'automate $\mathcal{A}$ à trois états B, $\circ$ et $+$.

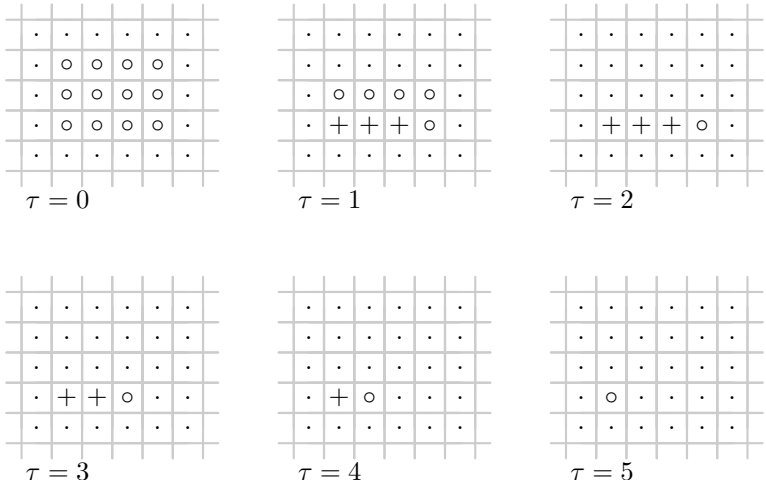

FIGURE 1.15 – Évolution de l'automate $\mathcal{A}$ sur une entrée de taille $(4, 3)$ (l'état B est représenté par un point pour éviter de surcharger la figure).

lettres initialement présentes sur le mot en entrée (représentées ici par l'état $\circ$) se déplacent à vitesse maximale vers l'origine en se déplaçant d'abord vers le bas puis vers la gauche. Les lettres qui se trouvent initialement sur des cellules (x, y) avec $x + y = d$ atteindront l'origine au temps d. Pour une valeur de d fixée, l'ensemble des lettres du mot en entrée qui atteignent l'origine au temps d sont sur une même diagonale. On remarque par ailleurs que seules l'origine et la lettre la plus éloignée du mot en entrée (celle qui se trouve le plus en haut et le plus à droite) sont seules sur leur diagonale si le mot est de dimensions (n, m) où n et m sont tous deux strictement supérieurs à 1.

Cela signifie que l'automate $\mathcal{A}$ que l'on a décrit est tel que sur une entrée de taille (n, m) avec $n, m > 1$ l'origine est dans l'état $\circ$ exactement aux temps 0 et $(n+m-2) = \mathrm{TR}_{V_{\mathrm{VN}}}(n, m)$, ce qui permet donc à l'origine de savoir exactement quand il s'est écoulé $\mathrm{TR}_{V_{\mathrm{VN}}}(n, m)$ étapes de calcul, ce qui lui permet alors de savoir qu'elle a pu recevoir les informations de toutes les lettres.

Ainsi, selon la définition de reconnaissance que nous venons de donner, il est possible pour un automate cellulaire fonctionnant sur le voisinage de Von Neumann de reconnaître si un mot w de taille (n, m) avec $n, m > 1$ appartient ou non à un langage non trivial en temps exactement $\mathrm{TR}_{V_{\mathrm{VN}}}(n, m)$.

La méthode que l'on a exposée ici ne fonctionne cependant pas si l'entrée est de taille $(1, m)$ ou $(n, 1)$ car dans ce cas toutes les lettres arrivent seules sur l'origine et il n'y a pas moyen de distinguer la dernière des précédentes. Dans le cas du voisinage de Von Neumann, si l'on demande donc à l'automate de reconnaître un langage non trivial la fonction de temps « minimale » qu'il faut lui donner est donc

$$T : \begin{cases} \mathbb{N}^2 & \to & \mathbb{N} \\ (1, m) & \mapsto & m \\ (n, 1) & \mapsto & n \\ (n, m) & \mapsto & (n + m - 2) \quad \text{si } n, m > 1 \end{cases}$$

C'est cette fonction qui correspond à la notion de temps réel dans le cas du voisinage de Von Neumann si l'on définit la reconnaissance d'un mot en imposant que l'automate accepte ou refuse lorsqu'il a toutes les informations et qu'il ne peut alors plus « changer d'avis ».

On voit sur cet exemple que cette fonction est beaucoup moins simple que la fonction $\mathrm{TR}_{V_{\mathrm{VN}}} : (n, m) \mapsto n+m-2$ que l'on obtient avec la définition que l'on a choisie, et qu'elle n'a pu être déterminée qu'en observant précisément la manière de déplacer les lettres vers l'origine « le plus rapidement possible ». Déterminer avec précision cette fonction dans le cas de voisinages bien plus complexes peut alors devenir très difficile.

On peut montrer que dans le cas général, cette nouvelle fonction de temps réel (correspondant à la définition de reconnaissance que l'on est en train de considérer) ne diffère de la fonction temps réel que l'on avait définie précédemment que d'au plus 1 (et elle est toujours supérieure ou égale à la précédente).

Cette différence est donc bien souvent négligeable, mais dans cette thèse il nous arrivera d'attacher une grande importance à ces différences d'une unité (uniquement dans le cas du temps réel) et il est donc important d'être capable de définir et de calculer précisément la fonction temps réel.

La définition usuelle de complexité en temps complique donc considérablement la définition du temps réel.

1.6.2 La définition « faible »

Dans la littérature, il est fréquent lorsque l'on manipule le temps réel d'adopter les définitions suivantes de reconnaissance de langage :

Définition 1.6.4 (Automate de reconnaissance (version faible))
Un automate cellulaire de reconnaissance sur l'alphabet Σ est un septuplet

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, \mathcal{Q}_{\text{acc}})$$

où :

- *d, $\mathcal{Q}$, δ et V ont la même signification que pour un automate cellulaire classique ;*
- *$\Sigma \subseteq \mathcal{Q}$ est un sous-ensemble d'états qui sera utilisé pour encoder le mot dans la configuration initiale de l'automate ;*
- *$B \in \mathcal{Q}$ est un état particulier n'appartenant pas à Σ appelé état* quiescent *ou état* blanc *de l'automate. Il vérifie $\delta(B, B, \ldots, B) = B$;*
- *$\mathcal{Q}_{\text{acc}} \subseteq \mathcal{Q}$ est un ensemble d'états dits « acceptants ».*

Définition 1.6.5 (Reconnaissance d'un mot (version faible))
On dira qu'un automate cellulaire de reconnaissance

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, \mathcal{Q}_{\text{acc}}, q_a, q_r)$$

accepte le mot $w \in {\Sigma^*}^d$ en temps $t \in \mathbb{N}$ si

$$\mathcal{A}^t(\mathfrak{C}_w)(0) \in \mathcal{Q}_{\text{acc}}$$

Définition 1.6.6 (Reconnaissance de langage (version faible))
On dira qu'un automate cellulaire de reconnaissance

$$\mathcal{A} = (d, \mathcal{Q}, \delta, V, \Sigma, B, \mathcal{Q}_{\text{acc}})$$

reconnaît le langage $L \subseteq {\Sigma^*}^d$ en temps $T : \mathbb{N}^d \to \mathbb{N}$ si $\mathcal{A}$ accepte un mot $w \in {\Sigma^*}^d$ de taille $(n_1, \ldots, n_d)$ en temps $T(n_1, \ldots, n_d)$ si et seulement si $w \in L$.

Cette définition est très différente de la précédente. Ici on ne demande pas à l'automate d'attendre d'avoir toutes les informations pour répondre car on considère qu'à tout instant il accepte ou refuse (les états qui ne sont pas des états d'acceptation sont considérés comme des états de refus). Cependant, l'automate peut changer sa réponse d'un temps sur l'autre mais on ne tiendra compte que de la réponse au temps $T(n_1, \ldots, n_d)$ sur une entrée de taille $(n_1, \ldots, n_d)$.

D'une certaine manière, on peut voir la réponse de l'automate ainsi : au lieu d'attendre un certain temps puis d'annoncer avec certitude « le mot que j'ai reçu en entrée appartient (ou non) au langage L » comme c'était le cas auparavant, l'automate déclare à chaque génération quelque chose signifiant « actuellement, étant données les informations que j'ai obtenues, je considère que le mot appartient (ou non) au langage L », mais il pourrait changer d'avis par la suite en recevant d'autres informations ou en poursuivant son calcul.

L'intérêt principal de cette définition est qu'elle est tout à fait adaptée à l'idée intuitive que l'on se fait du temps réel ce qui rend alors très simple la définition de la fonction temps réel.

En effet, à un instant donné la réponse de l'automate signifie « en fonction des informations que j'ai reçues actuellement, je considère que le mot appartient (ou non) au langage L » et donc si l'on définit (comme nous l'avons fait dans la définition 1.5.1) le temps réel comme étant le temps minimum qu'il faut pour que toute lettre du mot en entrée ait pu atteindre l'origine la réponse de l'automate peut bien dépendre de toute lettre du mot.

Malheureusement, cette définition de reconnaissance a d'importants inconvénients lorsque l'on considère des fonctions de temps différentes du temps réel.

En effet avec une telle définition, il n'est pas nécessairement vrai que pour deux fonctions T_1 et T_2 de $\mathbb{N}^d$ dans $\mathbb{N}$, si $T_1 \leq T_2$ alors tout langage reconnu en temps T_1 l'est également en temps T_2. Il existe même des langages reconnus par un automate cellulaire en une certaine fonction de temps T qui ne sont pas récursifs. Si l'on considère par exemple un ensemble $E \subseteq \mathbb{N}$ non récursif, puis que l'on définit le langage L_E des mots dont la longueur est dans E. Ce langage est reconnu par un automate cellulaire extrêmement simple dont l'origine est alternativement dans un état acceptant puis un état non acceptant (indépendamment de l'entrée) en temps

$$T : \begin{cases} \mathbb{N} & \to & \mathbb{N} & \\ x & \mapsto & 2x & \text{si } x \notin E \\ x & \mapsto & 2x+1 & \text{si } x \in E \end{cases}$$

(on pourrait même définir une fonction de temps qui ne prenne que les valeurs 0 et 1).

On peut d'ailleurs montrer que cette définition de reconnaissance est liée aux classes de complexité non uniforme connues :

Proposition 1.6.1
En dimension 1*, l'union sur toutes les fonctions* $T : \mathbb{N} \to \mathbb{N}$ *bornées par un polynôme des classes des langages reconnus (au sens faible des définitions 1.6.4, 1.6.5 et 1.6.6) en temps* T *est exactement la classe* $\mathrm{P} / \log$.

Preuve : Par double inclusion. Soit L un langage reconnu au sens faible par un automate cellulaire en temps T pour $T \leq R$ où R est un polynôme. Alors pour tout entier n, $T(n)$ est un entier qui peut être codé par un mot de longueur logarithmique en n.

Il suffit ensuite de construire une machine de Turing qui sur une entrée w et le conseil $T(|w|)$ simule le fonctionnement de l'automate cellulaire (en temps polynomial) pendant exactement $T(|w|)$ générations et renvoie la réponse de l'automate.

Réciproquement, si le langage L est reconnu en temps R (où R est un polynôme) par une machine de Turing M avec un conseil de taille logarithmique on peut construire un automate cellulaire qui sur l'entrée w va simuler le fonctionnement de la machine M sur tous les conseils possibles (il y a un nombre polynomial de conseils possibles). Chacune des simulations dure $R(|w|)$ étapes.

Il suffit alors de choisir $T(n)$ de telle sorte qu'on interroge l'automate au moment où il vient de finir la simulation de la machine M sur le conseil correspondant à la longueur n. □

Cette définition de la reconnaissance est donc trop puissante pour être utilisée dans le cas général, bien qu'elle soit bien mieux adaptée que la précédente en ce qui concerne le temps réel.

1.6.3 La définition « mixte »

Les deux définitions de la reconnaissance de langages fréquemment considérées présentent donc d'importants inconvénients lorsque l'on veut les utiliser à la fois sur le temps réel et sur les autres classes de complexité. Toutefois, il est possible de combiner les deux pour obtenir une définition qui corresponde parfaitement à ce que l'on attend.

Ainsi, les définitions 1.3.5, 1.3.6 et 1.3.7 contiennent les contraintes des deux définitions vues précédemment.

On dit que l'automate $\mathcal{A}$ reconnaît le mot w en temps t s'il le reconnaît au sens faible en t générations et au sens fort en $(t+1)$ générations.

La reconnaissance au sens fort va nous permettre de ne reconnaître que des langages récursifs, et pour toutes les classes de complexité qui ne sont pas affectées par une variation d'un temps de calcul on peut négliger la contrainte de reconnaissance faible puisque si l'on reconnaît au sens fort en $(t+1)$ générations on reconnaît également au sens faible en $(t+1)$ générations. Ainsi, notre définition coïncide avec la définition forte sur toutes les classes de complexité suffisamment haute pour qu'une variation constante du temps de calcul n'ait pas d'importance (ce qui est le cas de la grande majorité des classes de complexité usuelles pour lesquelles le temps de calcul est contraint à une constante multiplicative près).

Par contre si l'on s'intéresse à la reconnaissance en temps réel ou en temps réel plus une constante, notre définition coïncide avec la définition faible puisque tout langage reconnu en temps réel au sens faible est reconnu en temps réel plus une génération au sens fort (car en temps $(\mathrm{TR}_V + 1)$ l'automate est capable de savoir qu'il a reçu toutes les lettres du mot et donc de l'accepter ou le refuser « définitivement »).

Les définitions considérées nous permettent ainsi d'avoir une unique notion de reconnaissance et de complexité qui, selon la situation, est équivalente à la version forte ou à la version faible en conservant les avantages des deux (notamment, si un langage L est reconnu en temps T par un automate cellulaire il est également reconnu en temps T' si $T \geq T'$).

Chapitre 2

Équivalences de voisinages au sens du temps réel

All animals are equal, but some animals are more equal than others.

George Orwell – *Animal farm*

Ce chapitre sera consacré aux résultats permettant de montrer que certains voisinages sont équivalents au sens de la reconnaissance de langages en temps réel, c'est-à-dire qu'ils permettent de reconnaître les mêmes langages en temps réel.

Dans un premier temps nous considérerons les automates unidimensionnels et nous obtiendrons une très forte équivalence à l'aide d'un résultat d'accélération par une constante : nous rappellerons le résultat d'accélération connu concernant les automates cellulaires fonctionnant sur le voisinage standard puis nous verrons qu'il est possible d'étendre ce résultat à tous les voisinages semi-complets en dimension 1.

Nous utiliserons alors l'accélération constante pour montrer que tout voisinage semi-complet en dimension 1 est, vis-à-vis de la reconnaissance de langages en temps réel, équivalent au voisinage standard ou au voisinage one-way.

Enfin, nous généraliserons le résultat central obtenu en dimension 1 aux dimensions supérieures pour montrer que pour tout voisinage complet V en dimension d, tout langage reconnu en temps réel par un automate fonctionnant sur l'enveloppe convexe de V peut être reconnu en temps réel par un automate cellulaire fonctionnant sur V.

2.1 Le voisinage standard

En ce qui concerne le voisinage standard $\{-1, 0, 1\}$, on a le résultat suivant :

Théorème 2.1.1
En dimension 1, pour tout entier k, tout langage reconnu par un automate cellulaire en temps $(\mathrm{TR}_{V_{\mathrm{std}}} + k)$ fonctionnant sur le voisinage V_{std} est reconnu par un automate cellulaire en temps $\mathrm{TR}_{V_{\mathrm{std}}}$ sur V_{std}.

Preuve : On considère un automate $\mathcal{A}$ reconnaissant un langage L en temps $(\mathrm{TR}_{V_{\mathrm{std}}} + k)$ et l'on construit un automate $\mathcal{A}'$ qui simule le fonctionnement de $\mathcal{A}$ tout en étant capable d'anticiper cette simulation de k générations au temps $\mathrm{TR}_{V_{\mathrm{std}}}(n)$ sur une entrée de taille n. On peut de plus supposer que seules les cellules positives de $\mathcal{A}$ participent au calcul.

Pour décrire le fonctionnement de l'automate $\mathcal{A}'$, on considère une entrée valide de l'automate $\mathcal{A}$ (codant un mot quelconque). On prend ensuite comme référence les états des cellules de $\mathcal{A}$ au cours du temps. L'état de la cellule i de $\mathcal{A}$ au temps t sera noté $\langle i \rangle_t$. La configuration initiale de $\mathcal{A}$ (et également de $\mathcal{A}'$) est donc, pour une entrée de taille n

$\tau = 0$

…	B	$\langle 0 \rangle_0$	$\langle 1 \rangle_0$	…	$\langle n-1 \rangle_0$	B	…

La construction est basée sur une technique de groupage, c'est-à-dire qu'une cellule de l'automate $\mathcal{A}'$ va contenir plusieurs informations (ici, elle en contiendra exactement $(k+1)$) correspondant aux états de plusieurs cellules de l'automate $\mathcal{A}$ (un état de $\mathcal{A}'$ sera donc un $(k+1)$-uplet d'états de $\mathcal{A}$). Puisque dans la configuration initiale les états sont simples (non groupés), un état simple q sera considéré comme le $(k+1)$-uplet $(q, B, \ldots, B)$.

Il faut intuitivement voir ce $(k+1)$-uplet porté par la cellule i au temps t comme une estimation des états $(\langle i \rangle_t, \ldots, \langle i+k \rangle_t)$. Notons que cette estimation n'est pas nécessairement juste, comme nous allons le voir, mais elle sera corrigée au cours du calcul. L'évolution d'une cellule i au temps t s'effectue donc ainsi :

- La cellule regarde son état propre, c'est-à-dire la première coordonnée de son état (c'est l'estimation de $\langle i \rangle_t$ qui est donc l'état de la cellule i au temps t dans l'évolution de $\mathcal{A}$) ainsi que l'état propre de ses voisines $(i-1)$ et $(i+1)$.
- Ensuite, elle regarde le $(k+1)$-uplet porté par sa voisine $(i+1)$. Elle a alors une estimation des états $\langle x \rangle_t, x \in [\![i-1, i+k+1]\!]$.
- A partir de cette estimation, elle peut calculer une estimation des états $\langle x \rangle_{t+1}, x \in [\![i, i+k]\!]$ qu'elle prend donc comme nouvel état (il suffit d'appliquer la règle de transition locale de $\mathcal{A}$).

Il y a alors deux propriétés à remarquer :

- Lors de l'évolution de l'automate, l'état propre d'une cellule i au temps t est toujours une estimation juste de $\langle i \rangle_t$. En effet, on montre cette propriété par récurrence puisqu'elle est vraie pour toute cellule au temps 0 et qu'une cellule n'utilise que les états propres de ses voisines au temps t pour calculer son état propre au temps $(t+1)$.
- Au temps t, l'ensemble des estimations de toutes les cellules $i \geq n-1-t$ sont justes, c'est-à-dire qu'une cellule $i \geq n-1-t$ est exactement dans l'état $(\langle i \rangle_t, \ldots, \langle i+k \rangle_t)$ (voir figure 2.1). Cette propriété se montre également par récurrence. Au temps 0 l'hypothèse consistant à compléter

les $(k+1)$-uplets par l'état B s'avère être juste pour toutes les cellules à partir de la cellule $(n-1)$ (puisque la configuration est valide et donc après la dernière lettre du mot tous les états sont B). Par la suite, si les estimations de la cellule $(i+1)$ au temps t sont justes, alors puisque les estimations de i au temps $(t+1)$ sont effectuées à partir des états propres de $(i-1)$, i et des estimations faites par $(i+1)$, ces estimations sont justes également.

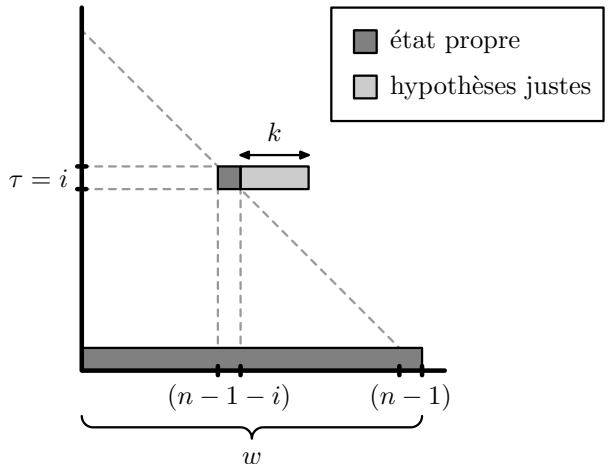

FIGURE 2.1 – Les hypothèses justes de la cellule $(n-1-i)$ au temps i.

La construction est alors terminée puisque l'on sait maintenant qu'au temps $\mathrm{TR}_{V_{\mathrm{std}}}(n) = n-1$ la cellule origine est dans l'état $(\langle 0\rangle_{n-1}, \ldots, \langle k\rangle_{n-1})$ et qu'elle connaît donc l'ensemble des états qui peuvent influer, lors de l'évolution de $\mathcal{A}$, sur l'état $\langle 0\rangle_{n-1+k}$. La cellule origine peut donc conclure sur l'acceptation ou non du mot en entrée par l'automate $\mathcal{A}$ (et donc l'appartenance ou non du mot à L) après seulement $\mathrm{TR}_{V_{\mathrm{std}}}(n) = n-1$ générations de $\mathcal{A}'$. □

Nous allons maintenant nous intéresser à une généralisation du théorème 2.1.1 qui permette d'obtenir le même résultat pour une classe de voisinages beaucoup plus vaste.

2.2 Équivalence des voisinages en dimension 1

2.2.1 Étude préliminaire : croissance des voisinages

Dans toute cette section, nous ne nous intéresserons qu'aux voisinages en dimension 1 et nous les considérerons indépendamment de la notion d'automate cellulaire pour nous concentrer sur une propriété purement algébrique.
Remarque. On rappelle qu'ici encore tous les voisinages considérés contiennent 0.

La proposition 1.2.2 permet d'obtenir les deux caractérisations suivantes en dimension 1 :

Proposition 2.2.1
Un voisinage V est complet si et seulement si ses éléments non nuls sont premiers entre eux dans leur ensemble et V contient au moins un élément strictement positif et un élément strictement négatif.

Preuve : En dimension 1 le $\mathbb{Z}$-module engendré par une partie finie V de $\mathbb{Z}$ est $\mathbb{Z}$ tout entier si et seulement si les éléments de V sont premiers entre eux dans leur ensemble (c'est une conséquence immédiate du théorème de Bézout). De plus, dire que V a un élément strictement positif et un élément strictement négatif est équivalent à dire que son enveloppe convexe contient un voisinage de l'origine.

La proposition 2.2.1 est donc équivalente à la proposition 1.2.2 en dimension 1. □

Proposition 2.2.2
Un voisinage V est semi-incomplet si et seulement si il contient 1 et n'a aucun élément strictement négatif.

Preuve : Si un voisinage V contient 1, tout entier naturel s'écrit comme somme de ses éléments (quitte à n'utiliser que 1 dans la somme) et V est donc semi-complet. Si par ailleurs V n'a aucun élément strictement négatif il est impossible d'en produire par somme de ses éléments et V n'est donc pas complet, il est semi-incomplet.

Réciproquement, si V ne contient aucun élément strictement négatif et ne contient pas 1, 1 ne pourra pas être obtenu par somme d'éléments de V puisque tous les éléments non nuls sont trop grands. Si par contre V contient un élément strictement négatif, alors si 1 peut être obtenu en sommant des éléments de V, V est complet (proposition 2.2.1), et sinon V n'est pas semi-complet. Dans les deux cas, V n'est pas semi-incomplet. □

Définition 2.2.1
Dans toute la suite du chapitre, si V est un voisinage semi-complet nous utiliserons sans les redéfinir les notations suivantes :

$$\begin{aligned} x_p &= \max V \\ -x_n &= \min V \\ t_c &= \min\{t \in \mathbb{N} \mid [\![-x_n, x_p]\!] \subseteq V^{t+1}\} \end{aligned}$$

Remarque. La proposition 2.2.2 nous assure que t_c est bien défini puisque $-x_n = 0$ dans le cas d'un voisinage semi-incomplet.

Proposition 2.2.3
Soit V un voisinage semi-complet. Pour tout $t \in \mathbb{N}$, on a

$$[\![-tx_n, tx_p]\!] \subseteq V^{t+t_c} \subseteq [\![-(t+t_c)x_n, (t+t_c)x_p]\!]$$

(voir figure 2.2)

Preuve : L'inclusion de droite est immédiate par induction. Montrons l'inclusion de gauche. Pour $t = 0$ et $t = 1$ la propriété est immédiate. Pour $t \geq 1$ supposons que

$$[\![-tx_n, tx_p]\!] \subseteq V^{t+t_c}$$

On a donc

$$[\![-(t+1)x_n, (t+1)x_p]\!] = [\![-tx_n, tx_p]\!] + \{-x_n, 0, x_p\} \subseteq V^{t+t_c} + V = V^{t+t_c+1}$$

ce qui nous permet de conclure par induction. □

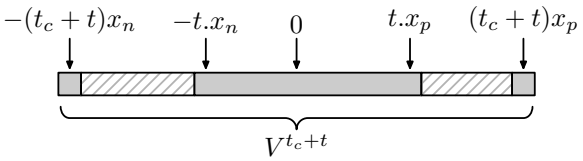

FIGURE 2.2 – Forme générique de $V^{(t+t_c)}$

Définition 2.2.2
Soit V un voisinage semi-complet. Nous appellerons ombre droite *de V au temps $(t + t_c)$ l'ensemble*

$$S^+_{t+t_c}(V) = \left(V^{t+t_c} \cap [\![tx_p, (t+t_c)x_p]\!]\right) - tx_p$$

De même, on appeler ombre gauche *de V au temps $(t + t_c)$ l'ensemble*

$$S^-_{t+t_c}(V) = \left(V^{t+t_c} \cap [\![-(t+t_c)x_n, -tx_n]\!]\right) + tx_n$$

La figure 2.3 illustre cette définition.

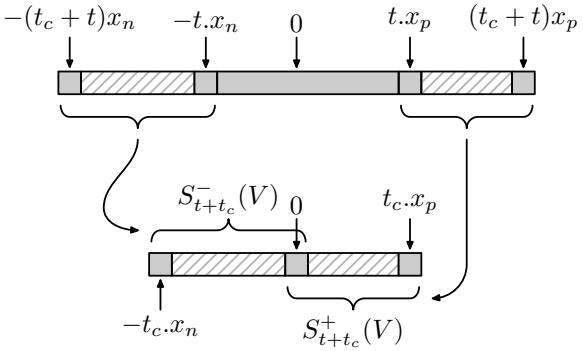

FIGURE 2.3 – Les ombres gauche et droite correspondant au voisinage V^{t_c+t}.

Remarque. D'après la proposition 2.2.3, on a

$$V^{t+t_c} = \left[S^-_{t+t_c}(V) - tx_n\right] \cup [\![-tx_n, tx_p]\!] \cup \left[S^+_{t+t_c}(V) + tx_p\right]$$

Proposition 2.2.4
Soit V un voisinage semi-complet. Les suites

$$\left(S^+_{t+t_c}(V)\right)_{t\in\mathbb{N}} \text{ et } \left(S^-_{t+t_c}(V)\right)_{t\in\mathbb{N}}$$

sont ultimement constantes.

Preuve : Soit $x \in S^+_{t+t_c}(V)$. Alors $(x + tx_p) \in V^{t+t_c}$ et donc

$$(x + (t+1)x_p) \in V^{t+t_c+1}$$

soit finalement

$$x \in S^+_{t+1+t_c}(V)$$

Ceci montre que $\left(S^+_{t+t_c}(V)\right)_{t\in\mathbb{N}}$ est une suite croissante au sens de l'inclusion.

Par ailleurs tous les éléments de cette suite sont des parties de $[\![0, t_c x_p]\!]$ qui est un ensemble fini, ce qui implique que la suite est constante à partir d'un certain rang. La preuve est identique dans le cas de l'ombre gauche. □

Définition 2.2.3
Soit V un voisinage semi-complet. Le temps de stabilisation t_s *de V est le plus petit rang à partir duquel les suites $\left(S^+_{t+t_c}(V)\right)_{t\in\mathbb{N}}$ et $\left(S^-_{t+t_c}(V)\right)_{t\in\mathbb{N}}$ sont constantes.*

On définit également $S^+(V) = S^+_{t_c+t_s}$, $S^-(V) = S^-_{t_c+t_s}$ et

$$x_0 = \min\{x \in \mathbb{N} \mid x \notin S^+(V)\}$$

Remarque. La proposition 2.2.4 assure que t_s est bien défini pour tout voisinage semi-complet.

Nous avons montré le résultat suivant :

Théorème 2.2.1
Pour tout voisinage semi-complet V il existe un entier t_s et deux ensembles $S^-(V) \subseteq [\![-t_c x_n, 0]\!]$ et $S^+(V) \subseteq [\![0, t_c x_p]\!]$ tels que pour tout $t \in \mathbb{N}$,

$$V^{t_c+t_s+t} = \left(S^-(V) - (t_s+t)x_n\right) \cup [\![-(t_s+t)x_n, (t_s+t)x_p]\!] \cup \left(S^+(V) + (t_s+t)x_p\right)$$

(voir figure 2.4)

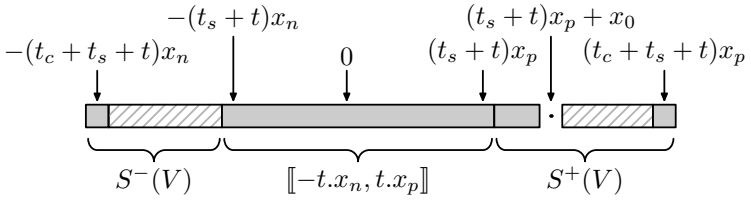

FIGURE 2.4 – Forme générique de $V^{(t_c+t_s+t)}$

2.2.2 Accélération par une constante

L'objectif de cette section est de montrer le théorème suivant :

Théorème 2.2.2
Soit V un voisinage semi-complet. Pour tout $k \in \mathbb{N}$, si un langage L est reconnu par un automate cellulaire fonctionnant sur le voisinage V en temps $\mathrm{TR}_V + k$ alors il est reconnu par un automate cellulaire fonctionnant sur le voisinage V en temps réel.

Considérons un voisinage V semi-complet (on utilisera également toutes les notations définies dans la section 2.2.1). Soit L un langage sur un alphabet Σ reconnu par un automate cellulaire $\mathcal{A}$ fonctionnant sur le voisinage V en temps $\mathrm{TR}_V + k$. Soit $\mathcal{Q}$ l'ensemble des états de $\mathcal{A}$.

Tout comme dans le cas du voisinage standard, nous allons construire un automate cellulaire $\mathcal{A}'$ fonctionnant sur le voisinage V qui simulera le fonctionnement de $\mathcal{A}$ mais en anticipant k étapes de la simulation.

Dans toute la suite de la preuve, on considère l'évolution de $\mathcal{A}$ à partir de la configuration initiale correspondant à un mot $w = w_0 w_1 \dots w_{l-1}$ de longueur l. La configuration initiale est donc

$$\dots BBBw_0w_1 \dots w_{l-1}BBB \dots$$

L'état de la cellule $c \in \mathbb{Z}$ au temps t dans l'évolution de $\mathcal{A}$ sera noté $\langle c \rangle_t$. Les états de $\mathcal{A}'$ seront ici encore des n-uplets de $\mathcal{Q}$ (un état de $\mathcal{A}'$ contiendra l'information de plusieurs états de $\mathcal{A}$).

L'évolution de $\mathcal{A}'$

L'évolution de l'automate $\mathcal{A}'$ à partir de la configuration initiale correspondant au mot w peut être décrite de la manière suivante :

- Temps 0. La configuration initiale de $\mathcal{A}'$ est la même que celle de $\mathcal{A}$. Chaque cellule c est donc dans l'état $\langle c \rangle_0$.
- Temps $0 \to (t_c + t_s)$. Les cellules stockent toute l'information qu'elles peuvent obtenir, c'est-à-dire qu'elles se contentent de regarder les états de leurs voisines et de les mémoriser sans appliquer la règle de transition de $\mathcal{A}$.
- Temps $(t_c + t_s)$. Chaque cellule c connaît exactement tous les états

  $$\{\langle c + x \rangle_0 \mid x \in V^{t_c+t_s}\}$$

  De plus, elle va « supposer » que tous les états de

  $$\{\langle c + x \rangle_0 \mid x \in [\![t_s x_p, (t_c + t_s + k)x_p]\!]\}$$

  qu'elle ne connaît pas encore (ceux qui ne sont pas accessibles parce qu'ils ne sont pas sur les cellules de $V^{t_c+t_s}(c)$) sont B. Remarquons que cette supposition est vraie pour les cellules qui sont suffisamment proches de la fin du mot tandis qu'elle est fausse pour les autres. De même chaque

cellule suppose symétriquement que les états de $\{\langle c-x\rangle_0 \mid x \in [\![t_s x_n, (t_c + t_s + k)x_n]\!]\}$ qu'elle ne connaît pas encore sont B.

À partir de maintenant, nous distinguerons bien, pour une cellule donnée, les informations qu'elle *connaît* avec certitude des informations qu'elle *suppose* (à sa droite ou à sa gauche).

- Temps $(t_c + t_s) \to \infty$. Pendant toute la fin de la simulation, chaque cellule applique la règle de transition de $\mathcal{A}$ à toutes les informations qu'elle contient dans son voisinage. Ainsi, au temps (t_c+t_s+t) la cellule c connaît ou « pense connaître » tous les états $\{\langle c+x\rangle_t \mid x \in V^{t_c+t_s+k}\}$. Il est possible que lors du calcul la cellule voie des informations contradictoires (entre ce qu'elle suppose par exemple et ce que supposent ses voisines pour certaines cellules). Dans ce cas, la cellule donnera toujours la priorité aux informations contenues sur sa voisine la plus à droite lorsqu'il s'agit de suppositions faites à droite, et de sa voisine la plus à gauche pour les suppositions sur sa gauche.

Explication du fonctionnement de $\mathcal{A}'$

Dans cette sous-section, nous allons montrer que l'automate $\mathcal{A}'$, dont le fonctionnement vient d'être décrit, reconnaît bien le langage L en temps réel. Nous continuerons à considérer le fonctionnement de $\mathcal{A}$ (et donc de $\mathcal{A}'$ également) à partir du mot w en entrée pour rendre l'explication plus claire, et nous conserverons les notations introduites précédemment.

Lemme 2.2.1
Au temps $(t_c + t_s)$ la cellule c connaît (correctement) les états $\{\langle c+x\rangle_0 \mid x \in V^{t_c+t_s}\}$.

Preuve : Par induction. Si au temps t la cellule c connaît les états $\{\langle c+x\rangle_0 \mid x \in V^t\}$, elle peut regarder ses voisines et l'information qu'elles portent. Au temps $(t+1)$, elle peut donc contenir l'information

$$\{\langle c+v+x\rangle_0 \mid v \in V, x \in V^t\} = \{\langle c+x\rangle_0 \mid x \in V^{t+1}\}$$

(on notera que le nombre d'états qu'une cellule doit garder en mémoire est borné indépendamment du mot en entrée, et qu'il suffira donc bien d'un nombre fini d'états pour représenter la mémoire d'une cellule) □

Lemme 2.2.2
Au temps $(t_c + t_s + t)$ la cellule c connaît les états $\{\langle c+x\rangle_t \mid x \in V^{t_c+t_s}\}$.

Preuve : Une autre induction. Pour calculer les états

$$\{\langle c+x\rangle_{t+1} \mid x \in V^{t_c+t_s}\}$$

la cellule c doit être capable de voir les états

$$\{\langle c+x\rangle_t \mid x \in (V^{t_c+t_s} + V)\}$$

qui sont tous contenus dans ce qu'elle connaît ou ce que connaissent ses voisines $(c+x_p)$ et $(c-x_n)$. □

Définition 2.2.4
Au temps t, on dira que la cellule c est d-correcte *(resp.* g-correcte*) si toutes les suppositions qu'elle fait concernant des états à sa droite (resp. gauche) sont justes. On dira qu'elle est* d-incorrecte *(resp.* g-incorrecte*) sinon.*

Remarque. Si au temps $(t_c + t_s + t)$ la cellule c est d-correcte cela signifie qu'elle connaît ou suppose connaître correctement tous les états

$$\{\langle c + x\rangle_t \mid x \in [\![-t_s x_n, (t_c + t_s + k)x_p]\!]\}$$

Lemme 2.2.3
Au temps $(t_c + t_s)$ toutes les cellules $c \geq (l - t_s x_p - x_0)$ sont d-correctes.

Preuve : Ces cellules supposent que l'état initial des cellules $c \geq l$ est B, ce qui est vrai. On rappelle que par définition de x_0, $(t_s x_p + x_0)$ est le plus petit entier positif n'appartenant pas à $V^{t_c+t_s}$ (voir figure 2.5). □

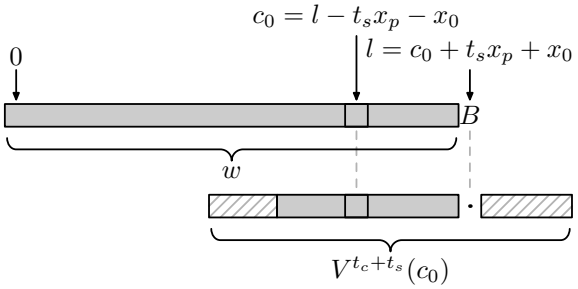

FIGURE 2.5 – Preuve du lemme 2.2.3

Lemme 2.2.4
Si la cellule $(c + x_p)$ est d-correcte au temps $(t_c + t_s + t)$ alors la cellule c est d-correcte au temps $(t_c + t_s + t + 1)$.

Preuve : Nous avons vu que lorsque la cellule c applique la règle locale de l'automate $\mathcal{A}$ aux informations qu'elle a elle donne une plus grande priorité aux informations portées par sa voisine $(c+x_p)$ en cas de conflit. Pour calculer correctement les états

$$\{\langle c + x\rangle_{t+1} \mid x \in [\![t_s x_p, (t_c + t_s + k)x_p]\!]\}$$

la cellule c doit connaître correctement les états

$$\{\langle c + x + v\rangle_t \mid v \in V, x \in [\![t_s x_p, (t_c + t_s + k)x_p]\!]\}$$

qui sont tous inclus dans

$$\{\langle c + x_p + x\rangle_t \mid x \in [\![-t_s x_n, (t_c + t_s + k)x_p]\!]\}$$

Ainsi, c a toutes les informations correctes dont elle a besoin au temps $(t_c + t_s + t)$ pour être d-correcte au temps $(t_c + t_s + 1)$. □

Lemme 2.2.5
Au temps (t_c+t_s+t), toutes les cellules $c \geq l-(t_s+t)x_p-x_0$ sont d-correctes.

Preuve : C'est une conséquence immédiate des lemmes 2.2.3 et 2.2.4. □

Lemme 2.2.6
Toutes les cellules $c \leq 0$ (et donc principalement l'origine, qui est la seule qui va nous intéresser) sont g-correctes en tout temps $t \geq t_c + t_s$.

Preuve : Il est immédiat que toutes ces cellules sont g-correctes au temps (t_c+t_s) puisqu'il n'y a que des états B à leur gauche. Par ailleurs, de même que précédemment, si une cellule est g-correcte au temps t et que toutes ses voisines à sa gauche le sont aussi, alors au temps $(t+1)$ la cellule est toujours g-correcte (on ne se soucie pas ici de la propagation de cette propriété, on veut juste vérifier qu'elle se conserve d'un temps sur l'autre). □

Lemme 2.2.7
Si la longueur du mot w est $l \geq (t_s+1)x_p + x_0 + 1$, alors le temps réel correspondant à ce mot selon le voisinage V vérifie

$$\mathrm{TR}_V(l) \geq t_c + \left\lfloor \frac{l-1-x_0}{x_p} \right\rfloor + 1$$

Preuve : Soit

$$\alpha = \left\lfloor \frac{l-1-x_0}{x_p} \right\rfloor$$

On a donc $\alpha x_p + x_0 \leq l-1$. De plus, puisque $l \geq (t_s+1)x_p + x_0 + 1$ on a $\alpha \geq t_s$. Ainsi, d'après le théorème 2.2.1 on a

$$(\alpha x_p + x_0) \notin V^{t_c+\alpha}$$

ce qui implique que $\mathrm{TR}_V(l) \geq t_c + \alpha + 1$. □

Lemme 2.2.8
Si la longueur du mot w est $l \geq (t_s+1)x_p + x_0 + 1$, alors au temps $\mathrm{TR}_V(l)$ l'automate $\mathcal{A}'$ est capable de déterminer si w appartient au langage L ou non.

Preuve : D'après le lemme 2.2.5, on sait que l'origine est d-correcte au temps

$$t \geq t_c + \left\lceil \frac{l-x_0}{x_p} \right\rceil = t_c + \left\lfloor \frac{l-1-x_0}{x_p} \right\rfloor + 1$$

Ainsi, d'après le lemme 2.2.7 l'origine est d-correcte au temps $\mathrm{TR}_V(l)$. Comme par ailleurs l'origine est toujours g-correcte (lemme 2.2.6), au temps $\mathrm{TR}_V(l)$ l'origine connaît avec exactitude tous les états de

$$\{\langle c \rangle_{\mathrm{TR}_V(l)-t_c-t_s} \mid c \in [\![-(t_c+t_s+k)x_n, (t_c+t_s+k)x_p]\!]\}$$

qui contiennent toute l'information nécessaire pour calculer l'état $\langle 0 \rangle_{\mathrm{TR}+k}$. L'automate $\mathcal{A}'$ est donc capable de déterminer si w appartient à L ou non. □

Le nombre de mots de longueur $l \leq (t_s + 1)x_p + x_0$ étant fini, on peut considérer que l'automate $\mathcal{A}'$ traite tous ces mots comme des cas particuliers en temps réel. Le lemme 2.2.8 achève donc la preuve du théorème 2.2.2.

2.2.3 Équivalences de voisinages au sens du temps réel

Définition 2.2.5
Pour tout voisinage V et tout entier naturel n on note $L^n_{\mathrm{TR}}(V)$ l'ensemble des langages sur l'alphabet $[\![0, n-1]\!]$ qui sont reconnus en temps réel par un automate cellulaire fonctionnant sur le voisinage V. On définit également

$$L_{\mathrm{TR}}(V) = \bigcup_{n\in\mathbb{N}} L^n_{\mathrm{TR}}(V)$$

On a le théorème suivant :

Théorème 2.2.3
Soit V un voisinage en dimension 1.

1. *Si V est complet, $L_{\mathrm{TR}}(V) = L_{\mathrm{TR}}(V_{\mathrm{std}})$;*
2. *Si V est semi-incomplet, $L_{\mathrm{TR}}(V) = L_{\mathrm{TR}}(\{0,1\})$.*

Toute la suite de cette section sera consacrée à la preuve de ce théorème.

Proposition 2.2.5
Pour tout voisinage semi-complet V, si l'on pose

$$-x_n = \min V \qquad \text{et} \qquad x_p = \max V$$

alors

$$L_{\mathrm{TR}}(V) = L_{\mathrm{TR}}([\![-x_n, x_p]\!])$$

Preuve : Il est immédiat qu'un automate fonctionnant sur $[\![-x_n, x_p]\!]$ est capable de simuler le fonctionnement d'un automate fonctionnant sur V sans aucune perte de temps. Par ailleurs, la proposition 2.2.3 nous assure que le temps réel sur $[\![-x_n, x_p]\!]$ diffère du temps réel sur V d'au plus t_c pour tout mot et donc d'après le théorème 2.2.2 on a $L_{\mathrm{TR}}(V) \subseteq L_{\mathrm{TR}}([\![-x_n, x_p]\!])$.

Réciproquement, si un langage L est reconnu en temps réel par un automate $\mathcal{A}$ fonctionnant sur $[\![-x_n, x_p]\!]$ il est facile de construire un automate $\mathcal{A}'$ fonctionnant sur V qui stocke des informations sur chacune de ses cellules pendant les t_c premières générations de telle sorte qu'au temps t_c chaque cellule c connaisse exactement tous les états initialement présents sur les cellules $\{(c + x) \mid x \in [\![-x_n, x_p]\!]\}$. À partir de cet instant chaque cellule peut appliquer la règle de transition de $\mathcal{A}$ à au groupe d'états qu'elle contient puisque

$$[\![-x_n, x_p]\!] + [\![-x_n, x_p]\!] = V + [\![-x_n, x_p]\!]$$

L'automate ainsi construit reconnaît donc L en temps $(\mathrm{TR}_{[\![-x_n,x_p]\!]} + t_c)$ ce qui est inférieur à $\mathrm{TR}_V + t_c$. Le théorème 2.2.2 nous donne alors l'inclusion $L_{\mathrm{TR}}([\![-x_n, x_p]\!]) \subseteq L_{\mathrm{TR}}(V)$. □

Proposition 2.2.6
Pour tout voisinage semi-complet V, on a $L_{\mathrm{TR}}(V) = L_{\mathrm{TR}}(V^k)$.

Preuve : La proposition précédente (2.2.5) nous permet de nous limiter aux voisinages connexes. Il est clair que pour tout voisinage connexe V, $\mathrm{TR}_V \geq k(\mathrm{TR}_{V^k} - 1)$. Par ailleurs, un automate fonctionnant sur V peut simuler une étape d'un automate fonctionnant sur V^k en k étapes ($(k-1)$ étapes de stockage, puis une étape pour appliquer la règle de transition) ce qui signifie que tout langage reconnu en temps TR_{V^k} sur V^k est reconnu en temps $k\mathrm{TR}_{V^k} \leq \mathrm{TR}_V + k$ sur V. D'après le théorème 2.2.2 on a donc $L_{\mathrm{TR}}(V^k) \subseteq L_{\mathrm{TR}}(V)$.

Réciproquement, on a $\mathrm{TR}_{V^k} \geq \lfloor \mathrm{TR}_V/k \rfloor$. Puisqu'un automate cellulaire fonctionnant sur le voisinage V^k peut simuler en une seule étape k étapes d'un automate fonctionnant sur V, tout langage reconnu en temps réel sur V est reconnu en temps $\lceil \mathrm{TR}_V/k \rceil \leq \mathrm{TR}_{V^k} + 1$ sur V^k. Encore une fois, le théorème 2.2.2 permet de conclure. □

Proposition 2.2.7
Pour tous $x_n \leq x'_n \in \mathbb{N}$ et $x_p \in \mathbb{N}^*$ on a

$$L_{\mathrm{TR}}([\![-x_n, x_p]\!]) \subseteq L_{\mathrm{TR}}([\![-x'_n, x_p]\!])$$

Preuve : Il suffit ici de remarquer que la fonction temps réel est la même pour les deux voisinages et que tout automate fonctionnant sur le plus petit voisinage peut être directement simulé par un automate fonctionnant sur le plus grand sans perte de temps. □

Proposition 2.2.8
Pour tous $x_n, x_p \in \mathbb{N}^*$ on a

$$L_{\mathrm{TR}}([\![-x_n, x_p]\!]) = L_{\mathrm{TR}}([\![-2x_n, x_p]\!])$$

Preuve : L'inclusion $L_{\mathrm{TR}}([\![-x_n, x_p]\!]) \subseteq L_{\mathrm{TR}}([\![-2x_n, x_p]\!])$ découle immédiatement de la proposition précédente. Intéressons-nous à l'inclusion réciproque.

Notons $V_1 = [\![-x_n, x_p]\!]$ et $V_2 = [\![-2x_n, x_p]\!]$. Soit L un langage appartenant à $L_{\mathrm{TR}}(V_2)$. Nous avons vu qu'il existe alors un automate cellulaire $\mathcal{A}$ fonctionnant sur V_2 reconnaissant L en temps réel tel que les cellules se trouvant à gauche de l'origine ne participent pas au calcul (elles restent toujours dans l'état B). Nous devons alors construire un automate cellulaire $\mathcal{A}'$ fonctionnant sur le voisinage V_1 capable de reconnaître L aussi rapidement que $\mathcal{A}$ (les deux voisinages partagent la même fonction temps réel).

La technique employée ici consiste à « compresser » la configuration initiale sur la diagonale ($c = t$) de telle sorte que chaque cellule contienne alors $(x_p + 1)$ états de $\mathcal{A}$. Puis on montre que l'on peut effectuer la simulation de $\mathcal{A}$ en maintenant cette compression (chaque colonne simule l'évolution de $(x_p + 1)$ cellules de $\mathcal{A}$). Le fonctionnement global de la transformation est illustré sur la figure 2.6 (on a représenté une compression d'un facteur 2).

Sur cette figure on a représenté à gauche le fonctionnement normal de $\mathcal{A}$ sous la forme d'un diagramme espace-temps et représenté en gris les colonnes qui correspondent chacune à l'évolution d'une cellule de $\mathcal{A}$. La ligne pointillée délimite le cône des états qui peuvent avoir une influence sur l'origine au temps réel (puisque l'on a pris $x_p = 2$, cette ligne est de pente $1/2$). Sur la partie droite on a représenté la transformation (toujours sous la forme d'un diagramme espace-temps) : dans un premier temps toutes les lettres du mot en entrée se déplacent vers la gauche jusqu'à atteindre la diagonale (dessinée en noir). Pendant cette première phase la règle de $\mathcal{A}$ n'est pas appliquée. La diagonale contient donc une représentation compressée de l'entrée (notons que la diagonale ne correspond à aucun instant, mais la première lettre du mot est sur l'origine au temps 0, puis les $(x_p + 1)$ lettres suivantes sont sur la cellule 1 au temps 1 et de manière générale les lettres indexées de $((x_p + 1)i - x_p)$ à $(x_p + 1)i$ se trouvent sur la cellule i au temps i).

Sur l'autre partie du diagramme espace-temps (au-dessus de la diagonale) on effectue ensuite la simulation de $\mathcal{A}$ à vitesse réelle. Cependant on simule $(x_p + 1)$ colonnes de $\mathcal{A}$ sur une unique colonne de $\mathcal{A}'$ et les simulations des colonnes de $((x_p + 1)i - x_p)$ à $(x_p + 1)i$ sont décalées dans le temps (puisque leur simulation commence au temps i sur la cellule i).

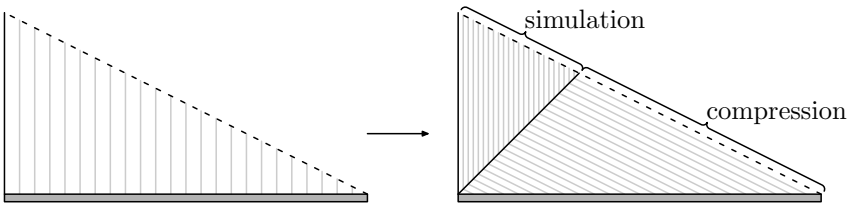

FIGURE 2.6 – Schéma général de la transformation permettant de montrer la proposition 2.2.8 (ici $x_p = 2$).

La figure 2.7 illustre le détail de la simulation (au-dessus de la diagonale).

Sur cette figure on a illustré le cas d'un mot en entrée de taille 8 et l'on va supposer que $x_n = 2$ et $x_p = 1$ (pour l'exemple). On ne s'intéresse qu'à la partie au-dessus de la diagonale puisque les états en dessous sont simplement les lettres du mot qui se déplacent vers la gauche. La figure est un diagramme espace-temps de $\mathcal{A}'$ et donc les cases grises dessinées sont les cellules de $\mathcal{A}'$, chaque ligne représentant la configuration à un instant donné (le temps de $\mathcal{A}'$ est indiqué par les repères de temps à droite de la figure).

Cependant, chaque cercle sur la figure représente un état de $\mathcal{A}$, et la transformation que l'on a appliquée (compression sur la diagonale) est telle que l'on « retrouve » le diagramme espace-temps de $\mathcal{A}$ dans celui de $\mathcal{A}'$. Ainsi, comme il a déjà été dit, chaque colonne au dessus de la diagonale représente une colonne du diagramme de $\mathcal{A}$ c'est-à-dire l'évolution d'une unique cellule dans le fonctionnement de $\mathcal{A}$. Les flèches verticales au dessus de l'image indiquent à quelle cellule de $\mathcal{A}$ correspond chaque colonne d'états. On voit notamment que chaque cellule de $\mathcal{A}'$ simule le fonctionnement de deux cellules de $\mathcal{A}$ (dans le cas général, il y a $(x_p + 1)$ colonnes par cellule). Enfin, on retrouve les configurations de $\mathcal{A}$ sous forme de de diagonales sur le diagramme de $\mathcal{A}'$. Nous savions déjà que le mot en entrée se trouvait sur la

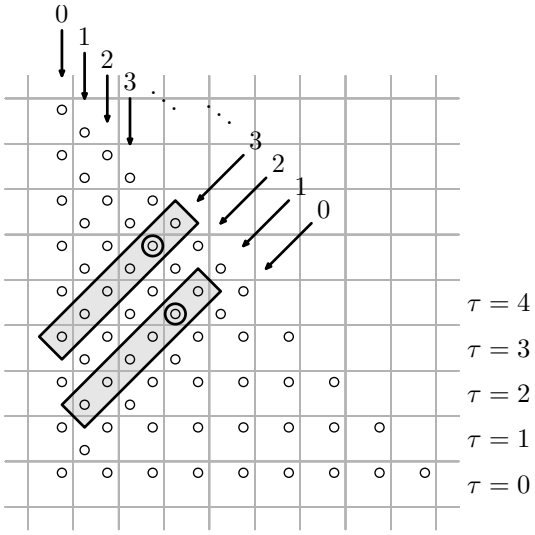

FIGURE 2.7 – Détail de la transformation permettant de montrer la proposition 2.2.8 (ici $x_p = 1$).

première diagonale, mais la configuration au temps 1 se trouve également sur la diagonale juste au dessus, et la configuration de $\mathcal{A}$ au temps 2 sur la diagonale du dessus... Les configurations de $\mathcal{A}$ sont indiquées par les flèches diagonales sur la figure.

On voit ainsi par exemple sur la figure que les deux états entourés correspondent respectivement à l'état de la cellule 4 au temps 3 et à celui de la cellule 5 au temps 1 dans le fonctionnement de $\mathcal{A}$.

Il est maintenant facile de vérifier que la simulation telle que nous l'avons décrite jusqu'ici (et telle qu'elle est représentée sur la figure 2.7) peut être réalisée par l'automate $\mathcal{A}'$ fonctionnant sur le voisinage V_1. Lorsque l'automate veut calculer un nouvel état, il peut être dans l'une de deux situations (dans le cas général il y a $(x_p + 1)$ situations possibles, mais le raisonnement est similaire). Soit c'est un état de $\mathcal{A}$ qui se trouve « en bas à gauche » d'une cellule de $\mathcal{A}'$ soit c'est un état qui se trouve « en haut à droite » d'une cellule (après compression).

Le premier cas est représenté par l'état entouré le plus en bas sur la figure 2.7. Les états dans le rectangle gris sont ceux que la cellule sur laquelle se trouve l'état encerclé doit connaître afin d'appliquer la règle de transition de $\mathcal{A}$ et calculer l'état suivant. Puisque l'on a considéré ici que $x_n = 2$, V_2 s'étend sur 4 cases vers la gauche. Les états sur la gauche de l'état encerclé correspondant à la même génération de $\mathcal{A}$ se trouvent donc tous sur les deux cellules à gauche de la cellule qui porte l'état encerclé (grâce à la compression par un facteur 2). Techniquement, ces états ne sont plus présents sur les voisines de la cellule mais on peut modifier notre automate pour que chaque cellule se souvienne des 2 derniers états qu'elle portait (des x_n derniers dans le cas général) et donc les deux voisines de la cellule contiennent les informations que la cellule doit connaître sur la gauche de l'état encerclé. Sur la droite,

l'état dont a besoin la cellule est porté par la cellule elle-même et donc la cellule peut calculer l'état qui succède à l'état encerclé dans l'évolution de $\mathcal{A}$. Notons que la cellule n'a pas eu besoin des informations portées par sa voisine de droite pour calculer cet état.

Le second cas est un tout petit peu plus compliqué, et il est illustré par le second état entouré sur la figure 2.7 (celui du dessus). Les informations que la cellule doit trouver à gauche de l'état s'obtiennent de la même manière en regardant les x_n voisines sur la gauche après compression. Il faut cependant trouver l'état qui se trouvera au temps suivant dans la partie inférieure gauche de la voisine de droite... Cet état n'est pas encore présent sur la configuration de $\mathcal{A}'$, mais la cellule a toute l'information dont elle a besoin pour calculer elle-même cet état : en effet, cet état est dans le cas observé précédemment et il peut donc être obtenu sans connaître les informations portées par la voisine de droite de la cellule qui le porte (selon la méthode exposée au paragraphe précédent). La cellule sur laquelle se trouve l'état entouré que l'on est en train de considérer peut donc calculer l'état qui se trouve dans la partie droite du rectangle gris et donc calculer l'état qui succède à l'état encerclé dans l'évolution de $\mathcal{A}$. Dans le cas général, les informations nécessaires se trouvent toutes sur les x_p voisines à droite de la cellule considérée.

Dans tous les cas, nous avons donc montré que $\mathcal{A}'$ pouvait simuler le fonctionnement de $\mathcal{A}$ sur chacune des colonnes après compression de l'entrée sur la diagonale. Puisque la colonne correspondant à l'origine n'est pas affectée par la transformation l'évolution de l'origine de $\mathcal{A}'$ est la même que celle de $\mathcal{A}$ et donc $\mathcal{A}$ reconnaît le mot w en temps réel. □

Proposition 2.2.9
Soient $x_n, x'_n, x_p \in \mathbb{N}^*$. On a alors

$$L_{\mathrm{TR}}([\![x_n, x_p]\!]) = L_{\mathrm{TR}}([\![x'_n, x_p]\!])$$

Preuve : On suppose que $x_n < x'_n$. La proposition 2.2.7 donne

$$L_{\mathrm{TR}}([\![x_n, x_p]\!]) \subseteq L_{\mathrm{TR}}([\![x'_n, x_p]\!])$$

Par ailleurs, il existe $k \in \mathbb{N}$ tel que $2^k x_n > x'_n$. Les propositions 2.2.7 et 2.2.8 donnent alors

$$L_{\mathrm{TR}}([\![-x'_n, x_p]\!]) \subseteq L_{\mathrm{TR}}([\![-2^k x_n, x_p]\!]) = L_{\mathrm{TR}}([\![x_n, x_p]\!])$$

□

Nous sommes à présent en mesure de prouver le théorème 2.2.3. Soit V un voisinage complet. Notons $-x_n = \min V$ et $x_p = \max V$. La proposition 2.2.1 nous assure que x_n et x_p sont strictement positifs.

On a alors (en indiquant pour chaque relation obtenue la proposition que l'on applique) :

$$\begin{aligned} L_{\mathrm{TR}}(V) &\overset{2.2.5}{=} L_{\mathrm{TR}}([\![-x_n, x_p]\!]) \\ &\overset{2.2.7}{=} L_{\mathrm{TR}}([\![-x_p, x_p]\!]) \\ &\overset{2.2.6}{=} L_{\mathrm{TR}}(\{-1, 0, 1\}) \end{aligned}$$

Par ailleurs, pour tout voisinage V semi-incomplet, la proposition 2.2.2 nous donne $\min V = 0$. En notant $x_p = \max V$, on a :

$$\begin{aligned} L_{\mathrm{TR}}(V) &\overset{2.2.5}{=} L_{\mathrm{TR}}(\llbracket 0, x_p \rrbracket) \\ &\overset{2.2.6}{=} L_{\mathrm{TR}}(\{0,1\}) \end{aligned}$$

Cette dernière égalité achève la preuve du théorème 2.2.3.

2.3 Généralisation en dimension quelconque

Nous venons de montrer qu'en dimension 1 tous les voisinages complets permettent de reconnaître les mêmes langages en temps réel. Ce résultat n'est cependant pas vrai en dimension 2 puisque l'on sait par exemple qu'il existe un langage reconnu en temps réel par un automate cellulaire fonctionnant sur le voisinage de Von Neumann qui n'est reconnu en temps réel par aucun automate cellulaire fonctionnant sur le voisinage de Moore [17].

Toutefois, le théorème 2.2.2 qui est à l'origine du résultat en dimension 1 peut être généralisé en dimension quelconque. Il s'énonce alors de la manière suivante :

Théorème 2.3.1
En dimension d quelconque, étant donné un voisinage complet V, tout langage reconnu en temps réel par un automate fonctionnant sur l'enveloppe convexe de V est reconnu en temps réel par un automate cellulaire fonctionnant sur le voisinage V.

En reprenant les notations introduites dans la section précédente on a donc :

$$L_{\mathrm{TR}}(\mathrm{CH}(V)) \subseteq L_{\mathrm{TR}}(V)$$

Ce théorème est un corollaire immédiat du résultat suivant, obtenu en collaboration avec Martin Delacourt dans le cadre de son stage de fin d'année en licence d'informatique à l'École Normale Supérieure de Lyon :

Proposition 2.3.1
En dimension d quelconque, étant donné un voisinage V complet et un automate cellulaire $\mathcal{A}$ d'états $\mathcal{Q}$ fonctionnant sur $\mathrm{CH}(V)$, il existe un automate cellulaire $\mathcal{A}'$ d'états $\mathcal{Q}'$ fonctionnant sur V et une fonction de traduction

$$\sigma : \mathcal{Q}' \to \mathcal{Q}$$

tels que, partant d'une configuration $\mathfrak{C}_w$ correspondant à un mot $w \in {\Sigma^*}^d$, à tout temps $t \geq \mathrm{TR}_V(w)$ l'origine de $\mathcal{A}'$ soit dans un état q' et $\sigma(q')$ est l'état dans lequel se trouve l'origine dans l'évolution de $\mathcal{A}$ au temps t à partir de $\mathfrak{C}_w$, soit plus formellement :

$$\forall t \geq \mathrm{TR}_V(w), \quad \sigma(\mathcal{A}'^t(\mathfrak{C}_w)(0)) = \mathcal{A}^t(\mathfrak{C}_w)(0)$$

Cette proposition signifie que d'une certaine manière, bien que l'automate $\mathcal{A}'$ fonctionne sur un voisinage plus restreint que celui de $\mathcal{A}'$, après $\mathrm{TR}_V(w)$ générations l'origine de $\mathcal{A}'$ contient toute l'information (et peut-être même plus) de l'origine de $\mathcal{A}$ sur la même entrée.

Ainsi tout langage reconnu en temps $f \geq \mathrm{TR}_V$ sur $\mathrm{CH}(V)$ est reconnu en temps f également sur V. Le théorème 2.3.1 est alors bien une conséquence immédiate de cette proposition puisque si un langage est reconnu en temps $\mathrm{TR}_{\mathrm{CH}(V)}$ par un automate fonctionnant sur $\mathrm{CH}(V)$ il est reconnu en temps TR_V sur $\mathrm{CH}(V)$ (car $\mathrm{TR}_V \geq \mathrm{TR}_{\mathrm{CH}(V)}$) et donc également en temps TR_V par un automate fonctionnant sur V.

La suite de cette section sera consacrée à la preuve de la proposition 2.3.1. Afin de simplifier les notations et pour rendre la preuve plus claire nous ne traiterons ici que le cas de la dimension 2. Le cas général en dimension d est cependant tout à fait analogue.

La preuve est organisée de manière similaire à la preuve du théorème 2.2.2. Elle consiste donc à considérer un langage $L \subseteq \Sigma^{**}$ reconnu en temps réel par un automate $\mathcal{A}$ fonctionnant sur le voisinage $\mathrm{CH}(V)$ puis construire un automate cellulaire $\mathcal{A}'$ fonctionnant sur le voisinage V qui simule le fonctionnement de $\mathcal{A}$ de telle sorte que pour tout mot w, l'origine sur l'automate $\mathcal{A}'$ soit capable au temps $\mathrm{TR}_V(w)$ de déterminer l'état dans lequel se trouverait l'origine de $\mathcal{A}$ au même temps $\mathrm{TR}_V(w)$ sur la même entrée.

Comme précédemment, on va considérer un mot $w \in \Sigma^{**}$ et noter $\langle c \rangle_t$ l'état dans lequel se trouve la cellule c au temps t dans l'évolution de $\mathcal{A}$ à partir de la configuration initiale correspondant à w. Nous allons alors décrire le fonctionnement de $\mathcal{A}'$ sur l'entrée w.

2.3.1 Croissance des voisinages en dimension 2

L'objectif de cette section est d'obtenir une généralisation du théorème 2.2.1 en dimension 2.

Étant donné un voisinage complet V en dimension 2, on définit l'entier t_c par

$$t_c = \min\{t \in \mathbb{N} \mid \mathrm{CH}(V) \subseteq V^{t+1}\}$$

On a alors la proposition suivante :

Proposition 2.3.2
Pour tout entier $t \in \mathbb{N}$,

$$\mathrm{CH}(V)^t \subseteq V^{t_c+t} \subseteq \mathrm{CH}(V)^{t_c+t}$$

La preuve est identique à celle de la proposition 2.2.3 en dimension 1.

Cela signifie que le voisinage V^{t+t_c} est de la forme illustrée sur la figure 2.8, c'est-à-dire qu'il est constitué d'un « noyau » polygonal $\mathrm{CH}(V)^t$ plein (représenté en gris) entouré d'une « couronne » non pleine d'épaisseur constante t_c. Sur la figure on a représenté l'enveloppe convexe de V en gris foncé (au centre). Notons que tous les sommets de la couronne appartiennent bien à V^{t_c+t} puisque ce sont les vecteurs de la forme $(t_c + t)s_i$ où les s_i sont les sommets de $\mathrm{CH}(V)$, qui appartiennent bien à V (on a représenté le point $(t + t_c)u$ pour un sommet u de V).

Considérons maintenant deux sommets consécutifs s_i et s_{i+1} de $\mathrm{CH}(V)$ et intéressons-nous au cône de V^{t_c+t} délimité par les deux demi-droites $[Os_i)$ et $[Os_{i+1})$ (voir figure 2.9).

Ce cône est constitué d'un triangle plein délimité par les vecteurs $t.s_i$ et $t.s_{i+1}$ et d'une « bande » de largeur t_c non pleine (elle est de largeur $t_c.s_i$ d'un côté et $t_c.s_{i+1}$ de l'autre).

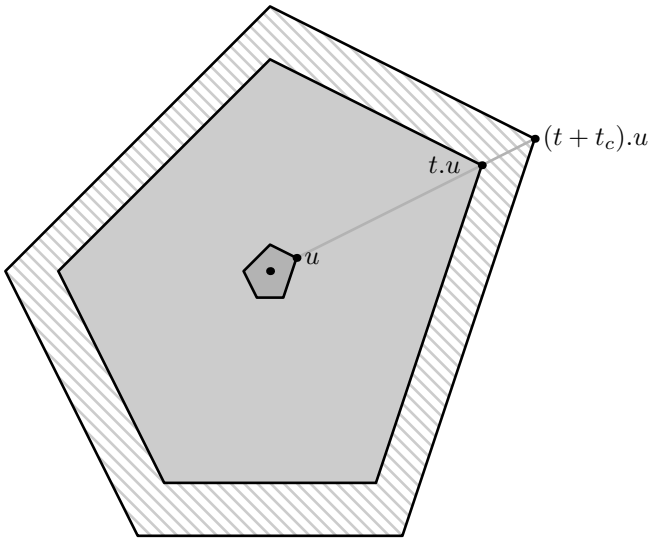

FIGURE 2.8 – La forme générique de V^{t_c+t} : tous les points dans la zone en gris appartiennent à V^{t_c+t}, mais ce n'est pas nécessairement le cas de tous les points dans la zone hachurée.

Considérons maintenant une « fenêtre » de forme trapézoïdale de base inférieure $(s_{i+1} - s_i)$, de base supérieure $t_c(s_{i+1} - s_i)$ et de côtés $t_c.s_i$ et $t_c.s_{i+1}$ (comme illustrée en pointillés gras sur la figure). On définit de plus le vecteur $h_i = s_{i+1} - s_i$.

La position de la fenêtre sera donnée par son sommet inférieur gauche (sur la figure, la fenêtre est donc en $(t.s_i + 3.h_i)$). On ne s'intéressera qu'aux fenêtres positionnées en des points de la forme $(t.s_i + j.h_i)$ avec $j \in [\![0, t-1]\!]$ (les différentes positions acceptables de la fenêtre sont représentées en traits pointillés sur la figure).

Puisque le voisinage V^{t+t_c} n'est pas « plein » sur la bande de largeur t_c, cela signifie que lorsque l'on place la fenêtre en un point $(t.s_i+j.h_i)$, certains points de la fenêtre sont dans V^{t+t_c} et certains ne le sont pas. On appellera « remplissage » (faute d'avoir trouvé un mot plus élégant) de la fenêtre placée en un point $(t.s_i + j.h_i)$ l'ensemble des points à l'intérieur du trapèze qui appartiennent à V^{t_c+t}.

De même que l'on avait montré que l'ombre droite de V^{t+t_c} était croissante au sens de l'inclusion lorsque t augmentait en dimension 1 (preuve de la proposition 2.2.4) on peut ici montrer la proposition suivante :

Proposition 2.3.3
Le remplissage observé dans la fenêtre placée au point $(t.s_i + j.h_i)$ sur le cône (s_i, s_{i+1}) du voisinage V^{t_c+t} est inclus dans les remplissages observés dans les fenêtres (de mêmes dimensions) placées en $((t+1)s_i+j.h_i)$ et $((t+1)s_i+(j+1)h_i)$ sur le cône (s_i, s_{i+1}) du voisinage V^{t_c+t+1}.

Preuve : En effet, puisque $(V^{t_c+t} + s_i)$ et $(V^{t_c+t} + s_{i+1})$ sont tous deux inclus dans V^{t_c+t+1}, si un point $x \in V^{t_c+t}$ se trouve dans la fenêtre placée en $(t.s_i + j.h_i)$ alors le point $(x + s_i) \in V^{t_c+t+1}$ se trouve à la même position

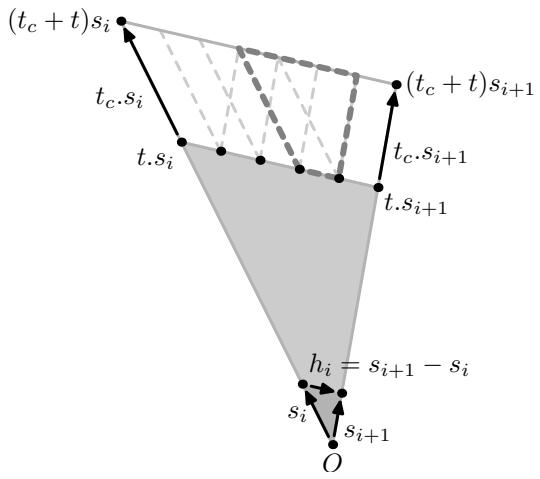

FIGURE 2.9 – Les trapèzes élémentaires sur le cône (s_i, s_{i+1}) du voisinage V^{t_c+t}.

dans la fenêtre placée en $((t+1)s_i + j.h_i)$ (on effectue une translation selon s_i) et le point $(x + s_{i+1}) \in V^{t_c+t+1}$ se trouve également à la même position dans la fenêtre placée en $((t+1)s_i + (j+1)h_i)$ (on effectue une translation selon s_{i+1}, et l'on rappelle que $s_i + h_i = s_{i+1}$). □

La figure 2.10 illustre ce raisonnement.

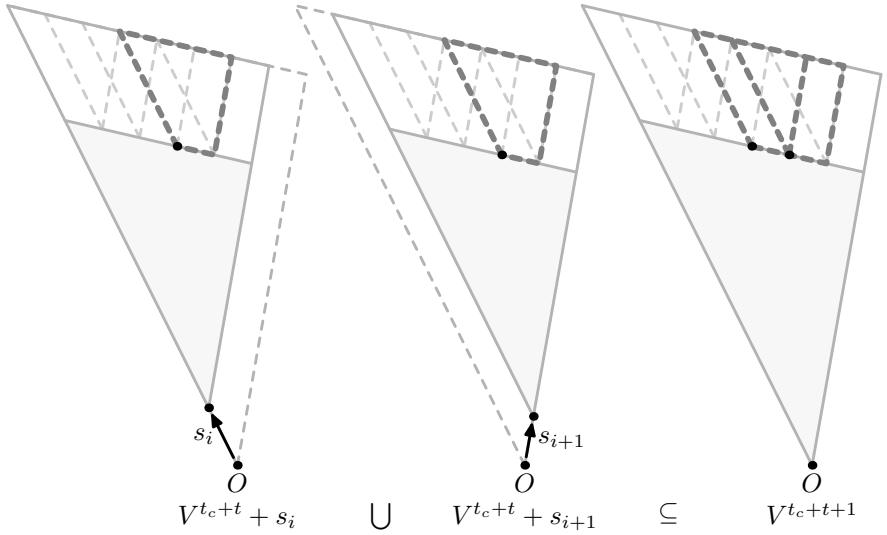

FIGURE 2.10 – Les deux fenêtres trapézoïdales indiquées sur V^{t_c+t+1} (à droite) sont plus remplies que la fenêtre qui leur correspond dans V^{t_c+t}.

La proposition 2.3.3 implique que le remplissage observé dans les fenêtres au temps (t_c+t+1) est plus grand (au sens de l'inclusion) que le remplissage observé dans les fenêtres correspondantes au temps (t_c+t). Cela signifie notamment que l'on atteint à un moment un remplissage maximum, c'est-à-dire qu'il existe un

temps t_m et un entier j_m tel que le remplissage observé dans la fenêtre placée en $(t_m.s_i + j_m.h_i)$ contient le remplissage observé dans toutes les autres fenêtres possibles.

Puisque ce remplissage est maximum et que les remplissages suivants sont plus grands, c'est que l'on atteint un point fixe. En d'autres termes, le remplissage observé sur les fenêtres en $((t_m + 1)s_i + j_m.h_i)$ et $((t_m + 1)s_i + (j_m + 1)h_i)$ est identique, et il en va de même pour toutes les fenêtres suivantes.

Au temps t_m, on peut alors séparer la bande en trois parties comme illustré sur le haut de la figure 2.11. À partir de cet instant, les bandes suivantes (correspondant à des temps $t \geq t_m$) peuvent également être séparées en trois parties disjointes telles que les parties gauche et droite (en gris clair sur la figure) aient toujours la même taille, et tous les remplissages observés sur la partie centrale soient identiques (la partie centrale de la bande est représentée en gris foncé, les différentes positions des fenêtres sont indiquées en pointillés, elles correspondent toutes à des remplissages maximaux identiques), le centre de la bande est donc périodique (voir la partie basse de la figure 2.11).

Si l'on travaille alors sur les parties gauche et droite de la bande, on peut appliquer le même raisonnement que précédemment pour montrer que le remplissage observé sur ces deux parties (qui ont toujours les mêmes dimensions) est croissant lorsque t augmente. Il atteint donc également un maximum et reste constant à partir de cet instant.

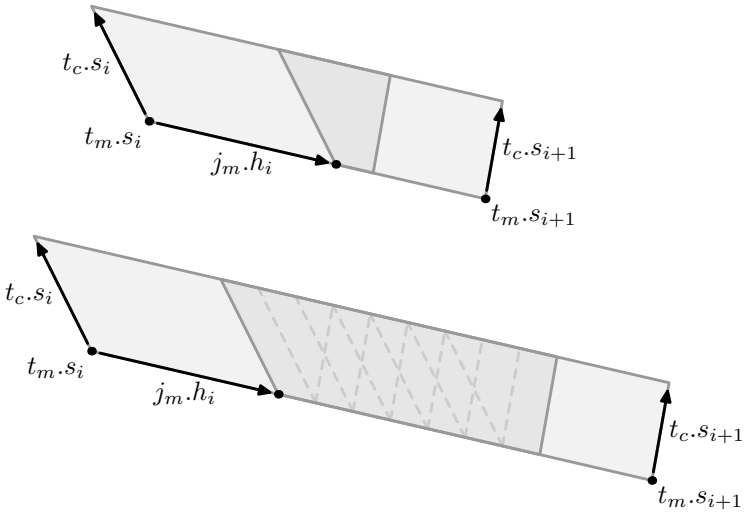

FIGURE 2.11 – Découpage de la bande en trois parties au temps t_m (en haut) et pour tout temps $t \geq t_m$ (en bas).

On a donc montré qu'il existait un temps t_i tel que pour tout $t \in \mathbb{N}$, les parties gauche et droite de la bande du cône (s_i, s_{i+1}) du voisinage V^{t_s+t} sont constantes et la partie centrale est une superposition de remplissages (en forme de trapèzes) tous identiques. Si l'on considère maintenant un entier t_s supérieur ou égal à tous les t_i pour tous les couples de sommets consécutifs de V, on a montré l'équivalent en dimension 2 du théorème 2.2.1 :

Théorème 2.3.2
Pour tout voisinage complet V en dimension 2, en notant $(s_1, s_2, \ldots, s_p)$ les sommets de V naturellement ordonnés, il existe un entier t_s tel que :

- *pour tout $i \in [\![1, p]\!]$, il existe un ensemble $A_i \subseteq V^{t_s+t_c} \setminus V^{t_s}$,*
- *pour tout entier $i \in [\![1, p]\!]$, il existe un ensemble B_i inclus dans le trapèze de bases $(s_{i+1} - s_i)$ et $t_c(s_{i+1} - s_i)$ et de côtés $t_c.s_i$ et $t_c.s_{i+1}$ (les indices sont considérés modulo p, c'est-à-dire que $s_{p+1} = s_1$)*

et pour tout entier $t \in \mathbb{N}$, le voisinage $V^{t_c+t_s+t}$ est exactement l'union des ensembles

- $\mathrm{CH}(V)^{t_s+t}$,
- $(A_i + t.s_i)$ *pour tout* $i \in [\![1, p]\!]$,
- B_i *disposé de manière régulière (chaque copie est obtenue par translation selon h_i de la précédente) sur la bande supérieure du cône (s_i, s_{i+1}) pour couvrir ce qui n'a pas été couvert par les A_i.*

La forme générique de $V^{t_c+t_s+t}$ est illustrée sur la figure 2.12 pour deux valeurs différentes de t.

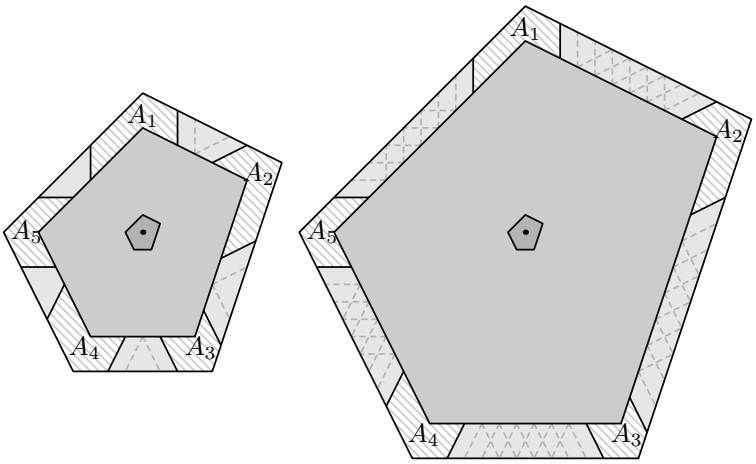

FIGURE 2.12 – La forme générique de $V^{t_c+t_s+t}$ (les remplissages des trapèzes dessinés en pointillés sont identiques sur une même bande).

Bien que difficile à énoncer, le théorème 2.3.2 est très important car il assure que tout voisinage complet, aussi irrégulier soit-il initialement, devient après un nombre fini d'itérations très régulier et périodique.

2.3.2 La propagation des hypothèses justes

Revenons maintenant aux automates $\mathcal{A}$ et $\mathcal{A}'$ sur l'entrée w.

Pour un certain entier t_0 suffisamment grand dépendant de V (nous décrirons par la suite comment choisir t_0) l'automate $\mathcal{A}'$ va passer les t_0 premières générations de son évolution à stocker toutes les informations qu'il peut sur chaque cellule.

Après t_0 générations, la cellule c de $\mathcal{A}'$ connaît donc tous les états $\{\langle c+x\rangle_0 \mid x \in V^{t_0}\}$. Si V^{t_0} est différent de $\mathrm{CH}(V)^{t_0}$, il y a des états dans $\mathrm{CH}(V)^{t_0}(c)$ que la cellule c ne connaît pas. Toutes les cellules vont alors supposer que les états qu'elles ne voient pas dans leur voisinage $\mathrm{CH}(V)^{t_0}$ sont B. Bien évidemment la plupart de ces suppositions sont fausses au temps t_0, mais pour les cellules suffisamment proches du bord du mot, certaines de ces hypothèses (dans certaines directions) sont justes.

Les cellules de $\mathcal{A}'$ vont alors appliquer la règle de transition de $\mathcal{A}$ aux états qu'elles portent (ceux qu'elles connaissent avec certitude et ceux qu'elles ont devinés). Au temps (t_0+t) chaque cellule de $\mathcal{A}'$ porte donc un ensemble d'états qu'elle pense être les états $\{\langle c+x\rangle_t \mid x \in \mathrm{CH}(V)^{t_0}\}$. Il faut maintenant montrer, comme dans le cas de la dimension 1, que les hypothèses justes que font certaines cellules initialement peuvent se propager à leurs voisines de telle sorte qu'après un certain temps que l'on calculera l'origine ne fasse que des hypothèses justes.

Si l'on note $\{s_1,\dots,s_p\}$ l'ensemble des sommets de V (que l'on suppose canoniquement ordonnés et dont les indices seront vus modulo p) on découpe le cône (s_i,s_{i+1}) du voisinage $\mathrm{CH}(V)^{t_0}$ en quatre parties comme indiqué sur la figure 2.13 :

- le triangle intérieur C_i de côtés $(t_0-t_c)s_i$ et $(t_0-t_c)s_{i+1}$ que l'on sait être inclus dans V^{t_0} ;
- une zone trapézoïdale T_i incluse dans la bande que nous avons étudiée dans la sous-section précédente dont les deux côtés sont parallèles aux côtés $[s_{i-1},s_i]$ et $[s_{i+1},s_{i+2}]$ de l'enveloppe convexe de V ;
- les deux parties S_i^d et S_{i+1}^g qui restent de chaque côté de la zone trapézoïdale centrale.

On définit de plus, pour tout i,

$$S_i = S_i^d \cup S_i^g$$

On choisit t_0 assez grand pour que, d'une part V^{t_0} soit de la forme « stabilisée » décrite par le théorème 2.3.2 (donc $t_0 \geq t_c+t_s$) et que d'autre part pour tout i le trapèze T_i ne s'étende pas au-delà de la partie centrale périodique de la bande du cône (s_i,s_{i+1}) sur V^{t_0} (lorsque t augmente, cette partie centrale devient arbitrairement large et il existe donc un temps t_0 tel que l'on peut choisir T_i dans cette zone).

La figure 2.14 illustre la forme générale d'un découpage total que l'on obtient dans le cas d'un voisinage dont l'enveloppe convexe est un pentagone.

Si l'on considère une cellule c de $\mathcal{A}$ au temps (t_0+t), elle connaît avec exactitude tous les états de $\{\langle c+x\rangle_t \mid x \in C_i\}$ pour tout i, mais pas nécessairement correctement les états dans les autres portions (mais elle fait quand même des hypothèses).

On dira qu'une cellule est *correcte* selon (s_i,s_{i+1}) si les hypothèses qu'elle fait dans la zone $(c+T_i)$ sont toutes correctes. On dira qu'une cellule est correcte selon s_i si elle est correcte selon (s_{i-1},s_i), (s_i,s_{i+1}) et que toutes les hypothèses qu'elle fait dans la zone $(c+S_i)$ sont correctes. Les figures 2.15 et 2.16 illustrent ces définitions. On a représenté dans chacun des cas en gris foncé les zones où toutes les hypothèses faites par la cellule doivent être justes pour que la cellule

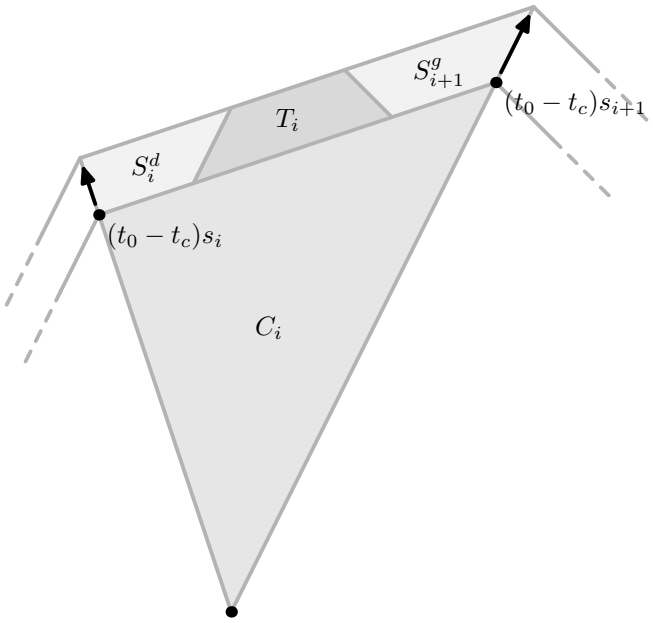

FIGURE 2.13 – Découpage du cône (s_i, s_{i+1}) du voisinage $\mathrm{CH}(V)^{t_0}$.

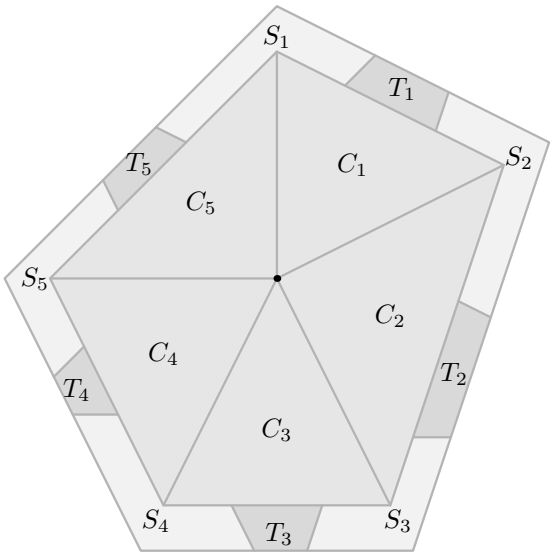

FIGURE 2.14 – Découpage de l'ensemble du voisinage $\mathrm{CH}(V)^{t_0}$.

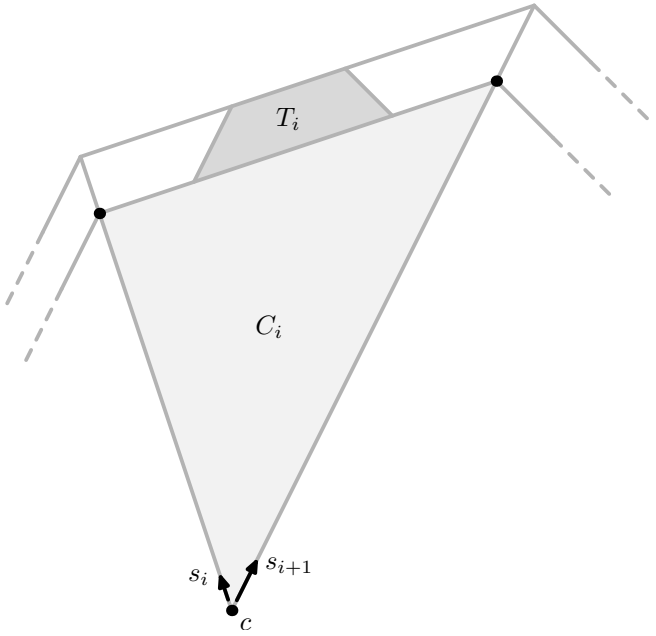

FIGURE 2.15 – Les hypothèses justes d'une cellule correcte selon (s_i, s_{i+1}).

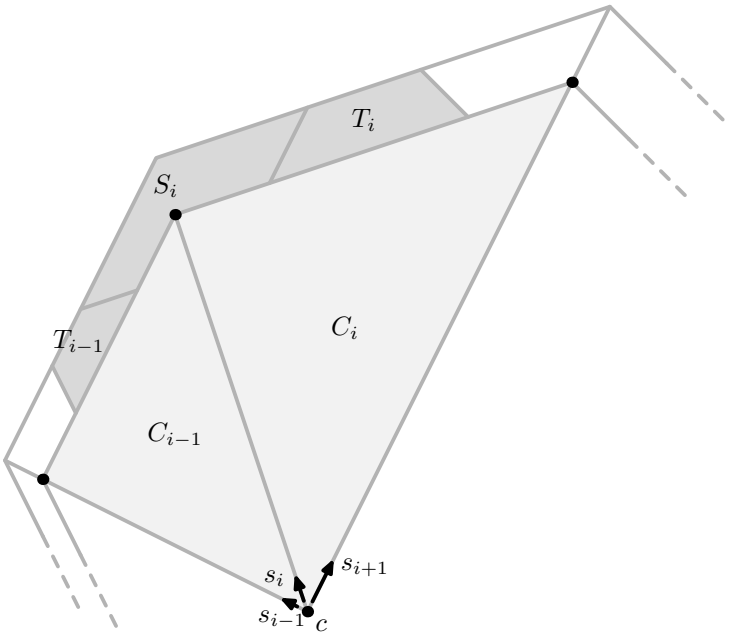

FIGURE 2.16 – Les hypothèses justes d'une cellule correcte selon s_i.

soit correcte (la cellule connaît par ailleurs tous les états dans les zones C_i représentées en gris clair).

Les définitions que l'on a données nous permettent alors de montrer les deux lemmes suivants :

Lemme 2.3.1
Si au temps (t_0+t) les cellules $(c+s_i)$ et $(c+s_{i+1})$ sont toutes les deux correctes selon (s_i, s_{i+1}) alors au temps (t_0+t+1) la cellule c est correcte selon (s_i, s_{i+1}).

Lemme 2.3.2
Si au temps (t_0+t) la cellule $(c+s_i)$ est correcte selon s_i et que les cellules $(c+s_{i-1})$ et $(c+s_{i+1})$ sont respectivement correctes selon (s_{i-1}, s_i) et (s_i, s_{i+1}) alors au temps (t_0+t+1) la cellule c est correcte selon s_i.

Les figures 2.17 et 2.18 illustrent les preuves de ces deux lemmes. Dans chacun des cas on a représenté à gauche le voisinage de la portion que la cellule doit connaître correctement pour être correcte ($(c+T_i)$ dans le premier cas et $(c+(T_{i-1}\cup T_i\cup S_i))$ dans le second cas) et à droite la zone des informations correctes qu'elle peut voir d'après les hypothèses sur la correction de ses voisines : dans le premier cas la partie qui nous intéresse de l'information provient de ce que contiennent les voisines de c selon s_i et s_{i+1} qui sont correctes, par hypothèse, selon (s_i, s_{i+1}). Dans le second cas, il faut séparer ce qui est apporté par la cellule $(c+s_i)$ (représenté en noir) et ce qui est apporté par les deux autres voisines $(c+s_{i-1})$ et $(c+s_{i+1})$ (représenté en gris).

On voit que dans les deux cas la cellule dispose d'assez d'informations justes au temps (t_0+t) pour être correcte au temps (t_0+t+1) (la pente des côtés du trapèze central T_i a été choisie identique à la pente des côtés de $\mathrm{CH}(V)$ pour que les choses se passent correctement, on utilise aussi le fait que $\mathrm{CH}(V)^{t_0}$ est convexe).

Ici encore, tout comme nous l'avions défini en dimension 1, en cas de désaccord entre les différentes informations portées par une cellule et ses voisines, la cellule donnera la priorité aux informations portées par la cellule $(c+s_i)$ en ce qui concerne les informations se trouvant le plus loin dans la direction s_i.

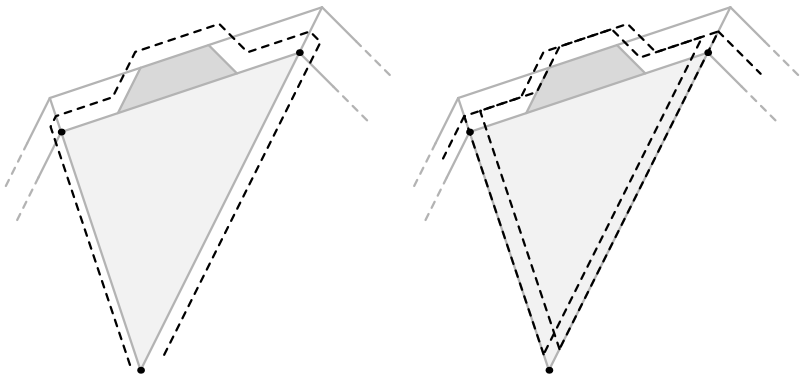

FIGURE 2.17 – Les états que la cellule c doit connaître pour être correcte selon (s_i, s_{i+1}) au temps suivant (à gauche) et les états corrects que portent ses voisines (à droite).

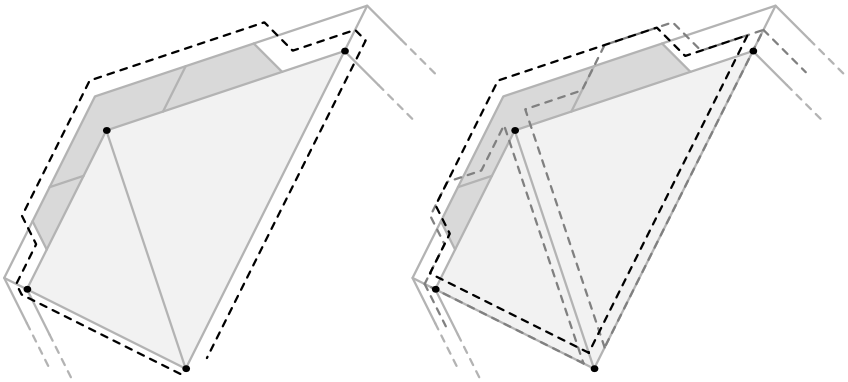

FIGURE 2.18 – Les états que la cellule c doit connaître pour être correcte selon s_i au temps suivant (à gauche) et les états corrects que portent ses voisines (à droite).

Nous savons qu'au temps t_0 toutes les cellules de $\mathcal{A}'$ qui sont « assez proches » des bords du mot w sont correctes selon la direction orientée vers l'extérieur du mot. Les lemmes 2.3.1 et 2.3.2 indiquent ensuite que la correction de ces cellules se propage jusqu'à l'origine selon les vecteurs s_i, jusqu'à ce que finalement à un certain temps $(t_0 + t_f)$ l'origine soit correcte selon toutes les directions possibles. À cet instant, l'origine connaît exactement tous les états de $\{\langle x \rangle_{t_f} \mid x \in \mathrm{CH}(V)^{t_0}\}$ et peut donc anticiper la simulation de $\mathcal{A}$ de t_0 pas. Au temps $(t_0 + t_f)$ l'origine est donc capable de connaître l'état $\langle 0 \rangle_{t_0+t_f}$ dans lequel se trouverait l'origine de l'automate $\mathcal{A}$ au temps $(t_0 + t_f)$.

L'automate $\mathcal{A}'$ est donc capable de rattraper le retard de t_0 générations qu'il a pris au début de la simulation. Il reste à montrer que $(t_0 + t_f)$ est exactement le temps réel correspondant au mot w pour montrer que le retard n'est pas compensé trop tard.

2.3.3 Calcul du temps réel

Étant donné un mot $w \in \Sigma^{**}$ on note M l'ensemble des cellules sur lesquelles le mot s'étend lorsqu'il est « écrit » sur la configuration initiale de $\mathcal{A}$ et $\mathcal{A}'$. En d'autres termes, si le mot w est de taille (n, m), on a

$$M = [\![0, n-1]\!] \times [\![0, m-1]\!]$$

Par définition du temps réel, on a alors

$$\mathrm{TR}_V(n, m) = \min\{t \in \mathbb{N} \mid M \subseteq V^t\}$$

Pour alléger les notations, dans cette sous-section nous noterons

$$t_r = \mathrm{TR}_V(n, m)$$

Nous voulons montrer qu'en temps t_r l'origine de $\mathcal{A}'$ est correcte selon toutes les directions possibles. Il faut alors séparer la correction dans les angles (selon s_i) et dans les cônes (selon (s_i, s_{i+1})).

Correction selon les angles

On a le résultat suivant :

Lemme 2.3.3
Si $t_r \geq t_0$ alors pour tout sommet s_i de V la cellule $(t_r - t_0)s_i$ est correcte selon s_i au temps t_0.

Preuve : D'après la définition du temps réel

$$M \subseteq V^{t_r}$$

Si $t_r \geq t_0$, le voisinage V^{t_r} est de la forme « stabilisée » décrite par le théorème 2.3.2. De plus, puisque l'on a choisi t_0 assez grand pour que V^{t_0} soit également stabilisé, on sait que la zone correspondant au sommet s_i dans les deux voisinages V^{t_0} et V^{t_r} est identique (voir figure 2.19).

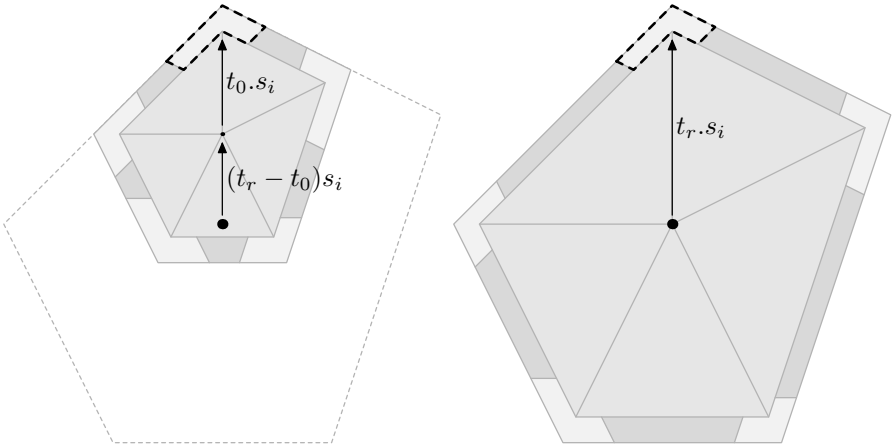

FIGURE 2.19 – Les ensembles $V^{t_0} + (t_r - t_0)s_i$ (à gauche) et V^{t_r} (à droite) coïncident sur la partie indiquée en pointillés noirs.

Ainsi, puisque M est inclus dans V^{t_r} il n'y a aucun point de M dans la surface correspondant à l'angle s_i (la surface indiquée en pointillés noirs sur la figure 2.19) qui ne soit pas dans la partie qui appartient effectivement au voisinage V^{t_r}.

Or par le théorème 2.3.2 on sait que tous les points de cette surface qui appartiennent à V^{t_r} appartiennent également à $V^{t_0} + (t_r - t_0)s_i$ puisque ces deux voisinages partagent la même forme dans le coin correspondant à s_i.

Ainsi lorsque la cellule $(t_r - t_0)s_i$ suppose au temps t_0 que tous les états qu'elle ne voit pas dans cette surface sont B elle a raison, et donc cette cellule est correcte selon s_i au temps t_0. □

Correction selon les cônes

La situation est similaire en ce qui concerne les cellules qui se trouvent à l'intérieur des cônes.

Puisque l'on a vu que la correction selon (s_i, s_{i+1}) d'une cellule se propage de $(c+s_i)$ et $(c+s_{i+1})$ à c et que l'on ne s'intéresse *in fine* qu'à la correction de l'origine, nous ne nous intéresserons qu'aux cellules de la forme $(a.s_i + b.s_{i+1})$ pour $a, b \in \mathbb{N}$.

On a alors le lemme suivant :

Lemme 2.3.4
Si $t_r \geq t_0$ alors pour tout sommet s_i de V toutes les cellules de la forme $(a.s_i + b.s_{i+1})$ avec $a, b \in \mathbb{N}$ et $a+b = t_r - t_0$ sont correctes selon (s_i, s_{i+1}) au temps t_0 (ces cellules sont toutes sur le segment $[(t_r - t_0).s_i, (t_r - t_0).s_{i+1}]$ comme illustré sur la figure 2.20).

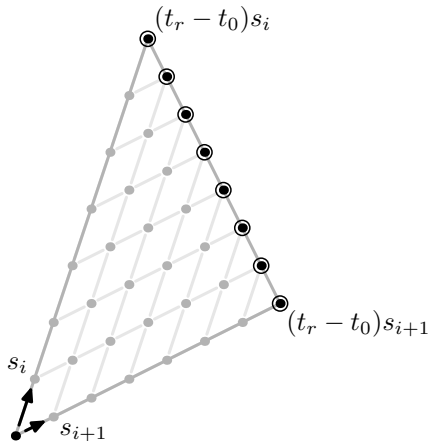

FIGURE 2.20 – Les cellules $(a.s_i + b.s_{i+1})$ avec $a, b \in \mathbb{N}$ et $a + b = t_r - t_0$ (en noir).

Preuve : Comme précédemment, si $t_r \geq t_0$ le voisinage V^{t_r} est de la forme décrite par le théorème 2.3.2, tout comme le voisinage V^{t_0}. Cela signifie que les trapèzes centraux de la bande d'épaisseur t_c sur ces deux voisinages sont tous les deux des superpositions des mêmes trapèzes élémentaires périodiquement translatés selon la direction $(s_{i+1} - s_i)$.

Cela signifie en particulier que pour toute cellule $c = (a.s_i + b.s_{i+1})$ avec $a, b \in \mathbb{N}$ et $a + b = t_r - t_0$ le remplissage de $V^{t_0}(c)$ sur la zone trapézoïdale selon (s_i, s_{i+1}) coïncide avec le voisinage V^{t_r} (voir figure 2.21)

Ainsi, puisque M est inclus dans V^{t_r} toutes les lettres du mot qui se trouvent dans la surface que la cellule c doit connaître pour être correcte selon (s_i, s_{i+1}) sont sur des points que la cellule peut voir. Elle ne fait donc que des hypothèses justes lorsqu'elle suppose que tous les états qu'elle ne voit pas sont B. Toutes les cellules de la forme $(a.s_i + b.s_{i+1})$ pour $a, b \in \mathbb{N}$ et $a + b = t_r - t_0$ sont donc correctes selon (s_i, s_{i+1}) au temps t_0. □

2.3.4 Fin de la preuve

Les lemmes 2.3.1, 2.3.2, 2.3.3 et 2.3.4 nous permettent alors de montrer par récurrence les deux lemmes suivants :

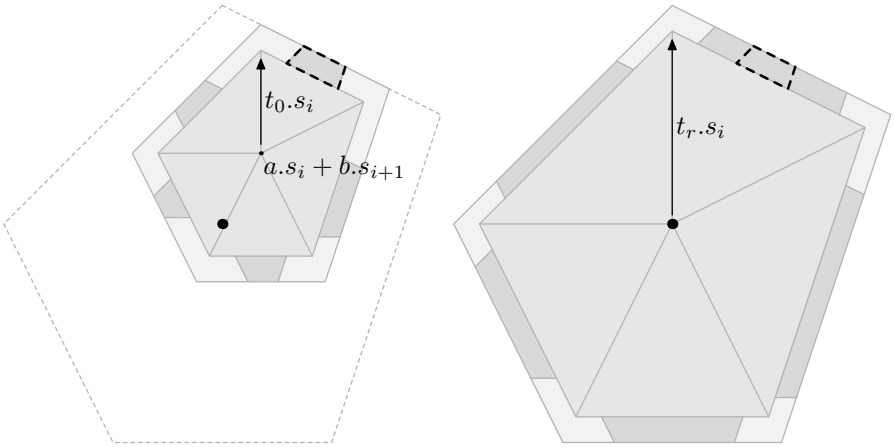

FIGURE 2.21 – Les ensembles $V^{t_0}(c)$ (à gauche) et V^{t_r} (à droite) coïncident sur la partie indiquée en trait noir pointillé.

Lemme 2.3.5
Si $t_r \geq t_0$, pour tout sommet s_i de V et tout $t \in \mathbb{N}$, la cellule $(t_r - t_0 - t)s_i$ est correcte selon s_i au temps $(t_0 + t)$.

Lemme 2.3.6
Si $t_r \geq t_0$, pour tout sommet s_i de V et tout $t \in \mathbb{N}$, toutes les cellules de la forme $(a.s_i + b.s_{i+1})$ avec $a, b \in \mathbb{N}$ et $a + b = t_r - t_0 - t$ sont correctes selon (s_i, s_{i+1}) au temps $(t_0 + t)$.

On conclut alors en observant que si $t_r \geq t_0$, l'origine est correcte selon toutes les directions au temps $t_0 + (t_r - t_0) = t_r$. Ainsi, sur une entrée w de taille (n, m), l'automate $\mathcal{A}'$ fonctionnant sur le voisinage V est capable à tout temps $t \geq \mathrm{TR}_V(n, m)$ de calculer l'état dans lequel se trouverait l'origine de l'automate $\mathcal{A}$ sur l'entrée w au temps t.

Puisqu'il n'existe qu'un nombre fini de mots $w \in \Sigma^{**}$ tels que le temps réel associé à ces mots soit inférieur à t_0 on peut modifier notre automate pour qu'il traite individuellement chacun de ces cas en temps réel, ce qui achève la preuve de la proposition 2.3.1.

2.4 Conclusion

On sait que les classes de langages $L_{\mathrm{TR}}(V_{\mathrm{std}})$ et $L_{\mathrm{TR}}(\{0, 1\})$ sont distinctes (avec une inclusion triviale de la seconde dans la première). Par exemple le langage des nombres premiers écrits en unaire est reconnu en temps réel sur le voisinage standard [8] mais pas sur le voisinage one-way [4]. Le théorème 2.2.3 nous indique donc qu'il existe exactement deux classes de voisinages semi-complets en termes de reconnaissance en temps réel.

En ce qui concerne les voisinages qui ne sont pas semi-complets, nous savons qu'ils sont strictement moins puissants que les voisinages complets en termes de reconnaissance en temps réel puisqu'il existe des cellules qui ne peuvent pas transmettre d'information à l'origine. Toute lettre placée sur une de ces cellules

ne peut donc avoir aucune influence sur la reconnaissance ou non du mot en entrée et des langages aussi simples que $\{w = w_0 w_1 \dots w_{l-1} \mid w_c = a\}$ pour un certain c ne peuvent donc pas être reconnus sur un voisinage tel que $c \notin V^\infty$.

Certains voisinages non semi-complets peuvent cependant reconnaître en temps réel des langages qu'il est impossible de reconnaître sur le voisinage one-way. En effet, bien que le voisinage $2V_{\text{std}} = \{-2, 0, 2\}$ ne soit pas semi-complet, pour tout langage L reconnu en temps réel sur V_{std}, le langage

$$L_d = \{w = w_0 w_1 \dots w_l \mid w_0 w_2 w_4 \dots w_{\lfloor l/2 \rfloor} \in L\}$$

est reconnu en temps réel sur $2V_{\text{std}}$. Si l'on considère un langage L qui n'est pas reconnu en temps réel sur le voisinage one-way, L_d ne l'est pas non plus.

Les voisinages qui ne sont pas semi-complets présentent donc encore une certaine complexité. Nous ne savons pas si la classe des langages que l'on peut reconnaître en temps réel sur un voisinage dépend uniquement des cellules accessibles ou non (est-ce que $V^\infty = V'^\infty$ implique $L_{\text{TR}}(V) = L_{\text{TR}}(V')$?). Mais leur intérêt en termes de reconnaissance de langages est limité.

En dimension quelconque, on sait qu'il existe différentes classes de voisinages en termes de capacité de reconnaissance en temps réel. Nous avons montré qu'un voisinage complet V était « au moins aussi puissant » que son enveloppe convexe (ce qui peut paraître contre-intuitif, mais il faut bien remarquer qu'un voisinage « petit » dispose de plus de temps qu'un voisinage « grand » en temps réel). Dans le cas général, nous ne savons pas si l'enveloppe convexe est réciproquement aussi puissante que le voisinage V (la réciproque est vraie si l'on dispose d'un théorème d'accélération constante sur $\text{CH}(V)$).

On conjecture par ailleurs que si deux voisinages complets V et V' ont des enveloppes convexes non semblables (c'est-à-dire que pour tous k et k', $\text{CH}(V)^k \neq \text{CH}(V')^{k'}$) alors les classes $L_{\text{TR}}(V)$ et $L_{\text{TR}}(V')$ sont incomparables au sens de l'inclusion, mais la question reste ouverte.

Chapitre 3

Accélération par une constante

> And the life of the ebony clock went out with that of the last of the gay. And the flames of the tripods expired. And Darkness and Despair and the Red Death held illimitable dominion over all.
>
> **Edgar Allan Poe** – *The Masque of the Red Death*

3.1 Le voisinage de Moore

En dimension 2 si l'on considère des automates cellulaires fonctionnant sur le voisinage de Moore il est assez facile de généraliser le résultat que l'on avait obtenu en dimension 1 sur le voisinage standard.

On a ainsi le théorème suivant :

Théorème 3.1.1
En dimension 2 et au-delà, tout langage reconnu en temps $(\mathrm{TR}_{V_{\mathrm{M}}} + k)$ *par un automate cellulaire fonctionnant sur le voisinage de Moore est également reconnu en temps réel par un automate cellulaire fonctionnant sur le voisinage de Moore.*

La preuve que nous présentons ici ne concerne que la dimension 2 afin d'en simplifier les notations. Elle est toutefois immédiatement généralisable à toute dimension (y compris la dimension 1 où l'on retrouve alors exactement la preuve du théorème 2.1.1).

Tout comme pour prouver le théorème 2.1.1, considérons un automate $\mathcal{A}$ capable de reconnaître un langage L en temps $(\mathrm{TR}_{V_{\mathrm{M}}} + k)$. On décrit alors le fonctionnement d'un automate $\mathcal{A}'$ correspondant à une entrée $w \in \Sigma^{**}$ de taille (n, m). On note $\langle c \rangle_t$ l'état dans lequel se trouve la cellule c au temps t dans l'évolution de $\mathcal{A}$ partant de la configuration initiale correspondant à w. On peut supposer comme nous l'avons fait précédemment que seules les cellules d'abscisse et d'ordonnée positives de $\mathcal{A}$ participent au calcul.

Ici encore l'automate $\mathcal{A}'$ va simuler sans perdre de temps le fonctionnement de $\mathcal{A}$ tout en essayant de connaître pour chaque cellule les états qui se trouvent

dans son voisinage ${V_{\mathrm{M}}}^k$ afin que l'origine puisse au temps $\mathrm{TR}_{V_{\mathrm{M}}}$ anticiper la reconnaissance du mot de k étapes.

Pour réaliser cette anticipation, chaque cellule de $\mathcal{A}'$ contient en réalité $(k+1)^2$ états de $\mathcal{A}$. Chaque cellule c essaie ainsi de « deviner » les états dans lesquels se trouvent ses voisines $\{c + (x, y) \mid x, y \in [\![0, k]\!]\}$.

Au temps initial, la cellule c ne reçoit comme information que l'état $\langle c \rangle_0$ qui est soit une lettre de w si $c \in [\![0, n-1]\!] \times [\![0, m-1]\!]$ soit l'état B. Toutes les cellules supposent alors que les états qu'elles cherchent à deviner (les états $\{\langle c + (x, y)\rangle_0 \mid x, y \in [\![0, k]\!]\}$ pour la cellule c) sont B. Pour certaines cellules ces hypothèses sont toutes justes (par exemple les cellules loin du mot), pour certaines cellules seules quelques hypothèses sont justes tandis que d'autres sont fausses (les cellules proches du bord du mot par exemple) et pour d'autres cellules toutes les hypothèses sont fausses (les cellules qui sont au cœur du mot).

On dira d'une cellule c qu'elle est *correcte* si toutes les hypothèses qu'elle fait à un temps t sont correctes, c'est-à-dire si elle connaît parfaitement tous les états $\{\langle c + (x, y)\rangle_t \mid x, y \in [\![0, k]\!]\}$. On dira qu'elle est *h-correcte* si elle connaît correctement les états des cellules se trouvant à sa droite, c'est-à-dire les états $\{\langle c + (x, 0)\rangle_t \mid x \in [\![0, k]\!]\}$ et enfin qu'elle est *v-correcte* si elle connaît les états correspondant aux cellules au-dessus d'elle : $\{\langle c + (0, y)\rangle_t \mid y \in [\![0, k]\!]\}$. Remarquons que toute cellule correcte est également h-correcte et v-correcte.

Les cellules de $\mathcal{A}'$ évoluent alors en appliquant la règle de transition de $\mathcal{A}$ à toutes les informations qu'elles portent. Lorsque les informations qu'une même cellule peut obtenir dans son voisinage sont incompatibles la priorité est donnée aux informations portées par la cellule la plus en haut à droite (celle dont les coordonnées sont les plus grandes). Il est immédiat par induction que pour tout temps t toute cellule c connaît correctement l'état $\langle c \rangle_t$.

On a alors les résultats suivants :

Lemme 3.1.1
Au temps 0, la cellule $(m-1, n-1)$ est correcte et pour tout $x \in \mathbb{Z}$ la cellule $(x, n-1)$ est v-correcte et la cellule $(m-1, x)$ est h-correcte. Toutes les cellules au-delà des lettres du mot w, c'est-à-dire les cellules $\{(x, y) \mid x \geq n$ ou $y \geq m\}$ sont correctes.

Preuve : Ce résultat provient du fait que les cellules décrites sont sur les bords du mot (ou bien au-delà) et que les cellules correspondant aux positions qu'elles cherchent à deviner sont bien dans l'état B au temps 0. □

Lemme 3.1.2
Si au temps t toutes les cellules sur la ligne $\{(x, k+1) \mid x \in \mathbb{Z}\}$ (pour un certain $k \in \mathbb{N}$) sont v-correctes alors au temps $(t+1)$ toutes les cellules sur la ligne $\{(x, k) \mid x \in \mathbb{Z}\}$ sont v-correctes. On a le résultat analogue concernant les colonnes.

Preuve : La preuve de ce lemme est illustrée par la figure 3.1. On a représenté en gris foncé les positions correspondant aux états que la cellule c doit connaître correctement au temps $(t+1)$ pour être v-correcte (les k positions se trouvant au-dessus d'elle). Les positions en gris clair correspondent au voisinage de ces positions. Enfin, les contours représentés en gras illustrent les

positions des états que les voisines de la cellule c connaissent correctement au temps t : les cellules sur la même ligne que c ou en dessous connaissent l'état se trouvant sur leur propre position, tandis que les trois voisines se trouvant sur la ligne du dessus connaissent par hypothèse tous les états se trouvant sur les k positions au-dessus d'elles au temps t puisqu'elles sont v-correctes. On voit alors que la cellule c dispose de toutes les informations nécessaires pour calculer correctement les états se trouvant au temps $(t+1)$ sur les k positions au-dessus d'elle. Elle sera donc v-correcte au temps $(t+1)$. □

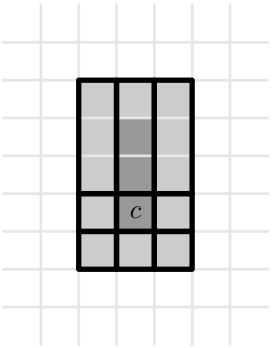

FIGURE 3.1 – Preuve du lemme 3.1.2 (ici $k = 2$).

Lemme 3.1.3
Au temps t toutes les cellules sur le demi-plan $\{(x, y) \mid y \geq m - 1 - t\}$ sont v-correctes et toutes les cellules du demi-plan $\{(x, y) \mid x \geq n - 1 - t\}$ sont h-correctes.

Preuve : Par induction. Le résultat au temps 0 est une conséquence du lemme 3.1.1. L'induction s'obtient en appliquant le lemme 3.1.2. □

Lemme 3.1.4
Si au temps t la cellule $(c + (1, 1))$ est correcte, toutes les cellules sur la ligne au dessus de c sont v-correctes et toutes les cellules sur la colonne à droite de c sont h-correctes alors au temps $(t + 1)$ la cellule c est correcte.

Preuve : Le raisonnement est illustré par la figure 3.2. On a ici représenté en gris foncé les positions correspondant aux états que la cellule c doit connaître correctement au temps $(t + 1)$ pour être correcte et en gris clair le voisinage de ces positions. Les contours en gras représentent ici encore les positions correspondant aux états connus par les cellules voisines de c. On voit alors que la cellule c dispose d'assez d'information pour être correcte au temps suivant. □

On peut maintenant conclure la preuve du théorème 3.1.1. Supposons $n \leq m$ (le mot w est plus haut que large). D'après le lemme 3.1.2 on sait que les cellules de la ligne $\{(x, n) \mid x \in \mathbb{Z}\}$ sont v-correctes au temps $(m - n - 1)$, tandis que

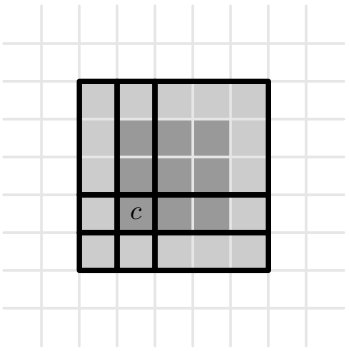

FIGURE 3.2 – Preuve du lemme 3.1.4 (ici $k = 2$)[1]

le lemme 3.1.1 indique que les cellules sur la colonne $\{(n, x) \mid x \in \mathbb{Z}\}$ sont h-correctes depuis le temps 0 et que la cellule (n, n) est correcte depuis le temps 0 également.

Au temps $(m - n - 1)$ la cellule $(n - 1, n - 1)$ est donc dans le cas décrit par les hypothèses du lemme 3.1.4 et elle sera correcte au temps $(m - n)$. À partir de cet instant, le lemme 3.1.4 peut être appliqué successivement à toutes les cellules de la diagonale jusqu'à l'origine : la cellule $(n - x, n - x)$ pour $x \geq 0$ est donc correcte au temps $(m - n - 1 + x)$ ce qui signifie que l'origine est correcte au temps

$$m - n - 1 + n = m - 1 = \mathrm{TR}_{V_{\mathrm{M}}}(n, m)$$

On a le même résultat dans le cas où $n \geq m$. On a donc montré qu'au temps $\mathrm{TR}_{V_{\mathrm{M}}}(n, m)$ l'origine de $\mathcal{A}'$ a toutes les informations nécessaires pour anticiper la simulation de $\mathcal{A}$ de k étapes et donc pour conclure sur l'appartenance de w à L. L'automate $\mathcal{A}'$ reconnaît ainsi le langage L en temps réel.

3.2 Le voisinage de Von Neumann

Dans le cas du voisinage de Von Neumann les choses sont très différentes. En effet, la méthode exposée précédemment ne permet pas de conclure parce que la correction des hypothèses d'une cellule ne se propage pas aussi bien que dans le cas du voisinage de Moore. On peut ainsi montrer que quelle que soit la forme selon laquelle les cellules essaient de deviner l'état de leurs voisines, il manque toujours des informations pour que la correction se propage jusqu'à l'origine en temps réel.

Toutefois, en exploitant le fait que dans un mot donné il n'existe qu'une unique lettre qui se trouve à distance maximale de l'origine au sens du voisinage de Von Neumann (comme il a été vu dans la sous-section 1.6.1) on peut également obtenir un théorème d'accélération constante sur le voisinage de Von Neumann : on construit pour cela un automate cellulaire qui va anticiper d'une étape la reconnaissance du mot qu'il reçoit en entrée en effectuant en parallèle les $|\Sigma|$ calculs correspondant aux différentes valeurs possibles de la lettre la

[1] On appréciera la manifestation d'une illusion d'optique bien connue qui donne l'impression que les lignes du quadrillage sont plus claires dans la zone en gris que dans la zone extérieure alors qu'elles sont en réalité de la même couleur.

plus éloignée de l'origine puis déterminer, au temps réel lorsque la lettre atteint l'origine, lequel de ces calculs était correct.

Nous n'entrerons cependant pas dans les détails de cette construction ici, pour nous intéresser plutôt à une technique inspirée des théorèmes d'accélération linéaire qui va nous permettre d'obtenir une accélération constante « partielle » sur le voisinage de Von Neumann. Bien qu'apportant un résultat plus faible que ce que l'on peut obtenir dans le cas particulier du voisinage de Von Neumann, cette méthode pourra être facilement généralisée à une vaste classe de voisinages (ceux pour lesquels il est possible d'obtenir un théorème d'accélération linéaire). Nous allons donc l'illustrer ici en détails pour étudier par la suite les applications possibles à d'autres voisinages dans le chapitre 4.

3.2.1 Un théorème d'accélération partielle

On a le théorème suivant :

Théorème 3.2.1
Soit $L \subseteq \Sigma^{**}$ un langage d'images sur l'alphabet Σ et k un entier naturel strictement positif.

Si L est reconnu par un automate cellulaire fonctionnant sur le voisinage de Von Neumann V_{VN} en temps

$$\text{TR}_{V_{\text{VN}}} + k : \left\{ \begin{array}{ccc} \mathbb{N}^2 & \to & \mathbb{N} \\ (n,m) & \mapsto & (n+m-2)+k \end{array} \right.$$

il est reconnu par un automate cellulaire fonctionnant sur le voisinage de Von Neumann en temps $(\text{TR}_{V_{\text{VN}}} + 1)$.

Ce théorème est moins satisfaisant que celui que l'on avait déjà obtenu en dimension 1 sur tout voisinage ou en dimensions supérieures sur le voisinage de Moore car bien qu'il permette une accélération constante pour tout langage reconnu en temps $(\text{TR}_{V_{\text{VN}}} + k)$ pour presque tout k, il ne permet pas de conclure dans le cas où $k = 1$. Nous dirons alors que nous avons obtenu une accélération constante partielle.

Nous verrons dans le chapitre 4 que pour tous les voisinages sur lesquels il est possible d'obtenir un théorème d'accélération linéaire nous pouvons obtenir un résultat d'accélération constante partielle. Dans le cas du voisinage de Von Neumann il est nécessaire de regarder précisément la construction pour assurer que l'on peut atteindre une accélération jusqu'à un temps de reconnaissance $(\text{TR}_{V_{\text{VN}}} + 1)$ (le théorème d'accélération usuel donne facilement une accélération jusqu'à $(\text{TR}_{V_{\text{VN}}} + 3)$).

Toute la suite de cette section sera consacrée à la preuve du théorème 3.2.1. Pour cela on considère un langage $L \subseteq \Sigma^{**}$ reconnu en temps $(\text{TR}_{V_{\text{VN}}} + k + 2)$ par un automate cellulaire $\mathcal{A}$ d'états $\mathcal{Q}$ fonctionnant sur le voisinage V_{VN} et l'on va construire un automate $\mathcal{A}'$ qui reconnaît L en temps $(\text{TR}_{V_{\text{VN}}} + k + 1)$ (pour n'importe quel $k \in \mathbb{N}$).

Le voisinage de Von Neumann étant symétrique selon chacun des deux axes, on peut utiliser la technique consistant à « replier » l'espace horizontalement et verticalement de telle sorte que le calcul n'ait lieu que dans le quart de plan supérieur droit (toutes les autres cellules restent toujours dans l'état B).

L'automate $\mathcal{A}'$ va fonctionner en deux phases. Pendant la première phase, les lettres du mot donné sur la configuration initiale vont être groupées quatre par quatre sur chaque cellule de manière à obtenir une copie de la configuration initiale de dimensions moitié. On dira que l'on a compressé l'entrée (voir figure 3.3).

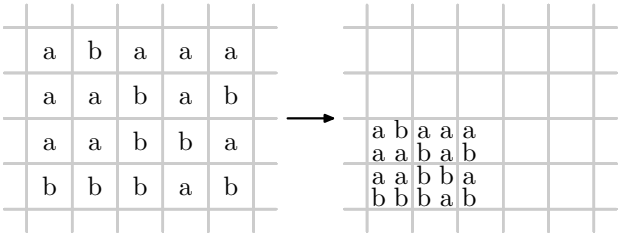

FIGURE 3.3 – Exemple de compression d'une configuration initiale par un facteur 2.

Par la suite, pour une plus grande clarté et afin d'illustrer plus généralement le fonctionnement de l'automate on ne représentera dans une configuration que les positions où se trouvent les états de l'automate initial $\mathcal{A}$ (représentés par des cercles). Ainsi la version simplifiée de la figure 3.3 est la figure 3.4.

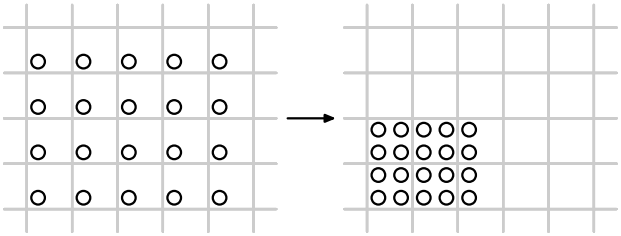

FIGURE 3.4 – Représentation simplifiée de la compression d'une configuration initiale.

Sur cette figure chaque cercle représente un unique état de $\mathcal{Q}$, mais l'on voit bien que les cellules de $\mathcal{A}'$ peuvent contenir jusqu'à 4 états de $\mathcal{Q}$ (nous verrons par la suite que pour des raisons techniques une cellule contiendra en réalité plus d'états mais nous n'en représenterons que 4).

3.2.2 La compression de l'entrée

L'idée générale ce cette compression est de déplacer les informations (lettres du mot en entrée) le plus possible vers la gauche puis de les déplacer vers le bas. Les règles représentées sur la figure 3.5 illustrent ces idées (les flèches représentent les déplacements des états de $\mathcal{Q}$).

Ainsi, la règle (a) concerne une cellule qui contient une unique lettre α, telle que sa voisine de gauche ne contienne également qu'une seule lettre (donc peut recevoir l'information α puisqu'il y a de la place où la mettre) et telle que sa voisine de droite contienne également une unique lettre β. Au temps suivant la cellule oublie la lettre α qu'elle connaissait (puisqu'elle sait que cette information se retrouvera sur sa voisine de gauche et ne sera donc pas perdue) et

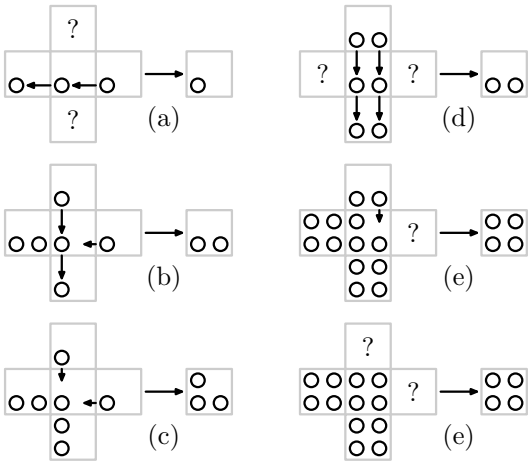

FIGURE 3.5 – Les règles de la phase de compression.

prend l'information β que sa voisine de droite contenait (les lettres se déplacent donc toutes vers la gauche comme indiqué par les flèches). Notons que si la cellule peut transmettre son information vers la gauche elle n'interagit pas avec ses voisines verticales.

La règle (b) par contre représente le cas où la cellule voit que sa voisine de gauche n'a plus la place de recevoir la lettre qu'elle porte (puisque sa voisine contient déjà 2 lettres horizontalement). Par contre sa voisine du bas a encore de la place, et donc la cellule va non seulement envoyer sa lettre à sa voisine du bas mais également prendre la lettre de sa voisine du haut : on est dans le cas où les informations ne peuvent plus circuler horizontalement donc on passe à un fonctionnement vertical. Par ailleurs, la voisine de droite contient une lettre qu'elle va vouloir transmettre vers la gauche (puisque cette dernière n'a pas les moyens de savoir que la circulation est bloquée vers la gauche) donc la cellule que nous observons doit prendre cette information en plus de la lettre portée par sa voisine de dessus.

La figure (c) représente le cas où l'information ne peut plus circuler ni vers la gauche ni vers le bas, la cellule garde alors son information et prend en plus celles de ses voisines de dessus et de droite. Enfin les deux dernières règles indiquent le fonctionnement lorsqu'une cellule a déjà deux lettres horizontalement et que les informations circulent donc verticalement.

Par ailleurs, les cellules dans l'état B sont considérées comme « totalement groupées » c'est-à-dire que leurs voisines savent qu'elles ne peuvent pas transmettre l'information dans leur direction (par exemple, au temps 0 l'origine applique la règle (c) puisqu'elle est dans un cas où elle sait que la lettre qu'elle porte ne sera acceptée ni par sa voisine de gauche ni par sa voisine du bas).

La figure 3.6 illustre le processus de compression dans le cas d'une image en entrée de taille $(6, 4)$.

Sur la figure, on a représenté par une croix les emplacements où une cellule reçoit un état B. Ces emplacements doivent être occupés lors de la construction pour indiquer aux cellules voisines le stade d'avancement de la compression.

On remarque que les règles sont telles que l'alignement vertical des lettres

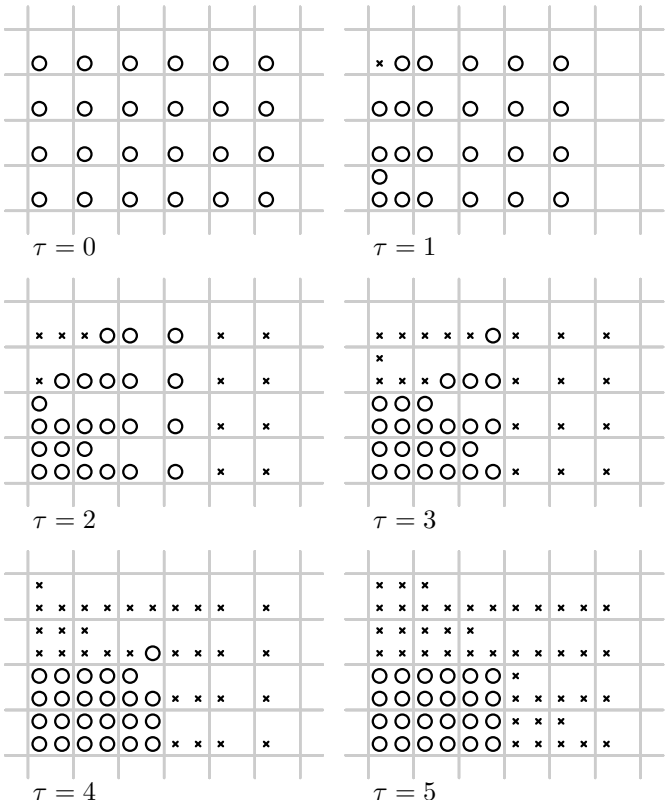

FIGURE 3.6 – La compression d'une image de taille $(6, 4)$.

du mot en entrée est conservé au cours de la construction (il y a toujours 4 cercles sur chaque colonne) mais que les lignes ne le sont pas puisque les lettres les plus à gauche descendent avant les autres.

Une observation qui nous sera utile par la suite lorsque l'on s'intéressera au temps précis que prend notre construction est que toute lettre du mot en entrée, tant qu'elle n'a pas atteint la cellule qu'elle doit atteindre après compression, se déplace d'exactement une cellule vers cette cellule cible à chaque génération (horizontalement d'abord, puis verticalement) nous assurant ainsi que chaque lettre atteint son objectif « le plus rapidement possible ».

On montre ainsi que l'origine est « complètement groupée » au temps 2 (il lui faut recevoir la lettre qui se trouve en haut à droite), puis ses voisines au temps 3 et de manière générale une cellule à distance d de l'origine recevra l'information groupée qu'elle attend au temps exactement $(2+d)$. Enfin, la dernière lettre du mot atteint sa cellule cible au temps $\lfloor n/2 \rfloor + \lfloor m/2 \rfloor$.

3.2.3 La simulation accélérée de $\mathcal{A}$

Nous avons vu que la compression dure environ la moitié du temps réel, et pour l'instant nous n'avons pas appliqué la règle de transition de l'automate $\mathcal{A}$. Pour compenser le temps utilisé à déplacer les lettres, il est nécessaire d'effectuer la simulation du fonctionnement de $\mathcal{A}$ deux fois plus rapidement, c'est-à-dire qu'il va falloir calculer deux générations de $\mathcal{A}$ par génération de $\mathcal{A}'$.

Lorsque la compression a été correctement effectuée dans une zone autour d'une cellule, on est dans la situation illustrée sur la partie gauche de la figure 3.7. Sur cette figure les cellules de $\mathcal{A}'$ sont représentées par des carrés gris, tandis que les états de $\mathcal{A}$ sont représentés par des points noirs (on est donc bien dans le cas où chaque cellule de $\mathcal{A}'$ contient quatre états de $\mathcal{Q}$). Par ailleurs, on a marqué par un contour noir les états que la cellule centrale voit (ici, ce sont simplement les états portés par les voisines de la cellule).

Pour simuler deux générations de $\mathcal{A}$ en un seul temps, la cellule centrale doit être capable de voir les états qui, dans le fonctionnement normal de $\mathcal{A}$ se trouveraient dans le voisinage double des états qu'elle porte (pour pouvoir connaître la valeur de ces quatre états deux temps plus tard). Sur la figure, on a encerclé les états se trouvant dans le voisinage double de l'état de la cellule centrale se trouvant en haut à gauche. On voit que l'un de ces états n'est pas dans ce que voit la cellule et il n'est donc pas possible de simuler $\mathcal{A}$ avec un facteur d'accélération de 2 dans cette situation.

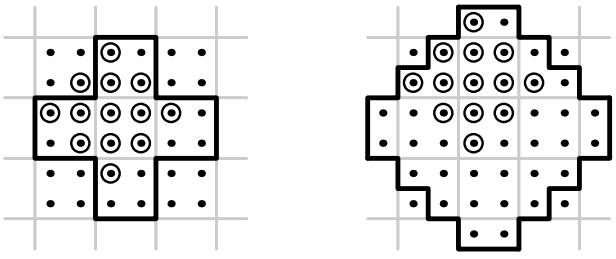

FIGURE 3.7 – Ce qu'une cellule voit, et ce qu'elle devrait voir dans le cas simple (à gauche) et le cas avec un unique temps de stockage (à droite).

Pour résoudre ce problème, on modifie légèrement le fonctionnement de l'automate $\mathcal{A}'$. Au lieu de commencer immédiatement la compression que nous avons décrite précédemment, les $(k+1)$ premières étapes de l'automate sont utilisées pour « stocker » des informations c'est-à-dire qu'à chacune de ces étapes chaque cellule de la configuration initiale va regarder l'état de ses voisines et s'en souvenir, augmentant ainsi à chaque étape le nombre d'états qu'elle connaît.

A partir de ce moment, la compression se déroule comme précédemment, à ceci près que chacun des cercles que l'on avait représentés n'est maintenant plus un état de $\mathcal{Q}$ mais plusieurs, puisque chaque cercle représente maintenant tous les états qui se trouvent dans le voisinage à la puissance $k+1$ de la position qu'il occupe (le cercle en position c représente tous les états initialement portés par les cellules de $V^{k+1}(c)$). Les informations sont maintenant très redondantes et c'est pourquoi on conserve la représentation par des cercles. Une représentation plus juste consisterait à représenter les cercles par des « losanges » qui se chevauchent pour indiquer qu'en réalité chaque losange contient de nombreux états, mais la figure deviendrait illisible. Il faut donc à partir de maintenant considérer que si une cellule contient un certain cercle, elle contient également tous les états sur des positions voisines à l'ordre $(k+1)$ de celle où se trouve le cercle.

Par ailleurs, il faut bien garder en mémoire que l'on a « perdu » $(k+1)$ générations initialement pour effectuer ce stockage, mais ceci est compensé par le fait que puisqu'au lieu d'un état de $\mathcal{A}$ on a maintenant en tout point le voisinage à l'ordre $(k+1)$ d'un état, cela signifie en quelque sorte que l'automate a localement la capacité d'anticiper la simulation de $(k+1)$ temps (nous reviendrons sur ce point à la fin de la construction).

Enfin, notons que ces temps de stockage initiaux permettent de faire en sorte que la lettre la plus en haut à droite du mot en entrée (celle qui se trouve le plus loin de l'origine) sache à partir de maintenant qu'elle est la plus éloignée (puisqu'elle voit que ses deux voisines vers le haut et la droite sont dans l'état B). Ceci nous sera utile par la suite.

La compression de la configuration que l'on a décrite précédemment ne pose pas plus de difficultés puisqu'il ne s'agit que de déplacer des informations sans avoir à appliquer la règle de transition de $\mathcal{A}$. On peut donc se ramener à l'image compressée de la même manière mais cette fois-ci la répartition de l'information a changé. La partie droite de la figure 3.7 illustre la nouvelle situation lorsque l'on veut effectuer la simulation 2 fois plus vite dans le cas où $k = 0$:

Les points noirs représentent toujours les états de $\mathcal{A}$ (chaque point ne représente que l'état qui correspond à sa position). Le contour noir montre alors le nouvel ensemble d'états que la cellule centrale voit. Cet ensemble est plus grand que le précédent (partie gauche de la figure) puisque maintenant en plus de tous les états que la cellule pouvait voir avant en regardant les informations portées par ses voisines, on peut voir tous les états qui sont sur des positions voisines (eux aussi se trouvent maintenant sur les cellules voisines de la cellule centrale).

Cependant, on ne peut plus se contenter de simuler le fonctionnement de $\mathcal{A}$ sur les quatre états portés par la cellule puisqu'alors cela signifierait que l'on perd les informations supplémentaires que l'on a mis une génération à obtenir, ce qui nous ramènerait au cas précédent. Il faut donc vérifier que la cellule centrale a non seulement assez d'information pour calculer deux générations de $\mathcal{A}$ sur les états qui correspondent aux 4 positions au centre de la cellule, mais également assez pour faire évoluer de deux générations les états ajoutés, c'est-à-dire ceux sur les positions immédiatement voisines de la cellule centrale. Les

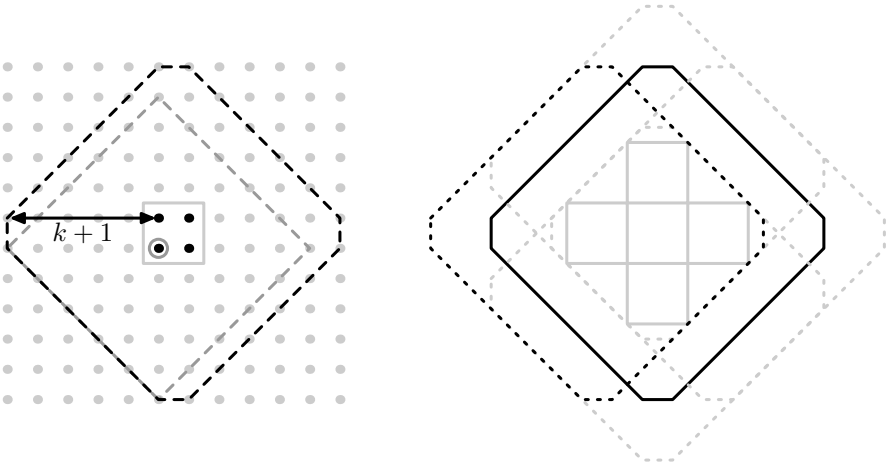

FIGURE 3.8 – Le cas général où l'on effectue $(k+1)$ étapes de stockage initialement.

cercles marqués sur la partie droite de la figure 3.7 représentent les informations nécessaires pour simuler deux générations de $\mathcal{A}$ sur la position au dessus de la cellule centrale, à gauche (le centre de la zone de cercles). On voit que toutes ces informations sont accessibles à la cellule centrale. On vérifie facilement que les informations nécessaires pour calculer l'évolution d'un des quatre états à l'intérieur de la cellule (celles qui étaient marqués par des cercles dans la partie gauche de la figure) sont aussi accessibles, et donc par symétrie la cellule centrale représentée sur la figure peut bien calculer deux générations de $\mathcal{A}$ sur toutes les informations qu'elle porte.

La figure 3.8 illustre ce même raisonnement pour un k quelconque. Sur la partie gauche on a représenté l'ensemble des informations portées par une cellule après groupage. La zone délimitée en pointillés gris représente le voisinage à l'ordre $(k+1)$ de la position se trouvant en bas à gauche de la cellule centrale (ici $k=4$). La zone en pointillés noirs est l'union des quatre voisinages d'ordre $(k+1)$ correspondant aux quatre positions de la cellule, c'est donc l'ensemble des positions dont la cellule connaît les états lorsqu'elle est correctement groupée. On appellera cette zone la *zone de connaissance* de la cellule.

Sur la partie droite de la figure 3.8 on a représenté une cellule et ses voisines. La zone délimitée par un trait plein noir est la zone de connaissance de la cellule centrale c. Si l'on veut simuler deux étapes de $\mathcal{A}$ tout en conservant les informations stockées par la cellule, il faut que cette cellule centrale soit capable de voir tous les états se trouvant dans le voisinage double de cette zone. La zone représentée en pointillés noirs est la zone de connaissance de la cellule se trouvant à gauche de c, c'est donc un ensemble d'états que c peut voir lorsqu'elle effectue sa transition. On a également représenté en pointillés gris clair les zones de connaissance des trois autres voisines. On voit alors que l'union de ces zones est exactement le voisinage double de la zone de connaissance de c, ce qui permet donc de simuler deux générations de $\mathcal{A}$ sur chacune des positions contenues dans la zone de connaissance de c.

3.2.4 Combinaison des deux étapes

Maintenant que l'on a vu comment se déroulait la compression puis comment $\mathcal{A}'$ pouvait simuler $\mathcal{A}$ deux fois plus vite quand les informations étaient groupées, voyons précisément comment les choses se combinent.

Nous considérerons ici que l'image en entrée est une image w de taille (n, m) où n et m sont tous deux pairs (nous verrons plus tard comment traiter les cas impairs). Pendant les $(k+1)$ premières étapes de la simulation, les informations ne circulent pas vraiment mais chaque cellule mémorise l'information portée par ses voisines à l'ordre $(k + 1)$ comme il a été expliqué auparavant. Ces étapes permettent à la lettre de l'image la plus éloignée de l'origine de se distinguer des autres (on dira donc que cette lettre « sait » qu'elle est la dernière à partir de maintenant).

La phase de compression commence alors et, en accord avec ce que l'on a vu précédemment mais en tenant compte du temps de stockage initial, on sait que l'origine est totalement groupée au temps $(k+3)$ et qu'à chaque temps qui suit, les voisines directes des cellules déjà groupées le deviennent à leur tour. Ainsi, la cellule (i, j) est complètement groupée au temps $(k + 3 + i + j)$.

À l'instant où une cellule achève son groupage, elle contient donc toutes les lettres du mot en entrée correspondant aux quatre positions qu'elle représente selon la compression ainsi que les lettres sur les positions voisines (à l'ordre $(k + 1)$). On dira alors que le *temps simulé* de la cellule est 0. De manière générale, on dira que le *temps simulé* d'une cellule est t si elle connaît les états correspondant aux positions qu'elle représente (y compris les voisines à l'ordre $(k + 1)$ des 4 positions centrales) au temps t dans l'évolution normale de $\mathcal{A}$ sur l'entrée w.

Avec cette dénomination, nous avons vu dans la sous-section précédente que si le temps simulé d'une cellule c de $\mathcal{A}'$ et de ses voisines était t à un certain instant dans l'évolution de $\mathcal{A}'$, alors au temps suivant le temps simulé de c est $(t+2)$. Ainsi par exemple, puisque l'origine est groupée au temps $(k+3)$ et que ses deux voisines sont groupées au temps $(k + 4)$ (on rappelle qu'aucun calcul ne se déroule en dehors du quart de plan supérieur donc seules les voisines en haut et à droite de l'origine sont intéressantes), au temps $(k + 5)$ de $\mathcal{A}'$ le temps simulé de l'origine est 2. Si l'on considère de plus que chaque cellule, au moment de calculer les nouveaux états de $\mathcal{A}$ correspondant aux positions qu'elle représente, se souvient des états précédents qu'elle portait (elle ne se souvient que des états qu'elle portait au temps précédent et non pas tout son historique), on peut considérer qu'une cellule dont le temps simulé est t à un instant donné et tel que toutes ses voisines aient un temps simulé de t ou $t + 2$ soit capable elle aussi de passer à un temps simulé de $t + 2$ à la génération suivante.

Ainsi, si l'on continue notre simulation, au temps $(k + 5)$ de $\mathcal{A}'$ les voisines des voisines de l'origine sont groupées à leur tour (au moment où l'origine passe à un temps simulé de 2), ce qui permet ainsi aux voisines de l'origine de passer à un temps simulé de 2 au temps $(k + 6)$ alors qu'une nouvelle diagonale de cellules atteint le stade groupé. Le début de cette évolution est illustré sur la figure 3.9, où l'on représente par un carré marqué de l'entier n une cellule qui est correctement groupée et dont le temps simulé est n.

On peut alors bien voir sur la figure qu'une cellule n'avance son temps simulé que lorsque les temps simulés de toutes ses voisines sont égaux ou supérieurs au sien. De plus on voit bien que chaque cellule avance son temps simulé de 2

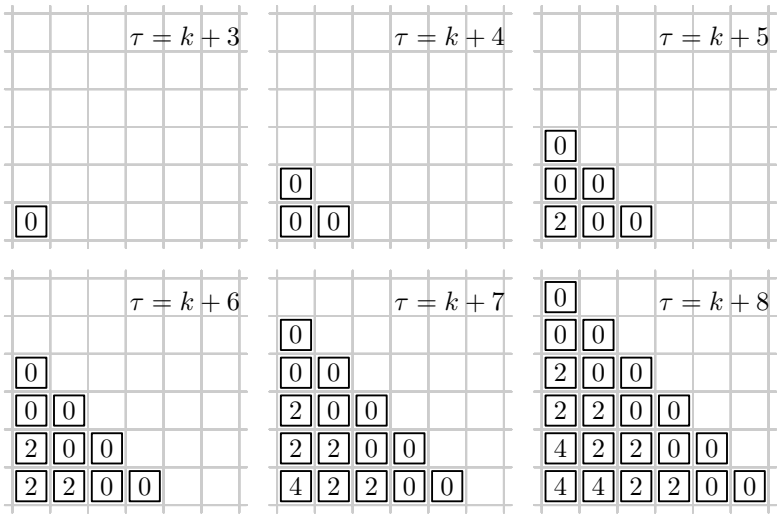

FIGURE 3.9 – Les premières étapes de la simulation.

toutes les deux générations. La cellule (i, j) étant groupée à partir du temps $(k + 3 + i + j)$, elle passe donc au temps simulé t (pour t pair) à la génération (de $\mathcal{A}'$) $(t + k + 3 + i + j)$.

Revenons maintenant à un mot w dont les dimensions (n, m) sont paires :

Au temps $(k + 1)$ la lettre la plus éloignée du mot est toujours sur sa cellule initiale $(n - 1, m - 1)$, et elle « sait » maintenant qu'elle est la dernière lettre du mot w. Puis elle se déplace d'exactement une case vers l'origine (d'abord horizontalement puis verticalement) à chaque étape avant d'être correctement groupée. Elle atteint donc sa cellule cible au temps $(k+1+n/2+m/2)$ puisque sa cible se trouve $n/2$ cellules sur la gauche et $m/2$ cellules plus bas. Pendant ce temps, la simulation de $\mathcal{A}$ a lieu sur les cellules déjà groupées, et selon ce que l'on a décrit plus haut, juste avant que la dernière lettre du mot atteigne sa position groupée (donc au temps $(k + n/2 + m/2)$), on est dans la situation illustrée sur la figure 3.10.

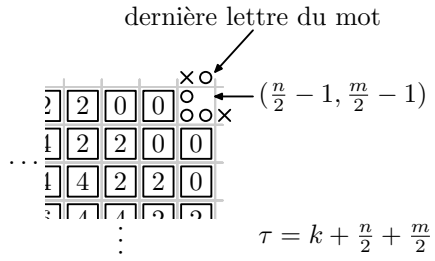

FIGURE 3.10 – Juste avant le groupage de la dernière lettre du mot.

Dans cette situation, la cellule $(n/2-1, m/2-1)$ est dans une configuration très particulière car elle peut passer immédiatement à un temps simulé de 2, sans avoir à faire le groupage au temps simulé 0 comme les autres. En effet puisqu'elle voit que la lettre qu'elle va recevoir est la dernière lettre du mot,

elle sait que ses voisines de droite et du haut ne portent aucune information utile, or ses deux autres voisines de gauche et du bas sont déjà groupées (au temps simulé 0). Ainsi, en une seule étape, la cellule $(n/2-1, m/2-1)$ a assez d'information pour obtenir tous les états de $\mathcal{A}$ au temps 0 sur toutes les positions qui lui correspondent mais également calculer les états au temps 2 sur ces mêmes positions.

Au temps $(k+n/2+m/2+1)$, le temps simulé de la cellule $(n/2-1, m/2-1)$ est donc 2. La figure 3.11 illustre alors les étapes suivantes (la cellule en haut à droite est toujours $(n/2-1, m/2-1)$). On voit que les voisines de la cellule $(n/2-1, m/2-1)$ peuvent directement calculer 4 étapes de l'automate $\mathcal{A}$ au temps suivant, puis leurs voisines à leur tour au temps d'après et ainsi de suite.

De manière générale, le temps simulé des cellules (i, j) sur la diagonale telle que $(i+j) = (n/2+m/2-2-d)$ est $(2k-2)$ au temps $(k+n/2+m/2+d)$, et $(2k+2)$ au temps $(k+n/2+m/2+d+1)$ (la diagonale qui vient de calculer 4 générations est indiquée en gris sur la figure).

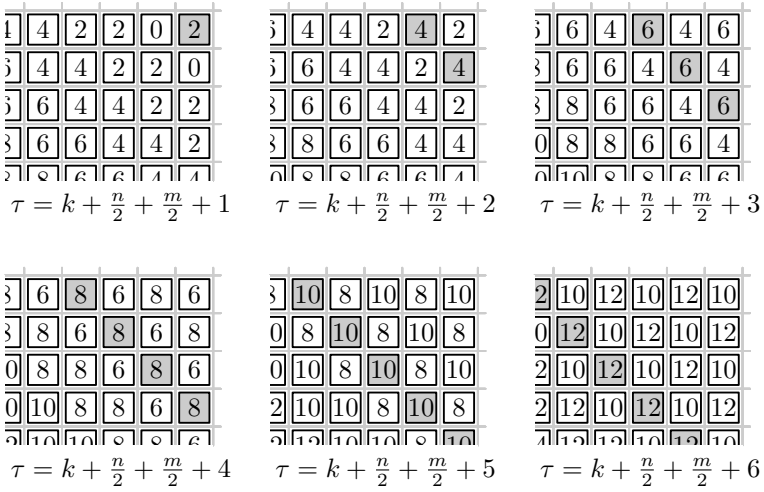

FIGURE 3.11 – Après le groupage de la dernière lettre.

Par ailleurs, une fois qu'une cellule a calculé 4 générations de $\mathcal{A}$ en un temps, elle calcule ensuite de nouveau 4 générations tous les deux temps, ce qui nous permet enfin de réaliser une simulation véritablement accélérée (auparavant, la simulation était limitée par le fait que toutes les cellules n'étaient pas correctement groupées).

Les observations sur la progression de la simulation que l'on vient de faire nous permettent finalement de savoir qu'au temps $(k+n+m-2)$, soit une génération avant le « temps réel plus k », le temps simulé de l'origine est $(n+m-6)$. En ce qui concerne ses deux voisines $(1, 0)$ et $(0, 1)$ qui sont sur la diagonale des cellules (i, j) vérifiant

$$(i+j) = 1 = (n/2+m/2-2) - (n/2+m/2-3)$$

correspondant donc à l'entier $d = (n/2+m/2-3)$ selon les notations du paragraphe précédent, leur temps simulé au temps

$$k + n/2 + m/2 + (n/2+m/2-3) + 1 = (k+n+m-2)$$

est

$$2(n/2 + m/2 - 3) + 2 = n + m - 4$$

Si $k = 0$, on est alors dans la situation illustrée sur la figure 3.12.

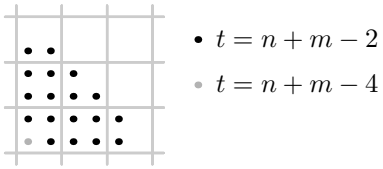

FIGURE 3.12 – La situation autour de l'origine au temps $(k + n + m - 2)$ pour $k = 0$.

Sur cette figure, on a représenté tous les états que l'origine voit. Les points noirs sont les états portés par les voisines de l'origine et représentent donc les états de l'automate $\mathcal{A}$ au temps $(n + m - 4)$ tandis que l'état représenté en gris est un état qui ne se trouve que sur la cellule origine et il correspond donc à l'état sur la cellule origine au temps $(n + m - 6)$ dans le fonctionnement de $\mathcal{A}$. De plus, par hypothèse de construction, les cellules voisines de l'origine (et l'origine elle-même d'ailleurs) contiennent tous les états correspondant aux positions marquées au temps $(n + m - 6)$.

L'état se trouvant sur l'origine au temps $(n+m-4)$ dans le fonctionnement de $\mathcal{A}$ peut facilement être déduit de ces informations. L'origine voit donc, une génération avant le temps réel, tous les états se trouvant sur un voisinage quadruple de l'origine au temps $(n+m-4)$, et peut donc calculer l'état dans lequel se trouve l'origine dans le fonctionnement de $\mathcal{A}$ au temps $(n + m)$.

La figure 3.13 illustre le cas général pour $k \geq 1$. Ici on a représenté les zones de connaissance des deux voisines de l'origine au temps $(k + n + m - 2)$ en pointillés gris. Le temps simulé de ces deux cellules est alors $(n + m - 4)$ et la cellule origine de $\mathcal{A}'$ dispose donc de tous les états dans le voisinage à l'ordre $(k+4)$ de l'origine de $\mathcal{A}$ (délimité par un trait plein noir) au temps $(n+m-4)$. L'origine de $\mathcal{A}'$ peut donc, au temps $(k+n+m-1)$ connaître l'état de l'origine de $\mathcal{A}$ au temps $(n + m + k)$.

On a donc bien montré que pour tout $k \in \mathbb{N}$, sur l'entrée $w \in \Sigma^{**}$, la cellule origine de l'automate $\mathcal{A}'$ connaît, au temps $\mathrm{TR}_{V_{\mathrm{VN}}}(n, m) + 1 + k = (n+m-1+k)$, l'état dans lequel se trouverait l'origine sur l'automate $\mathcal{A}$ après $\mathrm{TR}_{V_{\mathrm{VN}}}(n, m) + 2 + k = (n + m + k)$ générations.

Dans le cas d'une entrée de taille paire (selon les deux dimensions), si l'automate $\mathcal{A}$ reconnaît L en temps $(\mathrm{TR}_{V_{\mathrm{VN}}} + k + 2)$, l'automate $\mathcal{A}'$ que l'on vient de décrire reconnaît L en temps $(\mathrm{TR}_{V_{\mathrm{VN}}} + k + 1)$.

3.2.5 Réaliser des arrondis favorables

La construction que nous avons décrite ne fonctionne correctement que lorsque le mot en entrée est de dimensions paires. Dans le cas contraire il y a un petit délai lorsque la dernière lettre du mot est groupée et l'information n'arrive pas à l'origine à temps.

Cependant il est possible de construire l'automate $\mathcal{A}'$ pour qu'il fonctionne également sur les mots de longueur ou hauteur impaire. Étant donné un langage

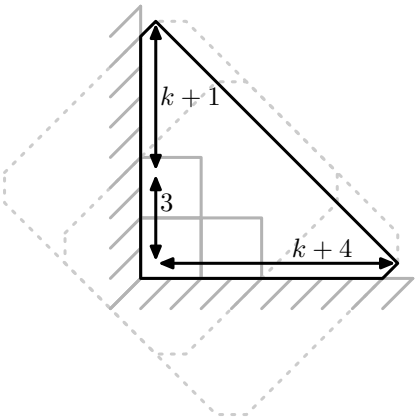

FIGURE 3.13 – La situation autour de l'origine au temps $\tau = k+n+m-2$ pour $k \geq 1$.

$L \subseteq \Sigma^{**}$, définissons le langage L_h comme étant un langage sur l'alphabet $(\Sigma \cup \{\#\})^5$, tel que pour tout mot w de L de taille $n \times m$ avec $n \geq 2$ il y ait un mot w_h de taille $(n-1) \times m$ dans L_h défini par

$$w_h : \left\{ \begin{array}{c} [\![0, n-2]\!] \times [\![0, m-1]\!] \to (\Sigma \cup \{\#\})^5 \\ (i,j) \mapsto \left(\begin{array}{ccc} & w(i,j+1), & \\ w(i-1,j), & w(i,j), & w(i+1,j) \\ & w(i,j-1), & \end{array} \right) \end{array} \right.$$

(lorsque $w(i,j)$ n'est pas défini, on met le symbole $\#$, que l'on suppose ne pas appartenir à Σ)

Ainsi, bien que le mot w_h ait une colonne de moins que w il contient la même information. De plus tout mot de L excepté les mots de largeur 1 se retrouvent dans L_h sous la forme d'un mot w_h.

Finalement, il reste à justifier que si L est reconnu en temps $(\mathrm{TR}_{V_{\mathrm{VN}}} + k)$ par un automate $\mathcal{A}$, le langage L_h est également reconnu en temps $(\mathrm{TR}_{V_{\mathrm{VN}}} + k)$.

Ceci se fait facilement, puisqu'un automate ayant en entrée le mot w_h peut appliquer la règle de transition de $\mathcal{A}$ à toutes les informations que contient une cellule. Pour la plupart des cellules ce processus est similaire à un groupage selon le voisinage, et pour ce qui est des cellules à droite du mot, leurs voisines n'ont pas l'information groupée qu'elles devraient avoir mais cette information n'a plus d'importance parce que la partie significative (les lettres du mot) est portée par les cellules du mot w_h et que les autres informations sont nécessairement des états B. Ainsi, l'automate peut simuler le fonctionnement de $\mathcal{A}$ sans perte de temps, tout en conservant sur chaque cellule tout un voisinage d'états.

Ainsi, si $\mathcal{A}$ reconnaissait le mot w de taille (n,m) en temps

$$\mathrm{TR}_{V_{\mathrm{VN}}}(n,m) + k = (n+m-2+k)$$

l'automate que l'on construit aura calculé, sur l'entrée w_h de taille $((n-1),m)$ au temps $(n+m+k-3)$ tous les états dans lesquels sont l'origine et ses voisines dans le fonctionnement de $\mathcal{A}$ et ces informations se trouveront sur l'origine. Ainsi, l'origine a assez d'information au temps

$$(n+m+k-3) = \mathrm{TR}_{V_{\mathrm{VN}}}((n-1),m) + k$$

pour conclure sur l'appartenance ou non du mot w à L et donc du mot w_h à L_h.

Remarquons enfin que, sur l'entrée $w \in L$, il est très facile pour un automate de construire le mot $w_h \in L_h$ en une génération (en effectuant un groupage au temps 1 et en effaçant la dernière colonne du mot).

On peut définir de manière similaire les langages L_v et L_{hv}, le premier correspondant à des mots ayant une ligne de moins que les mots de L et le second à des mots ayant une ligne et une colonne de moins que les mots de L. Chacun de ces langages est reconnu en temps $(\mathrm{TR}_{V_{\mathrm{VN}}} + k)$.

Revenons enfin à l'automate $\mathcal{A}'$. Tout mot $w \in L$ de taille (n, m) est dans l'un des cas suivants :

- n et m sont pairs. Dans ce cas la construction initialement exposée permet d'anticiper une étape de $\mathcal{A}$ en temps $(\mathrm{TR}_{V_{\mathrm{VN}}} + k + 1)$;
- $n \geq 2$ est impair et m est pair. Dans ce cas le mot w_h est de dimensions paires. Si $\mathcal{A}'$ commence par construire w_h en une génération puis que l'on applique la construction exposée précédemment, on arrive ici encore à anticiper la reconnaissance de w d'une étape ;
- n est pair et $m \geq 2$ est impair. Ce cas est le symétrique du précédent. Il se résout de manière similaire ;
- $n \geq 2$ est impair et $m \geq 2$ aussi. Dans ce cas, on calcule w_{hv} en deux étapes, puis on applique la construction précédente ;
- si l'une des deux dimensions vaut 1, on est dans le cas d'un mot unidimensionnel et l'on peut donc appliquer la méthode usuelle d'accélération par une constante en dimension 1.

Quelle que soit la situation initiale, nous avons donc un moyen d'anticiper la reconnaissance de w d'une étape. Si l'on effectue toutes ces constructions en parallèle (par un produit cartésien d'automates, chacun se chargeant d'une éventualité), on peut facilement savoir au temps réel dans quel cas se trouvait le mot en entrée (savoir en temps réel la parité de la longueur et la hauteur d'un mot est très facile) ce qui nous permet donc d'obtenir un automate unique capable d'accélérer la reconnaissance du mot w d'une étape, ce qui achève la preuve du théorème 3.2.1.

Chapitre 4

Accélération constante partielle et accélération linéaire

"I know some good games we could play,"
Said the cat.
"I know some new tricks,"
Said the Cat in the Hat.
"A lot of good tricks.
I will show them to you.
Your mother
Will not mind at all if I do."

Dr. Seuss – *The Cat in the Hat*

Dans le chapitre précédent, nous avons montré que tout voisinage reconnu en temps $(\mathrm{TR}_{V_{\mathrm{VN}}} + k)$ par un automate cellulaire fonctionnant sur le voisinage de Von Neumann en dimension 2 était reconnu en temps $(\mathrm{TR}_{V_{\mathrm{VN}}} + 1)$ par un automate cellulaire fonctionnant également sur le voisinage de Von Neumann. Ce résultat a été obtenu par des méthodes voisines de celles employées usuellement pour obtenir un théorème d'accélération linéaire.

Dans ce chapitre nous reviendrons sur les théorèmes d'accélération par une constante et montrerons qu'il existe plusieurs types de voisinages pour lesquels on peut obtenir des accélérations constantes partielles (et parfois même totales) par différents moyens. Nous verrons notamment qu'un résultat d'accélération linéaire implique une accélération constante partielle.

Nous nous intéresserons alors au problème de l'accélération linéaire en généralisant la méthode usuelle à une classe plus vaste de voisinages. Puis nous verrons que certains voisinages semblent ne pas permettre d'accélération linéaire (du moins pas selon les méthodes usuelles) et étudierons en détail les différentes conditions qui semblent poser problème.

4.1 Accélérations constantes partielles et totales

Dans cette section nous présenterons trois techniques permettant d'étendre des résultats d'accélération constante d'un voisinage à une classe de voisinages « dérivés » du premier.

Définition 4.1.1
On dira d'un voisinage V en dimension $d \in \mathbb{N}^*$ qu'il permet une accélération constante totale *si pour tout entier k tout langage reconnu en temps $\mathrm{TR}_V + k$ par un automate cellulaire fonctionnant sur le voisinage V est reconnu en temps TR_V par un automate fonctionnant également sur le voisinage V.*

Définition 4.1.2
On dira d'un voisinage V en dimension $d \in \mathbb{N}^*$ qu'il permet une accélération constante partielle d'ordre τ *si pour tout entier k tout langage reconnu en temps $(\mathrm{TR}_V + k)$ par un automate cellulaire fonctionnant sur le voisinage V est reconnu en temps $(\mathrm{TR}_V + \tau)$ par un automate fonctionnant également sur le voisinage V.*

Remarque. Une accélération constante totale est une accélération constante partielle d'ordre 0.

4.1.1 Puissances d'un voisinage

Lemme 4.1.1
Pour tout voisinage V et tout entier $p \geq 1$, on a

$$\mathrm{TR}_{V^p} = \left\lceil \frac{\mathrm{TR}_V}{p} \right\rceil$$

Preuve : Soit w un mot de taille (n, m). On note $M = [\![0, n-1]\!] \times [\![0, m-1]\!]$. On sait, par définition du temps réel, que $M \subseteq (V^p)^{\mathrm{TR}_{V^p}(w)}$ ce qui signifie que $p.\mathrm{TR}_{V^p}(w) \geq \mathrm{TR}_V(w)$ et donc

$$\mathrm{TR}_{V^p}(w) \geq \left\lceil \frac{\mathrm{TR}_V(w)}{p} \right\rceil$$

Par ailleurs, on a $M \subseteq V^{\mathrm{TR}_V(w)}$ soit encore $M \subseteq (V^p)^{\lceil \mathrm{TR}_V(w)/p \rceil}$ et donc

$$\left\lceil \frac{\mathrm{TR}_V(w)}{p} \right\rceil \geq \mathrm{TR}_{V^p}(w)$$

□

On a alors la proposition suivante :

Proposition 4.1.1
Soit V un voisinage en dimension d. Si V permet une accélération constante partielle d'ordre τ alors pour tout entier p le voisinage V^p permet une accélération constante partielle d'ordre $\lceil \tau/p \rceil$.

Preuve : Soit L un langage reconnu en temps $\mathrm{TR}_{V^p} + k$ par un automate cellulaire fonctionnant sur le voisinage V^p. Puisque tout automate fonctionnant sur le voisinage V^p peut être simulé en p fois plus de temps par un automate cellulaire fonctionnant sur le voisinage V, le langage L est reconnu en temps

$$p.\mathrm{TR}_{V^p} + p.k = p \left\lceil \frac{\mathrm{TR}_V}{p} \right\rceil + p.k \leq \mathrm{TR}_V + p.k + 1$$

par un automate fonctionnant sur le voisinage V. Puisque V permet une accélération constante partielle d'ordre τ, le langage L est reconnu en temps $(\mathrm{TR}_V + \tau)$ par un automate cellulaire fonctionnant sur V.

Puisque l'on peut simuler p générations d'un automate cellulaire fonctionnant sur V avec un automate fonctionnant sur V^p, le langage L est reconnu en temps

$$\left\lceil \frac{\mathrm{TR}_V + \tau}{p} \right\rceil \leq \left\lceil \frac{\mathrm{TR}_V}{p} \right\rceil + \left\lceil \frac{\tau}{p} \right\rceil \leq \mathrm{TR}_{V^k} + \left\lceil \frac{\tau}{p} \right\rceil$$

par un automate cellulaire fonctionnant sur le voisinage V^p. □

Réciproquement, on a la proposition :

Proposition 4.1.2
Soit V un voisinage en dimension d et $p \geq 1$ un entier. Si V^p permet une accélération constante partielle d'ordre τ alors le voisinage V permet une accélération constante partielle d'ordre $p(\tau + 1)$.

Preuve : Soit L un langage reconnu en temps $(\mathrm{TR}_V + k)$ sur V. L est reconnu en temps

$$\left\lceil \frac{\mathrm{TR}_V + k}{p} \right\rceil \leq \left\lceil \frac{\mathrm{TR}_V}{p} \right\rceil + \left\lceil \frac{k}{p} \right\rceil \leq \mathrm{TR}_{V^p} + \left\lceil \frac{k}{p} \right\rceil$$

sur V^p. L'accélération partielle sur V^p nous indique alors que L est reconnu en temps $(\mathrm{TR}_{V^p} + \tau)$ sur V^p et donc en temps

$$p.\mathrm{TR}_{V^p} + p.\tau = p \left\lceil \frac{\mathrm{TR}_V}{p} \right\rceil + p.\tau \leq \mathrm{TR}_V + p(\tau + 1)$$

sur V. □

Remarque. La proposition 4.1.1 est plus forte que la proposition 4.1.2 car elle n'induit pas de « perte de temps » due à des arrondis. Ainsi, si V permet une accélération constante totale ($\tau = 0$), V^p également. Par contre, si V^p permet une accélération constante totale, on sait uniquement que V admet une accélération partielle d'ordre p.

4.1.2 Enveloppes convexes

La proposition 2.3.1 nous permet également d'obtenir des résultats d'accélération constante partielle :

Proposition 4.1.3
Soit V un voisinage complet en dimension d, si $\mathrm{CH}(V)$ permet une accélération constante partielle d'ordre τ alors V permet également une accélération partielle d'ordre τ.

Preuve : Soit L un langage reconnu en temps $(\mathrm{TR}_V + k)$ sur V. On sait qu'il existe une constante t_V telle que pour tout mot w on ait

$$\mathrm{TR}_{\mathrm{CH}(V)}(w) \leq \mathrm{TR}_V(w) \leq \mathrm{TR}_{\mathrm{CH}(V)}(w) + t_V$$

Puisque $V \subseteq \mathrm{CH}(V)$ le langage L est reconnu sur $\mathrm{CH}(V)$ en temps $(\mathrm{TR}_V + k)$ également et donc en temps $(\mathrm{TR}_{\mathrm{CH}(V)} + k + t_V)$. Si $\mathrm{CH}(V)$ permet une accélération partielle d'ordre τ, L est donc reconnu sur $\mathrm{CH}(V)$ en temps $(\mathrm{TR}_{\mathrm{CH}(V)} + \tau)$.

D'après la proposition 2.3.1 le langage L est donc reconnu en temps $(\mathrm{TR}_{\mathrm{CH}(V)} + \tau)$ sur V si $\mathrm{TR}_{\mathrm{CH}(V)} + \tau \geq \mathrm{TR}_V$, sinon L est reconnu en temps TR_V.

Dans les deux cas, on a montré que L était reconnu en temps au plus $(\mathrm{TR}_V + \tau)$. □

La proposition 4.1.3 est très intéressante car il peut arriver qu'une accélération partielle sur $\mathrm{CH}(V)$ permette d'obtenir une accélération totale sur V. En effet si l'on considère par exemple le voisinage V_0 illustré sur la figure 4.1 il est immédiat qu'il est complet et que son enveloppe convexe est $(V_{\mathrm{VN}})^2$.

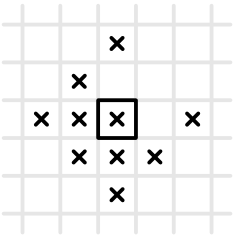

FIGURE 4.1 – Un voisinage surprenant (l'origine est encadrée en noir).

De plus si l'on regarde les puissances de ce voisinage elles ont toutes la forme illustrée sur la figure 4.2. On remarque que V_0^k contient toujours $({V_{\mathrm{VN}}}^2)^{k-1}$ (représenté en trait pointillés sur la figure) et que bien évidemment il est inclus dans $({V_{\mathrm{VN}}}^2)^k$. Par ailleurs aucun point de la diagonale représentée en gris (la diagonale $\{(x, y) \mid x + y = 2k - 1\}$) n'appartient à V_0^k.

Cela signifie que tout mot $w \in \Sigma^{**}$ tel que $\mathrm{TR}_{V_0}(w) = k$ s'étend sur un rectangle qui ne rencontre pas la diagonale « grise ». Il est donc recouvert par $({V_{\mathrm{VN}}}^2)^{k-1}$ d'où

$$\mathrm{TR}_{{V_{\mathrm{VN}}}^2}(w) \leq k - 1$$

Réciproquement, si $\mathrm{TR}_{{V_{\mathrm{VN}}}^2}(w) = k - 1$ alors puisque $({V_{\mathrm{VN}}}^2)^{k-1}$ est inclus dans V_0^k on a

$$\mathrm{TR}_{V_0}(w) \leq k$$

On a donc montré que $\mathrm{TR}_{V_0} = \mathrm{TR}_{{V_{\mathrm{VN}}}^2} + 1$.

Or le théorème 3.2.1 et la proposition 4.1.1 nous indiquent que le voisinage ${V_{\mathrm{VN}}}^2$ permet une accélération partielle d'ordre 1.

Considérons alors un langage L reconnu en temps $(\mathrm{TR}_{V_0} + k)$ pour un certain entier k par un automate cellulaire fonctionnant sur V_0. L est bien évidemment reconnu en temps $(\mathrm{TR}_{V_0} + k) = (\mathrm{TR}_{{V_{\mathrm{VN}}}^2} + k + 1)$ par un automate fonctionnant sur V_{VN} et donc en temps $(\mathrm{TR}_{{V_{\mathrm{VN}}}^2} + 1)$ par un certain automate $\mathcal{A}$ sur ${V_{\mathrm{VN}}}^2$.

Finalement, la proposition 2.3.1 nous indique que L est reconnu en temps $\mathrm{TR}_{{V_{\mathrm{VN}}}^2} + 1 = \mathrm{TR}_{V_0}$ sur le voisinage V_0.

Ainsi, étonnamment, nous sommes capables de montrer un véritable théorème d'accélération constante sur le voisinage V_0 alors que le résultat utilisé sur

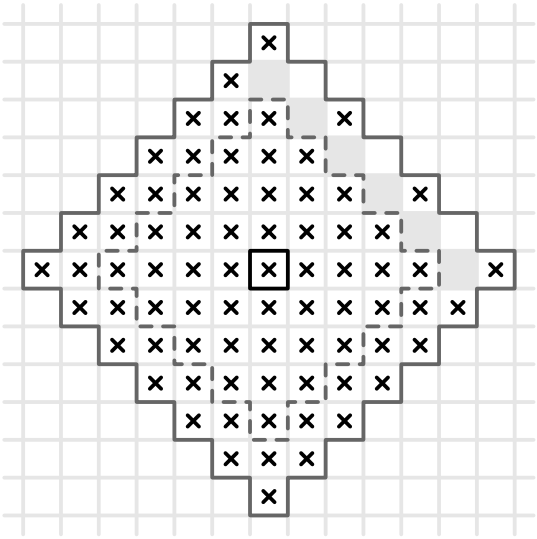

FIGURE 4.2 – La forme générique de V_0^k (ici $k = 3$).

$V_{\text{VN}}{}^2$ était partiel. Notons enfin qu'une réciproque du théorème 2.3.1 indiquant que tout langage reconnu en temps réel sur un voisinage V en dimension d est également reconnu en temps réel sur le voisinage $\text{CH}(V)$ permettrait d'obtenir un résultat d'accélération constante totale sur tout voisinage V tel qu'il existe un voisinage V' de même enveloppe convexe que V vérifiant $\text{TR}_{V'} \geq \text{TR}_{\text{CH}(V)} + 1$, ce qui n'est cependant pas le cas de tous les voisinages (c'est par exemple faux pour le voisinage de Von Neumann).

4.1.3 Accélération linéaire

Définition 4.1.3
On dira d'un voisinage V en dimension d qu'il permet une accélération linéaire s'il existe une constante C telle que pour toute fonction $f : \mathbb{N}^d \to \mathbb{N}$, tout langage reconnu en temps $(\text{TR}_V + f)$ par un automate cellulaire fonctionnant sur V soit reconnu en temps

$$\text{TR}_V + \lceil f/2 \rceil + C$$

par un automate fonctionnant sur le voisinage V également.

Remarque. En appliquant plusieurs fois la propriété d'accélération linéaire, on montre qu'elle implique que pour tout entier $a \geq 1$ il existe un entier b tel que tout langage reconnu en temps $(\text{TR}_V + f)$ sur le voisinage V soit reconnu en temps $(\text{TR}_V + \lceil f/a \rceil + b)$ sur le voisinage V.

On a alors la proposition suivante :

Proposition 4.1.4
Soit V un voisinage en dimension d, si V permet une accélération linéaire alors il permet une accélération constante partielle.

Preuve : Soit L un langage reconnu en temps $(\mathrm{TR}_V + k)$ sur V. D'après l'hypothèse sur V, L est reconnu en temps

$$\mathrm{TR}_V + \left\lceil \frac{k}{2} \right\rceil + C$$

sur V où C est la constante apparaissant dans la définition de l'accélération linéaire. Si $k > 2C + 1$ on a montré que L était reconnu en temps $\mathrm{TR}_V + k'$ avec

$$k' = \left\lceil \frac{k}{2} \right\rceil + C < \left\lceil \frac{k}{2} \right\rceil + \frac{k-1}{2} \leq \left\lceil \frac{k}{2} \right\rceil + \left\lfloor \frac{k}{2} \right\rfloor \leq k$$

soit $k' < k$. On peut alors réitérer la transformation jusqu'à obtenir un point fixe. Finalement, on a montré que le langage L était reconnaissable en temps $\mathrm{TR}_V + 2C + 1$ par un automate cellulaire fonctionnant sur V. □

Remarque. La preuve du théorème 3.2.1 est basée sur cette même idée. Toutefois, pour obtenir une accélération partielle d'ordre 1 il était nécessaire de détailler le fonctionnement de l'accélération linéaire, et de traiter séparément les mots selon la parité de leur largeur et leur hauteur pour éviter de perdre du temps à cause des arrondis. On peut toutefois appliquer directement la proposition 4.1.4 au voisinage de Von Neumann en utilisant une méthode d'accélération linéaire simple. On obtient alors une accélération partielle d'ordre 3.

4.2 Accélération linéaire

La proposition 4.1.4 nous amène tout naturellement à nous intéresser aux théorèmes d'accélération linéaire afin d'obtenir des résultats d'accélération constante sur une plus grande classe de voisinages.

Les théorèmes d'accélération linéaire connus (que ce soit dans le cas des machines de Turing, des automates cellulaires en dimension 1 sur le voisinage standard ou d'autres modèles similaires) ont quasiment tous été obtenus par la même méthode :

Étant donné un voisinage complet V, on considère un langage L reconnu en temps $(\mathrm{TR}_V + f)$ par un automate cellulaire $\mathcal{A}$ fonctionnant sur le voisinage V. On montre alors que ce même langage L peut être reconnu par un automate cellulaire $\mathcal{A}'$ fonctionnant sur le voisinage V en temps $(\mathrm{TR}_V + \lceil f/2 \rceil + C)$ où C ne dépend que de V.

La construction de l'automate $\mathcal{A}'$ est fondée sur les mêmes idées que les constructions déjà décrites en dimension 1 sur le voisinage usuel et en dimension 2 sur le voisinage de Von Neumann (preuve du théorème 3.2.1). Ces constructions peuvent être séparées en deux phases :

- Dans un premier temps, l'automate effectue une compression par un facteur 2 du mot en entrée, c'est-à-dire que l'on se ramène à une configuration où chaque cellule de l'automate contient maintenant 2^d lettres du mot de départ (où d est la dimension de l'automate). Cette compression se fait en temps environ $\mathrm{TR}_V/2$ (à une constante près).

- Une fois que la compression est faite, on effectue une simulation du fonctionnement de l'automate $\mathcal{A}$ mais avec une accélération permettant de simuler 2 générations de $\mathcal{A}$ par génération de $\mathcal{A}'$.

Comme nous l'avons fait précédemment, nous ne travaillerons ici qu'en dimension 2 mais tous les résultats que nous obtiendrons peuvent être généralisés aux dimensions supérieures.

Remarque. En dimension 1 l'équivalence des voisinages et les théorèmes connus d'accélération linéaire sur les voisinages standard et *one-way* permettent d'obtenir immédiatement une accélération linéaire sur tout voisinage complet ou semi-complet. Il est également possible d'étendre ce résultat aux voisinages incomplets en remarquant que V^∞ est nécessairement périodique à partir d'un certain rang et en ne travaillant alors que sur les cellules appartenant à V^∞.

Étudions maintenant en détail chacune des deux phases.

4.2.1 La simulation

Remarque. On rappelle que dans tout le document, $\mathrm{CH}(V)$ désigne l'enveloppe convexe discrète de V, $V^2 = \{x + y \mid x, y \in V\}$ et $2V = \{2x \mid x \in V\}$ (les définitions de V^n et nV sont semblables). On rappelle également que $\mathrm{CH}(nV) = \mathrm{CH}(V^n) = \mathrm{CH}(V)^n$.

Bien que cela puisse sembler contre-intuitif il est préférable de traiter le cas de la simulation accélérée avant celui de la compression car cela permet de définir correctement quel doit être le résultat de la phase de compression pour que la simulation se déroule correctement.

Nous avons déjà vu dans la sous-section 3.2.3 que dans le cas du voisinage de Von Neumann, on ne pouvait pas se contenter de grouper les lettres par 4 sur une cellule car alors l'automate ne disposait pas d'assez d'information pour calculer deux générations de $\mathcal{A}$ (voir figure 3.7). Pour résoudre ce problème, il a été nécessaire d'effectuer un temps de stockage en début de construction. Dans le cas général, il est un peu plus compliqué de justifier qu'il est toujours possible d'effectuer un stockage tel que toute cellule voie assez d'information pour effectuer la simulation accélérée. On utilise le lemme suivant :

Lemme 4.2.1
Pour tout voisinage V en dimension 2, on a

$$V + \mathrm{CH}(2V) = \mathrm{CH}(3V)$$

Preuve : L'inclusion $V + \mathrm{CH}(2V) \subseteq \mathrm{CH}(3V)$ est immédiate. Pour ce qui est du sens inverse, il faut raisonner sur les cônes de l'enveloppe convexe : soient u et v deux sommets consécutifs de $\mathrm{CH}(V)$ (donc u et v appartiennent à V). Si l'on note $T_{u,v}$ le triangle $(0, u, v)$, on remarque tout d'abord que $2T_{u,v} \subseteq 2V$ et donc $\mathrm{CH}(2T_{u,v}) \subseteq \mathrm{CH}(2V)$.

Par ailleurs

$$\begin{array}{rcl} \mathrm{CH}(3T_{u,v}) & = & (\mathrm{CH}(2T_{u,v})) \cup (u + \mathrm{CH}(2T_{u,v})) \cup (v + \mathrm{CH}(2T_{u,v})) \\ & \subseteq & T_{u,v} + \mathrm{CH}(2T_{u,v}) \\ & \subseteq & V + \mathrm{CH}(2V) \end{array}$$

(voir figure 4.3, sur laquelle $\mathrm{CH}(2T_{u,v})$ est représenté en gris clair, $(u + \mathrm{CH}(2T_{u,v}))$ est hachuré et $(v + \mathrm{CH}(2T_{u,v}))$ est dessiné en trait épais, l'ensemble de la figure représentant $\mathrm{CH}(3T_{u,v})$)

Enfin, il suffit de remarquer que $\mathrm{CH}(3V)$ est l'union de tous les triangles $\mathrm{CH}(3T_{u,v})$ pour tous couples (u, v) de sommets consécutifs.

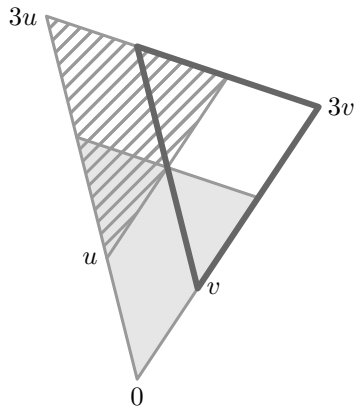

FIGURE 4.3 – Les triangles $\mathrm{CH}(2T_{u,v})$ (en gris clair), $(u+\mathrm{CH}(2T_{u,v}))$ (hachuré) et $(v+\mathrm{CH}(2T_{u,v}))$ (en trait épais) recouvrent le triangle $\mathrm{CH}(3T_{u,v})$.

On a donc

$$\begin{aligned}\mathrm{CH}(3V) &= \bigcup_{u,v} \mathrm{CH}(3T_{u,v}) \\ &\subseteq \bigcup_{u,v} [(\mathrm{CH}(2T_{u,v})) \cup (u+\mathrm{CH}(2T_{u,v})) \cup (v+\mathrm{CH}(2T_{u,v}))] \\ &\subseteq V+\mathrm{CH}(2V)\end{aligned}$$

□

Revenons maintenant au cas de la simulation accélérée de $\mathcal{A}$ par $\mathcal{A}'$. Le principe de la compression est similaire à ce que l'on a vu dans le cas du voisinage de Von Neumann, c'est-à-dire que la cellule c reçoit les lettres qui se trouvaient initialement sur les cellules $2c$, $2c+(1,0)$, $2c+(0,1)$ et $2c+(1,1)$ (comme illustré par la figure 3.4 dans le chapitre précédent). Notons

$$E = \{(0,0),(1,0),(0,1),(1,1)\}$$

la cellule c reçoit donc après compression les lettres qui se trouvaient initialement sur les cellules de $(2c+E)$, et elle continuera tout au long de la simulation de contenir les états qui, dans le fonctionnement de $\mathcal{A}$, se trouveraient sur les cellules de $(2c+E)$.

Dans le cas du voisinage de Von Neumann, nous avions ajouté un temps de stockage avant la compression de telle sorte que dès le début chaque cellule c connaisse en réalité toutes les lettres se trouvant initialement sur les cellules de $(c+V_{\mathrm{VN}})$. Ici nous allons utiliser plus de temps de stockage de manière à s'assurer que chaque cellule connaît correctement toutes les lettres correspondant aux positions de $(c+\mathrm{CH}(4V))$ (ce temps dépend du voisinage mais puisque V est complet, il existe $t\in\mathbb{N}$ tel que $\mathrm{CH}(4V)\subseteq V^t$).

Une fois cette période de stockage effectuée, on procède à la compression, puis à la simulation. Le but est de réussir à montrer que si à un temps donné chaque cellule c de l'automate $\mathcal{A}'$ connaît correctement tous les états dans lesquels seraient les cellules de $(c+E)$ ainsi que leurs informations groupées, soit finalement tous les états sur les cellules de $(c+E+\mathrm{CH}(4V))$, dans le fonctionnement de l'automate $\mathcal{A}$ au temps t, alors au temps suivant, la cellule c

de $\mathcal{A}$ connaît correctement tous les états dans lesquels sont ces mêmes cellules dans le fonctionnement de $\mathcal{A}$ au temps $(t+2)$, ce qui nous permettra donc bien d'effectuer la simulation de $\mathcal{A}$ deux fois plus vite.

On suppose que chaque cellule c de $\mathcal{A}'$ porte les états des cellules dans $(2c+E+\mathrm{CH}(4V))$ de $\mathcal{A}$. La cellule c « voit » donc (en regardant son voisinage) tous les états portés par les cellules de $(c+V)$ ce qui correspond alors aux positions

$$\mathrm{VISIBLE}(c) = 2(c+V)+E+\mathrm{CH}(4V) = 2c+E+2V+\mathrm{CH}(4V)$$

Or pour calculer deux générations de $\mathcal{A}$ sur les informations qu'elle porte, la cellule c doit être capable d'obtenir tous les états correspondant aux positions se trouvant dans le double voisinage des informations qu'elle porte. Elle doit donc être capable d'obtenir les états se trouvant sur les positions

$$\mathrm{NECESSAIRE}(c) = V^2 + (2c+E+\mathrm{CH}(4V)) = 2c+E+V^2+\mathrm{CH}(4V)$$

Or d'après le lemme 4.2.1

$$2V+\mathrm{CH}(4V) = \mathrm{CH}(6V) = \mathrm{CH}(3V^2) = V^2+\mathrm{CH}(2V^2) = V^2+\mathrm{CH}(4V)$$

et donc $\mathrm{VISIBLE}(c) = \mathrm{NECESSAIRE}(c)$ ce qui nous assure que si l'on effectue correctement les générations de stockage initiales et la compression, la simulation de $\mathcal{A}$ peut être réalisée par $\mathcal{A}'$ avec une accélération par un facteur 2.

Sur une entrée $w \in \Sigma^{**}$, la phase de simulation dure donc $(\mathrm{TR}_V(w)+\lceil f/2\rceil)$ générations. Si l'on veut obtenir un résultat d'accélération linéaire, il faut donc que la compression dure au plus $(\lceil f/2\rceil + C)$ générations, pour un certain entier C ne dépendant que de V.

Intéressons-nous maintenant à la compression.

4.2.2 La compression de l'entrée

Comme il a été déjà dit, le but de cette phase est de passer de la configuration initiale où chaque lettre du mot est sur une certaine cellule à une configuration où la cellule c contient les quatre lettres qui se trouvaient initialement sur les cellules $2c$, $2c+(1,0)$, $2c+(0,1)$ et $2c+(1,1)$. On ignorera ici les quelques générations initiales de stockage nécessaires mentionnées dans la section précédente puisqu'elles n'ont aucune incidence sur le reste du calcul (une fois que chaque cellule a fait son stockage, toute l'information se déplace en bloc et donc on peut se contenter de décrire le déplacement des lettres initiales).

Il est nécessaire que chaque lettre du mot (initialement sur la cellule $c = (x,y) \in \mathbb{Z}^2$) se déplace vers sa « cible »

$$\left\lfloor \frac{c}{2} \right\rfloor = \left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor \right)$$

en au plus $(\lceil \mathrm{TR}_V(w)/2\rceil + C)$ sur une entrée $w \in \Sigma^{**}$.

Nous montrerons dans cette section qu'il est possible d'effectuer une telle compression de l'entrée sur une certaine classe de voisinages restreintes par de fortes hypothèses (afin de simplifier la preuve), puis nous généraliserons le résultat à une plus vaste classe de voisinages.

Afin de décrire correctement les voisinages sur lesquels nous allons prouver qu'il est possible d'effectuer efficacement la compression, il est nécessaire de poser quelques définitions.

Définition 4.2.1
Étant donné un polygone convexe $P \in \mathbb{R}^2$ contenant un voisinage de l'origine, on dira que P a k sommets positifs si exactement k des sommets de P ont leurs deux coordonnées strictement positives. On dira d'un voisinage V qu'il a k sommets positifs si son enveloppe convexe continue $\mathrm{CHC}(V)$ *a k sommets positifs.*

Remarque. Cela signifie notamment que la restriction de la frontière du polygone P au quart de plan $(\mathbb{R}^+)^2$ est une ligne brisée constituée d'exactement $(k+1)$ segments. La figure 4.4 illustre cette définition.

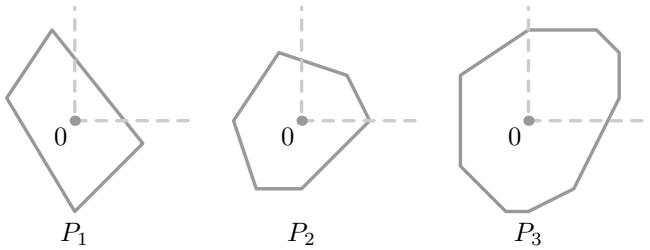

FIGURE 4.4 – Les polygones P_1, P_2 et P_3 ont respectivement 0, 1 et 3 sommets positifs (le quart de plan positif est délimité par des pointillés).

Définition 4.2.2
Soit V un voisinage complet convexe ayant 0 ou 1 sommet positif. On définit les vecteurs v_h, v_v et v_d de la manière suivante :

- *v_h est à l'intersection de l'axe des abscisses positives avec la frontière de* $\mathrm{CHC}(V)$*, soit*

$$v_h = (x_h, 0) \qquad \text{où} \qquad x_h = \max\{x \in \mathbb{R} \mid (x,0) \in \mathrm{CHC}(V)\}$$

- *v_v est à l'intersection de l'axe des ordonnées positives avec la frontière de* $\mathrm{CHC}(V)$*. Il est donc défini par*

$$v_v = (0, y_v) \qquad \text{où} \qquad y_v = \max\{y \in \mathbb{R} \mid (0,y) \in \mathrm{CHC}(V)\}$$

- *si V n'a pas de sommet dans le quart de plan positif, on pose $v_d = v_h$. Si par contre* $\mathrm{CHC}(V)$ *a un sommet de coordonnées strictement positives on appelle v_d ce sommet.*

Remarque. Les vecteurs v_h, v_v et v_d on des coordonnées rationnelles mais pas nécessairement entières.

La figure 4.5 illustre ces définitions.

On a alors la proposition suivante :

Proposition 4.2.1
Soit V un voisinage complet en dimension 2 ayant 0 ou 1 sommet positif. Si V est convexe, symétrique et que les vecteurs v_h, v_v et v_d tels qu'ils ont été définis précédemment appartiennent tous les trois à V, alors V permet une accélération linéaire.

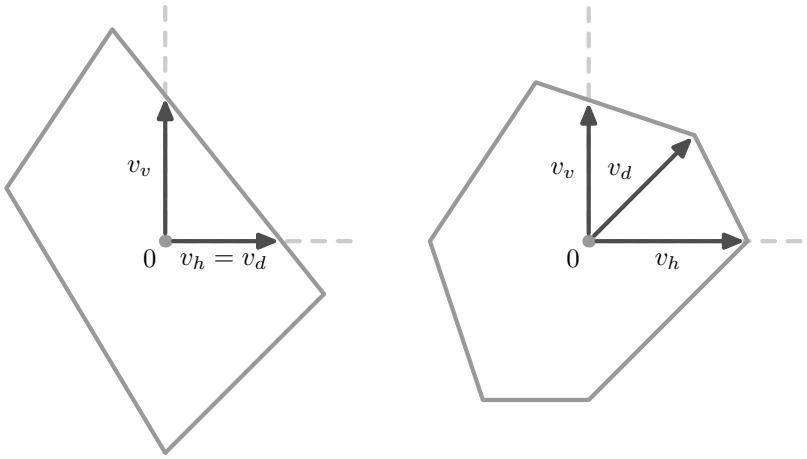

FIGURE 4.5 – Les vecteurs v_h, v_g et v_d dans le cas de voisinages ayant respectivement 0 (à gauche) et 1 (à droite) sommets positifs.

Pour montrer ce résultat, nous montrerons dans un premier temps que sur un voisinage V vérifiant les hypothèses de la proposition 4.2.1 il est possible de compresser efficacement l'entrée.

4.2.3 Une solution optimale

Nous présentons ici la construction d'un automate cellulaire en dimension 2 fonctionnant sur un voisinage V vérifiant les hypothèses de la proposition 4.2.1 qui, partant d'une configuration initiale correspondant à un mot $w \in \Sigma^{**}$ déplace chacune des lettres (toutes en parallèle) de la cellule c où elle se trouve initialement vers la cellule $\lfloor c/2 \rfloor$ où elle doit se trouver après compression.

La construction que nous présentons revient à faire ce que l'on a déjà décrit dans le cas du voisinage de Von Neumann dans le chapitre 3 mais nous regarderons les choses différemment. En effet au lieu de regarder le comportement global de la compression (où se trouvent toutes les informations à chaque temps et quelles cellules ont déjà reçu les 4 lettres qu'elles attendaient) comme nous l'avions fait précédemment nous adopterons le point de vue d'une unique lettre et verrons comment elle peut se déplacer vers sa cible, sans que son comportement dépende de ce que font les autres lettres.

Toutes les illustrations correspondent au cas d'un voisinage V ayant un sommet positif. Le cas où V n'a aucun sommet positif est un cas « dégénéré » du précédent puisqu'alors les vecteurs v_d et v_h coïncident et il peut être traité de la même manière.

Les signaux des axes

Au temps 0, chaque cellule de la configuration initiale « sait » si elle correspond à une lettre du mot ou si elle se trouve en dehors de la surface sur laquelle est écrit le mot. Puisqu'à aucun moment les informations ne sortent de la zone où le mot était initialement écrit, toute cellule initialement dans l'état B le reste tout au long du calcul. La première génération de la compression sert

à initialiser le marquage que nous utiliserons par la suite. Au cours de cette première génération, toutes les cellules qui se trouvent sur un axe (donc celles telles que leur voisine gauche ou inférieure est dans l'état B) sont marquées d'un état B_1. L'origine quant à elle est marquée d'un état particulier B_0.

À partir de là, à chaque temps suivant, toute cellule c telle que sa voisine $(c-v_d)$ est dans l'état B_1 prend l'état B_1 à son tour, et la cellule c dont la voisine $(c-v_d)$ est dans l'état B_0 est à son tour marquée B_0 (l'état B_0 n'est porté que par une seule cellule à tout instant et donc une cellule ne reste marquée par cet état que pendant une génération, après quoi elle devient une cellule B_1).

La zone des cellules marquées par l'état B_1 se propage donc suivant le vecteur v_d à chaque génération en dessinant des tranches en forme de « L » comme illustré sur la figure 4.6.

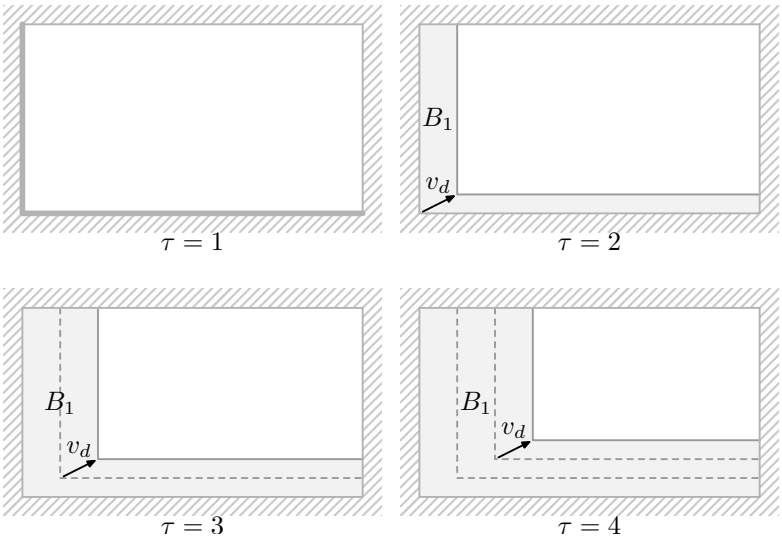

FIGURE 4.6 – La progression du marquage B_1.

Par ailleurs, la cellule marquée B_0 émet deux signaux que l'on appellera F se déplaçant respectivement selon les vecteurs v_h et v_v. La propagation du marquage B_0 est telle qu'au temps $(t+1)$ c'est la cellule $t.v_d$ qui émet ces deux signaux. La propagation de ces nouveaux signaux est illustrée sur la figure 4.7.

Les déplacements des lettres

Maintenant que nous avons expliqué comment les axes pouvaient mettre en place une série de signaux et marquages sur la surface du mot en entrée, voyons comment les lettres du mot peuvent s'en servir pour atteindre rapidement leur cible.

Pour ne pas désynchroniser les déplacements des lettres et ceux des signaux émis par les axes, les lettres attendent une génération avant de commencer à se déplacer (la génération pendant laquelle les cellules des axes sont marquées par B_1).

Si l'on considère une lettre initialement placée sur une cellule c, cette lettre va dans un premier temps se déplacer selon le vecteur $-v_d$. Elle se déplace ainsi

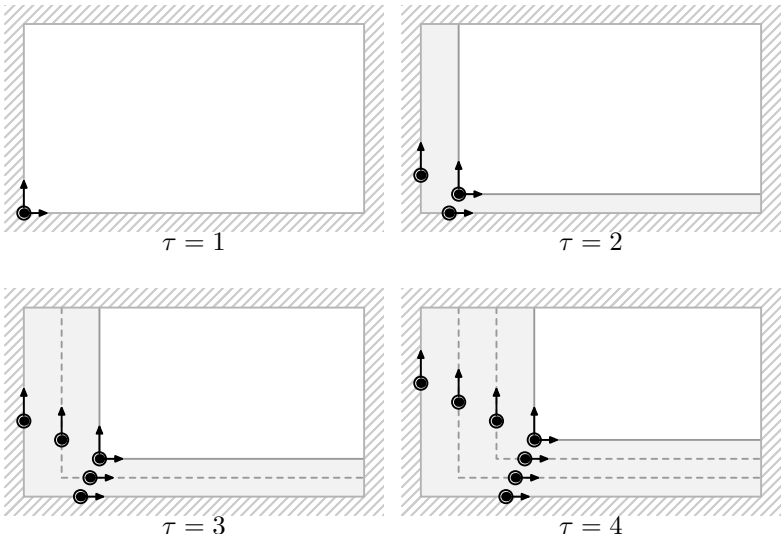

FIGURE 4.7 – La progression des signaux F se déplaçant selon v_h et v_v.

jusqu'à rencontrer une cellule marquée B_1 comme illustré sur la figure 4.8 à un instant t_1.

Sur la figure, on a représenté les différentes positions par lesquelles passe la lettre qui part de la cellule c (en indiquant au-dessus de chacune des positions le temps auquel la lettre s'y trouve). On a également représenté la progression de la zone de cellules B_1 en dessinant en traits gris pointillés les différentes frontières au cours du temps (également indexées par le temps auquel elles correspondent). Les signaux F représentés par des points noirs encerclés sont ceux qui se trouvent sur la configuration au temps où la lettre atteint la zone B_1 (sur la figure on a $t_1 = 5$).

Si la cellule c d'où la lettre est partie se trouve au-dessus de la diagonale $[0v_d)$ la frontière que la lettre rencontre est verticale, mais si c est en dessous de la diagonale alors la lettre rencontre une frontière horizontale. Dans la suite, nous supposerons que la cellule a rencontré une frontière horizontale pour simplifier la description. Le cas d'une frontière verticale est tout à fait symétrique.

Puisque $\lfloor c/2 \rfloor$ est le milieu du segment $[0, c]$, c'est également le milieu de tout segment de la forme $[0 + k.v_d, c - k.v_d]$. Ainsi à l'instant t_1 la cible de la lettre (représentée par une croix sur la figure) se trouve exactement au milieu du segment dont les extrémités sont d'une part la position actuelle de la lettre et d'autre part l'angle de la frontière de la zone B_1 où naissent les deux nouveaux signaux F.

En voyant la frontière de la zone B_1 la cellule peut donc déduire l'ordonnée de $\lfloor c/2 \rfloor$ relativement à sa position actuelle (dans le cas d'une frontière verticale elle déduit l'abscisse de $\lfloor c/2 \rfloor$) et s'en souvenir par la suite.

Au temps suivant, la lettre se place sur la frontière de la zone B_1 (la convexité de V nous assure qu'elle peut y arriver en une étape) comme illustré sur la figure 4.9. Pour se placer sur la frontière elle se déplace selon un vecteur $(x_1, y_1) \in V$ et elle mémorise l'abscisse x_1 de ce vecteur qui lui permettra par la suite de trouver l'abscisse de $\lfloor c/2 \rfloor$.

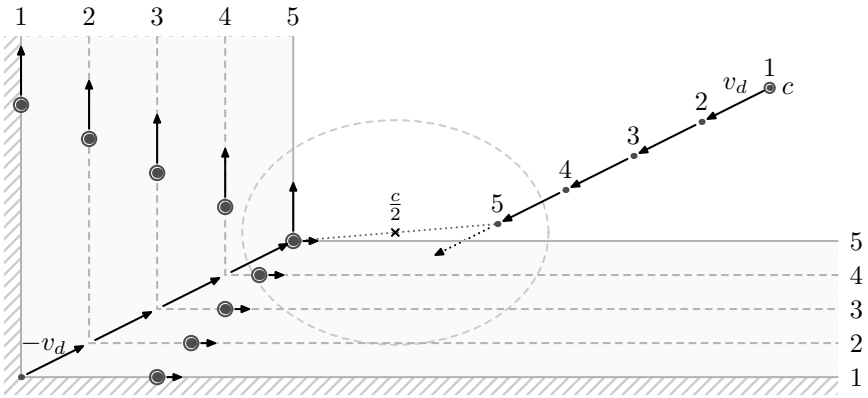

FIGURE 4.8 – Les premiers déplacements de la lettre initialement placée en c. On a ici $t_1 = 5$.

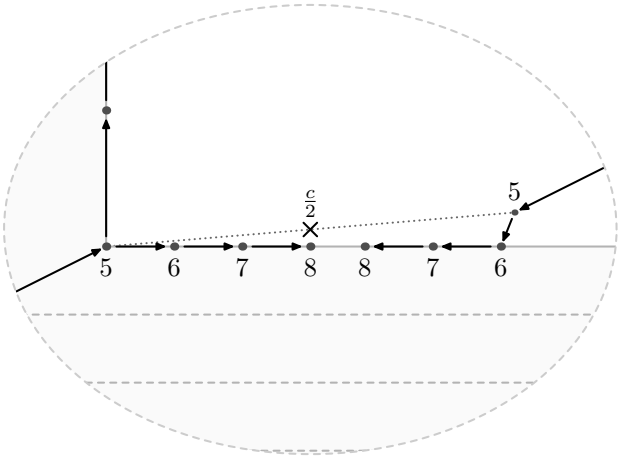

FIGURE 4.9 – La suite du déplacement de la lettre se trouvant initialement sur la cellule c.

La lettre se déplace ensuite selon le vecteur $-v_h$. Elle se déplace ainsi en sens inverse du signal F qui se trouve à la même hauteur et finit par le rencontrer en un certain point. Lorsque la lettre rencontre un signal F elle est capable de déterminer la position exacte de la cellule $\lfloor c/2 \rfloor$.

En effet au temps t_1 la cellule $\lfloor c/2 \rfloor$ se trouve exactement au milieu du segment délimité par la position de la lettre et la position du signal F qu'elle rencontrera par la suite. Par la suite la lettre se déplace selon (x_0, y_0) une fois puis k fois selon $-v_h$ (où k est une valeur que la lettre ne connaît pas) tandis que le signal F se déplace $(k+1)$ fois selon v_h. Si la lettre mémorise la valeur x_0 elle est capable de retrouver l'abscisse de $\lfloor c/2 \rfloor$ en voyant sa position par rapport au signal F.

Après avoir rencontré le signal F elle sait donc exactement où se trouve sa cible et elle peut donc aller s'y placer. Puisqu'il n'existe qu'un nombre fini de cas possibles, on peut borner le temps que mettra la lettre à atteindre sa cible après avoir rencontré le signal F.

Remarque. Dans le cas où la lettre rencontre une frontière verticale, elle se déplace ensuite selon $-v_v$ et rencontre alors le signal F qui se déplace selon v_v.

Remarque. Si c se trouve sur la diagonale $[0, v_d)$ la lettre rencontrera la zone B_1 au niveau de l'angle et donc elle rencontrera le signal F immédiatement sans avoir à se déplacer selon aucun des deux vecteurs $-v_h$ ou $-v_v$, cette éventualité correspond donc à un cas dégénéré des deux situations précédentes.

Durée de la compression

Récapitulons les déplacements qu'effectue la lettre qui se trouve initialement sur la cellule c :

- la première génération est perdue, car on attend que l'initialisation des marquages sur les axes soient faits ;
- puis la lettre se déplace un certain nombre de fois selon le vecteur $-v_d$;
- en atteignant la zone B_1 la lettre effectue un déplacement pour se placer sur la frontière de la zone B_1 ;
- la lettre se déplace alors un certain nombre de fois selon $-v_h$;
- lorsque la lettre rencontre le signal F il ne lui reste plus qu'à effectuer un nombre borné (en fonction de V) de déplacements pour atteindre sa cible $\lfloor c/2 \rfloor$.

La lettre se déplace donc de c à $\lfloor c/2 \rfloor$ en effectuant tous ses déplacements sauf un nombre borné (notons α cette borne sur le nombre de déplacements « parasites ») selon les vecteurs $-v_d$ et $-v_h$ qui sont par construction sur le même segment de la frontière de l'enveloppe convexe de V.

Ainsi, si la lettre met τ générations à atteindre $\lfloor c/2 \rfloor$, cela signifie que

$$\frac{c}{2} \in c + \{-v_d, -v_h\}^{\tau-\alpha} + V^{\alpha}$$

Pour montrer l'optimalité de notre construction, on définit la distance associée à V par :

$$d_V : \left\{ \begin{array}{rcl} \mathbb{Z}^2 & \to & \mathbb{N} \\ (c, c') & \mapsto & \min\{t \in \mathbb{N} \mid c' \in V^t(c)\} \end{array} \right.$$

Dans le cas général d_V n'est pas symétrique mais ici puisque V est symétrique on a

$$c' \in V^t(c) \Leftrightarrow c \in V^t(c')$$

et donc d_V est bien une distance. On étend alors naturellement la définition aux parties de $\mathbb{Z}^2$ par

$$\forall E, E' \subseteq \mathbb{Z}^2, \qquad d_V(E, E') = \min\{d_V(x, y) \mid x \in E, y \in E'\}$$

qui n'est alors qu'une pseudo-distance car elle n'est pas antisymétrique.

On a les deux propositions suivantes :

Proposition 4.2.2
Soient u et v deux vecteurs de V appartenant à un même segment de la frontière de l'enveloppe convexe continue de V. Alors
$\forall c \in \mathbb{Z}^2, \quad \forall a, b \in \mathbb{N},$

$$d_V(c, c + a.u + b.v) = a + b$$

Preuve : Il faut remarquer que si l'on note S un segment de la frontière d'un polygone convexe P contenant un voisinage de l'origine et x un point sur ce segment, alors on a trivialement

$$(x + S) \subseteq S^2$$

et donc $(x + S)$ est inclus dans un segment de la frontière de P^2.

Ainsi, dans le cas qui nous intéresse, on montre inductivement que pour tous $a, b \in \mathbb{N}$, $a.u + b.v$ est sur la frontière de $\mathrm{CHC}(V^{a+b})$ et donc puisque V contient un voisinage de l'origine, aucun point de la frontière de $\mathrm{CHC}(V^{a+b})$ n'appartient à V^{a+b-1}. Or il est immédiat que $a.u + b.v \in V^{a+b}$ et donc

$$d_V(c, c + a.u + b.v) = a + b$$

□

Proposition 4.2.3
$\forall x \in \mathbb{Z}^2, \quad \forall E, F \subseteq \mathbb{Z}^2, \quad \forall k, k' \in \mathbb{N}$

$$\left.\begin{array}{rcl} \forall f \in F, \exists e \in E \quad d_V(e, f) & \leq & k \\ \text{et } \forall e \in E \quad d_V(x, e) & \geq & k' \end{array}\right\} \Rightarrow d_V(x, F) \geq k' - k$$

Preuve : Par l'absurde. Supposons $d_V(x, F) < k' - k$ alors il existe $f \in F$ tel que $d_V(x, f) < k' - k$. Or par hypothèse il existe $e \in E$ tel que $d_V(e, f) \leq k$ et donc par inégalité triangulaire on a la contradiction

$$d_V(e, x) < k' - k + k = k'$$

□

Revenons maintenant à la compression. Nous savons que si la lettre se trouvant initialement sur la cellule c arrive sur la cellule $\lfloor c/2 \rfloor$ après τ générations on a

$$\left\lfloor \frac{c}{2} \right\rfloor \in c + \{-v_d, -v_h\}^{\tau-\alpha} + V^{\alpha}$$

Or d'après la proposition 4.2.2 on a

$$d_V(c, c + \{-v_d, -v_h\}^{\tau-\alpha}) = \tau - \alpha$$

Par ailleurs, par définition de la distance d_V on a
$\forall x \in c + \{-v_d, -v_h\}^{\tau-\alpha} + V^\alpha, \quad \exists y \in c + \{-v_d, -v_h\}^{\tau-\alpha},$

$$d_V(x, y) \leq \alpha$$

et donc finalement d'après la proposition 4.2.3 on a

$$d_V\left(c, \left\lfloor \frac{c}{2} \right\rfloor\right) \geq d_V(c, c + \{-v_d, -v_h\}^{\tau-\alpha} + V^\alpha) \geq \tau - 2\alpha$$

ce qui signifie que la lettre qui se trouve initialement sur la cellule c se déplace jusqu'en $\lfloor c/2 \rfloor$ en un temps optimal à 2α générations près.

Or le temps optimal pour aller de c à $\lfloor c/2 \rfloor$ est le plus petit temps t tel que $\lfloor c/2 \rfloor \in V^t(c)$. Puisque V est convexe, ce temps est égal à la moitié (arrondie au-dessus) du temps t' qu'il faut pour que l'origine soit dans $V^{t'}(c)$ et puisque V est symétrique c'est la moitié du plus petit temps t' tel que $c \in V^{t'}$.

Finalement, on a montré que pour toute cellule c si c reçoit une lettre dans la configuration initiale cette lettre peut atteindre la cellule $\lfloor c/2 \rfloor$ en au plus la moitié du temps réel correspondant au mot en entrée plus une constante.

Parallélisation

Il ne reste plus qu'à vérifier qu'il est possible de déplacer simultanément toutes les lettres du mot comme cela a été décrit.

Le marquage à partir des axes et les signaux F ne posent aucun problème puisqu'ils servent à toutes les lettres à la fois.

Lorsque l'on déplace les lettres selon $-v_d$ cela ne pose pas de problème puisque cela correspond à une translation de la totalité de la configuration. Cependant, lorsque l'on arrive dans la zone B_1 les choses se compliquent un peu. Tout d'abord toutes les cellules qui devraient aller dans la zone B_1 à un instant donné se placent sur la frontière ce qui implique que les informations se superposent sur une même cellule. De plus chaque lettre qui se place sur la frontière de la zone B_1 doit mémoriser l'ordonnée relative de sa cible et l'abscisse du déplacement qu'elle effectue pour se placer sur la frontière qui lui permettront de trouver sa cible par la suite.

Puisque les déplacements ont lieu selon les vecteurs de V, on peut majorer le nombre de lettres qui se placeront sur une cellule de la frontière de la zone B_1 à un instant donné. Puisque la frontière de la zone B_1 avance avec le temps les lettres ne s'amalgament qu'une seule fois sur une cellule. Ainsi, puisqu'il n'arrive qu'une quantité d'information bornée sur une même cellule cela ne pose pas de problème.

Les lettres se déplacent ensuite selon les vecteurs $-v_h$ et $-v_v$ ce qui ne pose pas de problème supplémentaire car toutes les informations se déplacent parallèlement et à la même vitesse.

Enfin, lorsqu'une lettre (ou dans le cas parallèle, un groupe de lettres) rencontre le signal F qui se déplace en sens inverse toutes les lettres sont libérées et chacune se déplace à vitesse maximale vers sa cible. Ici encore tout ceci a lieu dans un périmètre borné et le nombre de lettres mis en jeu ainsi que le

temps avant qu'elles atteignent leur cible et se fixent définitivement sont bornés également.

Ainsi, bien qu'il faille utiliser un grand nombre d'états pour réaliser cette opération, la compression de la totalité du mot en entrée peut être réalisée sur le voisinage V avec un nombre fini d'états.

Il nous reste maintenant à décrire la construction dans son ensemble pour montrer qu'il est possible d'obtenir un résultat d'accélération linéaire sur V.

Combinaison des deux phases

Nous avons vu comment il était possible de réaliser la compression de l'entrée sur un voisinage vérifiant les hypothèses de la proposition 4.2.1. Si l'on effectue quelques temps de stockage initialement on peut augmenter l'information que connaît chaque cellule (ce qui ne modifie pas le fonctionnement de la compression, il faut juste déplacer des informations plus importantes) et assurer ainsi que la phase de simulation accélérée se déroulera correctement. Pour éviter de devoir synchroniser toutes les cellules après la compression, il est possible de démarrer la simulation sur toute cellule qui voit que ses voisines sont correctement groupées (comme nous l'avions fait dans le cas du voisinage de Von Neumann dans le chapitre 3).

Si l'automate initial $\mathcal{A}$ fonctionnait en temps $(\mathrm{TR}_V + f)$, l'automate $\mathcal{A}'$ que l'on construit ainsi effectue la compression en temps $\lceil \mathrm{TR}_V/2 \rceil + C$ pour une certaine constante C, puis la simulation accélérée permet de reconnaître le même langage que l'automate initial en temps $\lceil \mathrm{TR}_V/2 + f/2 \rceil$. En combinant les deux étapes on obtient donc un bien un automate cellulaire reconnaissant le même langage que l'automate $\mathcal{A}$ en temps

$$\left\lceil \frac{\mathrm{TR}_V}{2} \right\rceil + C + \left\lceil \frac{\mathrm{TR}_V + f}{2} \right\rceil = \mathrm{TR}_V + \left\lceil \frac{f}{2} \right\rceil + C + 1$$

comme annoncé, ce qui termine la preuve de la proposition 4.2.1.

4.3 Généralisation

Les hypothèses faites sur V dans la proposition 4.2.1 étaient très fortes pour pouvoir obtenir sans trop de difficultés un résultat d'accélération linéaire. Dans cette section nous allons voir quelles hypothèses peuvent être relâchées pour obtenir une classe bien plus vastes de voisinages permettant une accélération linéaire.

4.3.1 Symétrie

Soit V un voisinage complet et convexe en dimension 2 ayant au plus 1 sommet positif et tel que les vecteurs v_d, v_h et v_v définis par rapport à V appartiennent à V. On ne suppose plus ici que V est symétrique.

La compression que nous avons décrite précédemment dans le cas d'un voisinage symétrique semble ne pas pouvoir être réalisée si le voisinage n'est « pas assez symétrique » c'est-à-dire s'il ne contient pas les vecteurs $-v_d$, $-v_h$ et $-v_v$ en particulier car ces vecteurs sont nécessaires pour pouvoir propager la zone B_1 (on rappelle une fois encore que les informations se propagent dans le sens

contraire des vecteurs du voisinage et que pour déplacer une information selon x il faut donc que $-x$ appartienne à V).

Étonnamment pourtant, si le voisinage V contient les vecteurs $-v_d/k$, $-v_h/k$ et $-v_v/k$ pour un certain entier k alors il est possible de modifier un peu la compression pour qu'elle compresse maintenant par un facteur $(k+1)$ au lieu de 2.

On utilisera la proposition très pratique suivante :

Proposition 4.3.1
Soit V un voisinage en dimension d. Si pour un certain entier p le voisinage V^p permet une accélération linéaire, alors le voisinage V permet également une accélération linéaire.

Preuve : La preuve est semblable à la preuve de la proposition 4.1.2. On considère un langage reconnu en temps $(\mathrm{TR}_V + f)$ sur f. L est reconnu en temps $\lceil (\mathrm{TR}_V + f)/p \rceil$ sur V^p ce qui implique qu'il est reconnu en temps $(\mathrm{TR}_{V^p} + f/p + C)$ et donc en temps $(\mathrm{TR}_{V^p} + \lceil f/2p \rceil + C')$ sur V^p. Finalement, L est reconnu en temps $(\mathrm{TR}_V + \lceil f/2 \rceil + C'')$ sur V (et l'entier C'' ne dépend que du voisinage V et de C). □

Il nous faut maintenant montrer que pour un entier k bien choisi le voisinage V^k permet une accélération linéaire. On choisit k suffisamment grand pour que les vecteurs $-v_d$, $-v_h$ et $-v_v$ soient inclus dans V^k (puisque V est complet, il existe un tel k).

Sur ce nouveau voisinage V^k ce sont les vecteurs $k.v_d$, $k.v_h$ et $k.v_v$ qui correspondent aux définitions initiales de v_d, v_h et v_v. Ces trois vecteurs appartiennent bien à V^k, ainsi que $-v_d$, $-v_h$ et $-v_v$ par construction.

On peut alors effectuer la compression telle qu'elle a été décrite précédemment à ceci près que la zone B_1 se déplace maintenant selon v_d à chaque étape et les signaux F se déplacent selon v_h et v_v tandis que les lettres du mot se déplacent selon $-k.v_d$ initialement, puis selon $-k.v_h$ et $-k.v_v$ lorsqu'elles se trouvent sur la frontière de la zone B_1.

Le fait que les lettres se déplacent k fois plus rapidement que le marquage B_1 et les signaux F fait que la lettre partant de c se dirige maintenant vers la cellule $c/(k+1)$ (et non plus $\lfloor c/2 \rfloor$ comme avant).

Cependant le reste de la construction fonctionne de la même manière, et la compression (par un facteur $(k+1)$) se fait ici encore de manière optimale à une constante près. La compression est donc terminée en temps $k.\mathrm{TR}_{V^k}/(k+1) + C$ (au lieu de $\mathrm{TR}_V/2 + C$) où C ne dépend que du voisinage V^k.

Or la nouvelle compression permet d'effectuer la simulation avec un facteur d'accélération $(k+1)$. Ainsi, si l'on pouvait initialement reconnaître un langage L en temps $(\mathrm{TR}_{V^k} + f)$ sur V^k, on peut maintenant reconnaître ce langage en temps

$$\left\lceil \frac{k.\mathrm{TR}_{V^k}}{k+1} \right\rceil + \left\lceil \frac{\mathrm{TR}_{V^k} + f}{k+1} \right\rceil = \mathrm{TR}_{V^k} + \left\lceil \frac{f}{k+1} \right\rceil + C$$

sur V^k.

Le langage V^k permet ainsi une accélération linéaire et c'est donc également le cas de V (proposition 4.3.1). Ainsi, la symétrie n'est pas nécessaire pour obtenir un résultat d'accélération linéaire.

4.3.2 Les vecteurs v_d, v_h, et v_v

Il nous a été très utile de supposer dans la construction de la section 4.2.2 que les vecteurs v_d, v_h, et v_v appartenaient au voisinage V car les déplacements clés se faisaient selon ces directions. Toutefois, cette hypothèse n'est pas véritablement nécessaire.

Considérons en effet un voisinage V ayant au plus 1 sommet positif.

Les vecteurs v_d, v_h et v_v n'appartiennent pas nécessairement à V mais ils s'écrivent tous les trois comme combinaison linéaire à coefficients positifs et rationnels de sommets de V tels que la somme des coefficients est 1 (proposition 1.2.1).

Puisque pour tout $k \in \mathbb{N}$ le voisinage V^k est convexe et que son enveloppe convexe continue est $(\mathrm{CHC}(V))^k$, si l'on choisit k de telle sorte qu'il soit multiple de tous les dénominateurs des coefficients dans les décompositions linéaires de v_d, v_h et v_v en éléments de V, on a

$$\{k.v_d, k.v_h, k.v_v\} \subseteq V^k$$

Cela signifie que les vecteurs $k.v_d$, $k.v_h$ et $k.v_v$ qui correspondent aux définitions de v_d, v_h et v_v pour le voisinage V^k appartiennent tous les trois à V^k. Ainsi V^k permet une accélération linéaire et d'après la proposition 4.3.1 c'est également le cas du voisinage V.

4.3.3 Convexité

L'hypothèse de convexité peut également être omise grâce à la proposition 2.3.1.

En effet nous savons déjà que si un voisinage V est complet, le temps réel sur V et le temps réel sur $\mathrm{CH}(V)$ diffèrent d'au plus une constante et que $\mathrm{TR}_V \geq \mathrm{TR}_{\mathrm{CH}(V)}$.

Soit V un voisinage complet n'ayant pas plus d'un sommet positif. Nous avons déjà montré qu'alors $\mathrm{CH}(V)$ permet une accélération linéaire.

Soit L un langage reconnu en temps $(\mathrm{TR}_V + f)$ sur le voisinage V pour une certaine fonction f. Puisque $V \subseteq \mathrm{CH}(V)$, L est reconnu par un automate cellulaire fonctionnant sur le voisinage $\mathrm{CH}(V)$ en temps $(\mathrm{TR}_V + f)$ et donc en temps $(\mathrm{TR}_{\mathrm{CH}(V)} + f + C)$ pour une certaine constante C.

Puisque $\mathrm{CH}(V)$ permet une accélération linéaire il existe une constante C' telle que le langage L soit reconnu en temps $(\mathrm{TR}_{\mathrm{CH}(V)} + \lceil f/2 \rceil + C')$ par un automate fonctionnant sur $\mathrm{CH}(V)$.

Puisque cette fonction de temps est supérieure au temps réel sur V, la proposition 2.3.1 indique que le langage L est reconnu en temps $(\mathrm{TR}_{\mathrm{CH}(V)} + \lceil f/2 \rceil + C')$ sur V ce qui signifie qu'il est reconnu en temps $(\mathrm{TR}_V + \lceil f/2 \rceil + C')$ sur V. Le voisinage V permet donc également une accélération linéaire.

4.3.4 Complétude

Enfin, l'hypothèse de complétude sur V peut également être affaiblie. On peut ainsi montrer que tout voisinage V dont l'enveloppe convexe continue contient un voisinage de l'origine (c'est-à-dire que V n'est pas inclus dans un demi-plan dont la frontière passe par l'origine) et n'ayant pas plus d'un sommet positif permet une accélération linéaire.

Nous ne prouverons pas ce résultat en détail ici, mais nous nous contenterons d'exposer les principales idées.

Tout d'abord il faut remarquer que l'ensemble des cellules pouvant transmettre une information jusqu'à l'origine est $V^\infty = \bigcup_k V^k$. Cet ensemble est par définition stable par translation selon tout vecteur de V :

$$\forall x \in V, \qquad x + V^\infty \subseteq V^\infty$$

L'hypothèse faite sur V nous assure que V^∞ contient des points dans chaque direction rationnelle. En effet puisque $\mathrm{CHC}(V)$ contient un voisinage de l'origine, pour tout vecteur à coordonnées rationnelles u il existe un vecteur $u' \in \mathrm{CHC}(V) \cap \mathbb{Q}^2$ qui est colinéaire à u et de même sens. Ce vecteur s'écrit comme combinaison linéaire à coefficients rationnels positifs des éléments de V. En multipliant tous les coefficients par le PPCM de leurs dénominateurs on obtient un vecteur $u'' \in V^\infty$ colinéaire à u et de même sens.

En combinant cette propriété avec la stabilité par translation on peut montrer que V^∞ est invariant par translation selon chacun des vecteurs de V :

$$\forall x \in V, \qquad V^\infty = x + V^\infty$$

V^∞ est donc un sous-ensemble périodique de $\mathbb{Z}^2$. Et l'on peut noter $\overline{V}$ sa période :

$$\overline{V} = V^\infty \cap \mathrm{CH}(V)$$

qui sera dans le cas non complet l'équivalent de $\mathrm{CH}(V)$ dans le cas complet. On a en particulier l'identité suivante :

$$V + \overline{V} = \overline{(V^2)}$$

et puisqu'il existe un entier k_0 tel que $\overline{V} \subseteq V^{k_0}$, si un automate cellulaire fonctionnant sur V commence par effectuer k_0 générations de stockage de manière à ce que chaque cellule connaisse les états dans son voisinage $\overline{V}$ il peut par la suite appliquer à chaque génération la règle de transition locale d'un automate cellulaire fonctionnant sur le voisinage $\overline{V}$.

Si les vecteurs v_h, v_v et v_d (définis comme précédemment) appartiennent à $\overline{V}$ on va alors pouvoir appliquer la construction précédente. Les déplacements ne posent pas de problèmes mais on ne peut pas toujours envoyer une lettre de la cellule c vers la cellule $\lfloor c/2 \rfloor$ car cette dernière n'appartient pas nécessairement à $V^\infty(c)$. On effectue alors la compression en envoyant chaque lettre de c vers la cellule de $V^\infty(c)$ la plus proche de $\lfloor c/2 \rfloor$ (cela complique légèrement la construction car il est plus difficile pour les lettres de trouver leur cible après avoir rencontré le signal F, mais on travaille toujours sur un périmètre borné et donc le nombre de cas reste fini et l'on peut résoudre chacune des situations).

Une fois que l'entrée est compressée la simulation accélérée ne pose pas de problème.

Si les trois vecteurs v_h, v_v et v_d n'appartiennent pas à $\overline{V}$ on utilise la méthode vue dans le cas des voisinages complets en considérant une puissance de $\overline{V}$ convenable puis en appliquant la proposition 4.3.1.

Ainsi, si un langage L est reconnu en temps $(\mathrm{TR}_V + f)$ sur V il est également reconnu en temps $(\mathrm{TR}_V + \lceil f/2 \rceil + C)$ et V permet donc une accélération linéaire.

Remarque. On rappelle que dans le cas des voisinages non complets il faut modifier légèrement la définition du temps réel pour qu'elle ait un sens et considérer donc que pour un mot w de taille (n, m) le temps réel est

$$\mathrm{TR}_V(w) = \min\{t \in \mathbb{N} \mid [\![0, n-1]\!] \times [\![0, m-1]\!] \cap V^\infty\}$$

Savoir s'il est nécessaire d'imposer que le voisinage puisse s'étendre dans toutes les directions reste une question ouverte. En effet il n'est pas possible d'effectuer une construction telle que celle que nous avons décrite sur un voisinage ne permettant pas la communication dans les deux sens car il n'est alors pas possible d'envoyer une information de c à $\lfloor c/2 \rfloor$ (il est possible de faire passer l'information par $\lfloor c/2 \rfloor$ mais elle ne peut pas savoir quand elle est arrivée si aucun message ne se déplace en sens inverse). Le fait que l'on puisse obtenir un théorème d'accélération linéaire sur le voisinage *one-way* en dimension 1 (par une méthode assez différente) laisse cependant entrevoir la possibilité d'une généralisation en dimensions supérieures sur des voisinages non complets.

4.3.5 Le théorème général

Les fortes hypothèses de la proposition 4.2.1 nous ont permis de montrer facilement un résultat d'accélération linéaire sur une première famille de voisinages. Puis nous avons vu comment il était possible de se passer de certaines hypothèses une par une. Nous avons donc obtenu le théorème d'accélération linéaire suivant :

Théorème 4.3.1
Soit V un voisinage en dimension 2 tel que $\mathrm{CHC}(V)$ contienne un voisinage de l'origine. S'il n'y a pas plus d'un sommet de V dont les deux coordonnées sont strictement positives (voir figure 4.4) alors il existe une constante C telle que tout langage L reconnu en temps $(\mathrm{TR}_V + f)$ par un automate cellulaire fonctionnant sur le voisinage V pour une certaine fonction $f : \mathbb{N} \to \mathbb{N}$ soit reconnu en temps

$$\mathrm{TR}_V + \left\lceil \frac{f}{2} \right\rceil + C$$

par un automate cellulaire fonctionnant sur le voisinage V.

4.3.6 La dernière hypothèse

Il peut paraître étonnant que de toutes les hypothèses initiales de la proposition 4.2.1 la seule qui n'ait pas pu être éliminée soit celle qui semble la plus extravagante et artificielle. Il est en effet surprenant de devoir imposer que notre voisinage n'ait pas plus d'un sommet dans le quart de plan supérieur droit, d'autant plus que l'on n'ajoute aucune condition sur la partie du voisinage se trouvant en dehors de ce quart de plan.

Pourtant cette hypothèse provient d'un véritable problème lié à des questions d'orientation sur le plan à partir de données locales.

La frontière de l'enveloppe convexe continue d'un voisinage V correspond, comme nous l'avons déjà vu plusieurs fois, à la vitesse maximale à laquelle une information peut (asymptotiquement) se déplacer dans une direction donnée au cours d'un calcul sur un automate cellulaire fonctionnant sur le voisinage V.

Lorsque l'on veut transmettre un message dans une direction donnée (par exemple dans le cas qui nous intéresse, d'une cellule c vers la cellule $\lfloor c/2 \rfloor$), le plus court chemin est bien sûr d'y aller en ligne droite. Toutefois si l'on ne dispose que d'informations locales il n'est pas possible de connaître assez rapidement la direction exacte de la cible et donc de savoir précisément comment se diriger en ligne droite vers cette dernière. Cependant, le fait que l'on travaille sur des polygones convexes fait qu'il est aussi rapide de se déplacer selon des vecteurs appartenant au même segment de la frontière de l'enveloppe convexe de V que le vecteur correspondant au chemin en ligne droite (pourvu que l'on arrive à atteindre la cible en se déplaçant selon de tels vecteurs).

Ainsi, dans les constructions précédentes, il n'était pas gênant de se déplacer selon v_d puis selon v_h ou v_v car les deux vecteurs selon lesquels on se déplaçait étaient sur le même segment que le vecteur optimal (on perdait un temps constant en effectuant des déplacements qui ne correspondaient justement pas à ces vecteurs).

Au début du calcul il n'est pas possible de savoir dans quelle direction se trouve l'origine par rapport à une cellule donnée. On sait seulement que l'origine se trouve « quelque part vers le bas et la gauche ». Lorsqu'il n'y a qu'un seul sommet positif dans le voisinage, on sait qu'il n'y a pas plus de deux segments dans le quart positif de l'enveloppe convexe de V, et le vecteur v_d que l'on a défini appartient alors à ces deux segments. Cela nous permet d'effectuer les premiers déplacements selon un vecteur que l'on sait être sur le même segment que le vecteur optimal. On peut alors envoyer des indications à partir des axes qui atteindront le signal envoyé par la cellule c à temps pour lui indiquer dans quelle continuer pour atteindre sa cible en ne faisant que des déplacements optimaux.

Or dans le cas d'un voisinage ayant plusieurs sommets positifs l'enveloppe convexe de V a trop de segments dans le quart de plan positif et comme il est impossible de déterminer assez rapidement auquel de ces segments appartient le vecteur correspondant au déplacement optimal (en ligne droite) on ne peut pas être sûr d'effectuer des déplacements optimaux lors des premiers pas. Une telle situation est illustrée sur la figure 4.11 où l'on considère que V est le voisinage en octogone représenté sur la figure 4.10.

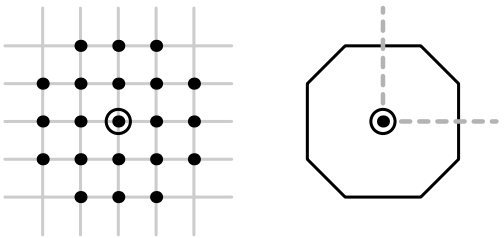

FIGURE 4.10 – Le voisinage en octogone (à gauche) et son enveloppe convexe continue (à droite).

Sur cette figure, on a divisé le plan en trois parties délimitées par des droites en traits pointillés passant par l'origine et dirigées par les sommets positifs de V. Chacune de ces zones correspond à un segment de l'enveloppe convexe de V selon lequel il serait optimal de se déplacer. Ainsi si l'on considère par exemple la cellule c_1 représentée en haut à gauche de la figure, pour se déplacer le plus

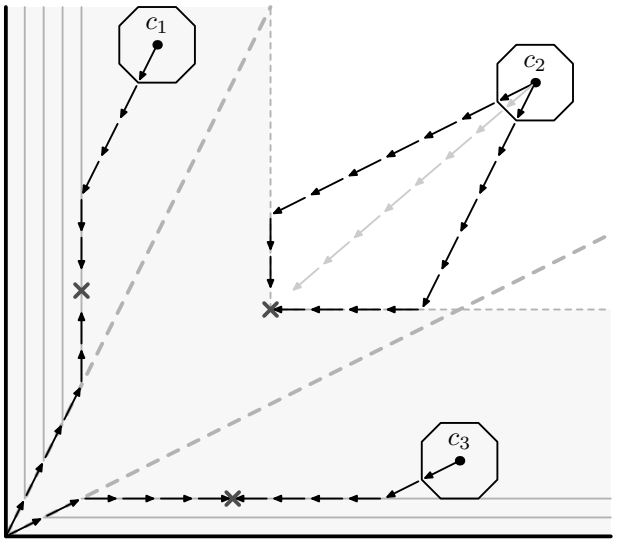

FIGURE 4.11 – Transmission d'information de c à $\lfloor c/2 \rfloor$ dans chacune des trois zones.

rapidement possible vers l'origine il faut effectuer des déplacements correspondant à des vecteurs de V se trouvant sur le segment inférieur de l'enveloppe convexe de V. La cellule c_3 dessinée en bas à droite de la figure se trouve dans la zone symétrique, et les déplacements optimaux sont ceux qui se font selon le côté gauche de $\mathrm{CHC}(V)$. Enfin, la troisième zone est la zone centrale dans laquelle on a représenté la cellule c_2 en haut à droite de la figure. Les déplacements optimaux à partir d'une cellule dans cette zone sont ceux qui se font selon des vecteurs appartenant au côté diagonal en bas à gauche de l'enveloppe convexe de V.

Les deux zones du bord ne posent pas véritablement de problème. En effet, on peut effectuer la construction que l'on a utilisée précédemment pour s'orienter vers $\lfloor c/2 \rfloor$ (représentée par une croix) comme c'est représenté sur la figure : il faut imaginer que les droites en gris parallèles aux axes se déplacent à chaque temps tandis que le signal issu de la cellule c se déplace en diagonale. Lorsque le signal atteint la droite parallèle à l'axe il change de direction pour se déplacer parallèlement à l'axe qu'il a rencontré. On fait de plus circuler des signaux le long des droites parallèles aux axes (les signaux F de la construction précédente) pour indiquer au signal issu de c quand il arrive sur $\lfloor c/2 \rfloor$.

Pour englober ces deux cas il suffit de faire en sorte que chaque cellule envoie des signaux dans les deux directions (cela ne fait que deux fois plus de signaux et c'est donc réalisable sur un automate cellulaire avec un nombre fini d'états même dans le cas où toutes les cellules produisent des signaux en même temps) et seul l'un de ces signaux rencontrera la droite qu'il attend, l'autre rencontrera le mauvais axe et disparaîtra.

Le problème intervient alors dans la zone centrale. Au départ la cellule n'a aucun moyen de savoir dans quelle direction se trouve $\lfloor c/2 \rfloor$ par rapport à sa

position et donc elle ne sait pas dans quelle direction envoyer ses signaux. Par ailleurs seules les cellules au bord du mot en entrée sont capables de lui apporter des informations. Cela signifie que pendant un certain nombre d'étapes (non borné car ce nombre augmente si l'on éloigne c de l'origine) le signal envoyé par la cellule doit se diriger sans aucune information extérieure.

En supposant que le signal envoyé par la cellule c_2 se déplace le plus rapidement possible vers les axes (afin d'obtenir une information sur sa position le plus rapidement possible) on voit que ces informations atteignent le signal alors qu'il a déjà effectué la moitié du parcours vers l'un des axes : on a représenté par des flèches noires les deux trajectoires correspondant à des déplacements se rapprochant le plus possible des axes et la zone grise délimite la position à partir de laquelle le signal envoyé par la cellule peut recevoir une information provenant de l'un des axes. À cet instant il est déjà trop tard pour pouvoir atteindre la cellule cible en temps optimal puisque la fin du trajet (qui peut être arbitrairement longue si l'on éloigne c_2 de l'origine) doit se faire selon un vecteur non optimal (horizontal ou vertical).

Si la cellule c se trouve dans la zone centrale, il n'est donc pas possible, en n'envoyant qu'un nombre fini de signaux, d'être sûr que l'un de ces signaux atteindra la cible en temps optimal (à une constante près). Or on ne peut pas permettre à la cellule d'envoyer un nombre non borné de signaux car cela rendrait la construction impossible à réaliser sur un automate cellulaire ayant un nombre fini d'états si l'on veut que toutes les cellules effectuent la construction en parallèle (cela représenterait un nombre cubique de signaux à placer sur un nombre quadratique de cellules).

La construction que l'on a présentée pour obtenir le théorème d'accélération linéaire ne peut donc pas être réalisée sur un voisinage ayant deux sommets positifs car il n'existe alors pas un unique vecteur qui corresponde toujours à un déplacement optimal vers la cellule cible (le vecteur v_d avait cette propriété dans le cas d'un voisinage ayant au plus un sommet positif) et il n'est donc pas possible pour une cellule d'orienter correctement les signaux qu'elle envoie pour assurer qu'ils atteindront leur cible en un temps minimal à une constante près.

Savoir si des voisinages ayant deux sommets positifs ou plus permettent une accélération linéaire (obtenue par une autre méthode) reste cependant une question ouverte.

Chapitre 5

Séparer le temps réel et l'espace linéaire

In the midst of the word he was trying to say,
In the midst of his laughter and glee,
He had softly and suddenly vanished away –
For the Snark *was* a Boojum, you see.

Lewis Carroll – *The Hunting of the Snark*

Jusqu'ici nous avons beaucoup travaillé sur la classe des langages reconnus en temps réel par des automates en dimension 1. Nous avons montré que cette classe ne dépendait que peu du voisinage choisi et qu'elle était égale à la classe des langages reconnus en temps réel plus une constante. Une importante question ouverte depuis des années consiste à déterminer si cette classe est égale à la classe des langages reconnus en temps linéaire sur des automates cellulaires.

Le fait que l'on ait pas encore réussi à trouver de transformation permettant de passer d'un automate fonctionnant en temps linéaire à un automate fonctionnant en temps réel pourrait porter à croire que ces deux classes sont distinctes, mais la difficulté à trouver un candidat à cette séparation semble indiquer le contraire. En effet, jusqu'à aujourd'hui, tous les langages que l'on arrive à décrire « simplement » reconnus en temps linéaire le sont également en temps réel.

Cependant, notre incompréhension des relations entre les classes de complexité va encore plus loin. À l'instar de la situation dans le monde des machines de Turing, très peu de relations sont connues entre des classes de complexité en temps et des classes de complexité en espace. Dans le cas qui nous intéresse, il n'est aujourd'hui pas possible de savoir si tout langage reconnu en espace linéaire est également reconnu en temps réel ou non.

Pour toute fonction $f : \mathbb{N} \to \mathbb{N}$, si l'on note $\mathrm{CATIME}(f)$ (resp. $\mathrm{CASPACE}(f)$) la classe des langages reconnus par un automate cellulaire fonctionnant sur le voisinage standard en temps f (resp. espace f), on a les inclusions bien connues suivantes

$$\mathrm{CATIME}(\mathrm{Id}) \subseteq \mathrm{CATIME}(2\,\mathrm{Id}) \subseteq \mathrm{CASPACE}(\mathrm{Id})$$

mais l'on ignore si ces inclusions sont strictes ou s'il s'agit en réalité d'égalités.

Dans ce chapitre nous nous intéresserons aux deux extrêmes, et montrerons que si ces deux classes étaient égales cela aurait de très nombreuses conséquences sur d'autres relations entre classes de complexité (y compris dans le cas de la complexité Turing).

Dans ce chapitre nous ne considérerons que des automates cellulaires en dimension 1 fonctionnant sur le voisinage standard.

5.1 Définitions et propriétés

Définition 5.1.1 (Fonctions t-constructibles)
On dira qu'une fonction $f : \mathbb{N} \to \mathbb{N}$ est t-constructible s'il existe un automate cellulaire $\mathcal{A}$ d'états $\mathcal{Q}$, deux états q_1 et q_f de $\mathcal{Q}$, tels que pour tout entier $n \in \mathbb{N}$, partant de la configuration correspondant au mot q_1^n (l'encodage unaire de l'entier n) au temps $\tau = 0$, l'origine de l'automate $\mathcal{A}$ soit dans l'état q_f pour la première fois au temps $f(n)$ (voir [11]).

La définition précédente signifie simplement que sur l'entrée n (donné sous forme unaire), l'automate $\mathcal{A}$ est capable de « compter » $f(n)$ générations puis de marquer l'origine.

Définition 5.1.2 (Fonctions s-constructibles)
On dira qu'une fonction $f : \mathbb{N} \to \mathbb{N}$ est s-constructible s'il existe un automate cellulaire $\mathcal{A}$ d'états $\mathcal{Q}$, deux états q_1 et q_f de $\mathcal{Q}$, tels que pour tout entier $n \in \mathbb{N}$, partant de la configuration correspondant au mot q_1^n, après un certain temps toutes les cellules sur le segment $[\![0, f(n) - 1]\!]$ passent dans l'état q_f, qu'aucune cellule n'a jamais été dans l'état q_f auparavant, et que le calcul n'a eu lieu que sur ces cellules (toute cellule hors de ce segment est dans l'état B à chaque génération).

Cette définition ici signifie que l'automate est capable, sur l'entrée n en unaire, de calculer $f(n)$ en unaire sans utiliser plus de cellules que celles qui sont nécessaires pour écrire le résultat.

On a la propriété suivante :

Proposition 5.1.1 (Firing squad)
Il est possible de synchroniser un segment de k cellules adjacentes (leur faire prendre un même état pour la première fois au même instant). La synchronisation peut être réalisée en temps $ak + b$ pour tout $a \geq 2$ et $b \in \mathbb{N}$, à partir de l'instant où la cellule de gauche émet le signal de synchronisation. Le bord droit du segment n'a besoin d'être marqué qu'à partir du temps $(a - 1)k$.

Ce problème de synchronisation a été très étudié et de nombreuses solutions particulièrement ingénieuses ont été trouvées. Dans ce chapitre, nous n'aurons besoin que d'une solution capable de retarder la synchronisation pour qu'elle ait lieu au temps ak pour n'importe quel $k \geq 2$, et telle que le bord droit du segment puisse être marqué le plus tard possible (donc au temps exactement $(a - 1)k$). Une telle solution est un cas particulier des solutions « avec délai » exposées dans [12].

La proposition 5.1.1 nous permet donc de justifier que la contrainte de synchronisation dans la définition des fonctions s-constructibles n'est pas restrictive

(il suffit donc qu'un automate soit capable de marquer la cellule $f(n-1)$ pour que la fonction soit s-constructible).

On a les résultats suivants :

Proposition 5.1.2
Soit $f : \mathbb{N} \to \mathbb{N}$ une fonction t-constructible. Si pour tout $x \in \mathbb{N}$ on a $f(x) \geq 3x$ alors la fonction $x \mapsto f(x) - x$ est également t-constructible.

Preuve : On utilise la construction illustrée sur la figure 5.1. Sur cette figure, tous les signaux en trait plein se déplacent à vitesse 1 et les signaux en pointillés se déplacent à vitesse 1/2.

Dans un premier temps, un signal part de l'origine vers la droite à vitesse 1/2 tandis qu'un autre part du bord droit du mot (donc de la cellule n) vers la gauche à vitesse 1. Lorsque ces deux signaux se rencontrent (sur la cellule $n/3$) on lance un signal de synchronisation pour que la totalité du segment $[\![0, n/3]\!]$ soit synchronisé au temps n.

Au temps n on est donc dans une situation qui correspond à l'entrée en unaire de l'entier $n/3$. On peut alors considérer cette configuration comme une représentation de n en unaire groupée par un facteur 3. Sur cette configuration on peut alors simuler le fonctionnement de l'automate $\mathcal{A}_f$ qui calcule la fonction f (puisqu'on a supposé que f était t-constructible) avec un facteur d'accélération 3 (on maintient le groupage par 3 et l'on calcule 3 générations de $\mathcal{A}_f$ en une seule étape). Cette construction dure donc $f(n)/3$ étapes et va nous permettre de marquer l'origine au temps $(n + f(n)/3)$. Un signal s_1 part alors de l'origine vers la droite à vitesse 1.

Pendant ce temps, en parallèle, l'automate marque l'origine au temps $2n$ en envoyant un signal de l'origine au bord droit du mot puis de retour à l'origine. Au temps $2n$ l'origine émet alors un signal s_2 se déplaçant à vitesse 1/2 vers la droite.

Si le signal s_1 est émis après le signal s_2, le premier rattrape le second sur la cellule $(f(n)/3 - n)$. Un nouveau (et dernier) signal est alors émis qui se déplace vers la gauche à vitesse maximale. Ce dernier signal atteint alors l'origine $(f(n)/3 - n)$ étapes après avoir été émis, soit $2(f(n)/3 - n)$ étapes après l'apparition du signal s_1, il peut donc marquer l'origine au temps $(n + f(n)/3) + 2(f(n)/3 - n) = f(n) - n$.

Remarquons enfin que cette construction ne fonctionne que si le signal s_2 est émis avant le signal s_1. Ceci se produit si $2n \leq (f(n)/3 + n)$ soit $3n \leq f(n)$. □

Proposition 5.1.3
Soit $f : \mathbb{N} \to \mathbb{N}$ une fonction t-constructible et $k \geq 2$ un entier. Si pour tout $x \in \mathbb{N}$ on a $f(x) \geq 2kx$ alors la fonction $x \mapsto f(x)/k$ est également t-constructible.

Preuve : On utilise ici la construction illustrée sur la figure 5.2. On utilise un signal se déplaçant à vitesse $1/(k-1)$ pour marquer le point n/k au temps $n(k-1)/k$. On synchronise alors le segment $[\![0, n/k]\!]$ au temps n puis on peut démarrer le calcul de $f(n)$ groupé d'un facteur k. On marque ainsi l'origine au temps $f(n)/k + n$. La fonction $n \mapsto f(n)/k + n$ est donc t-constructible.

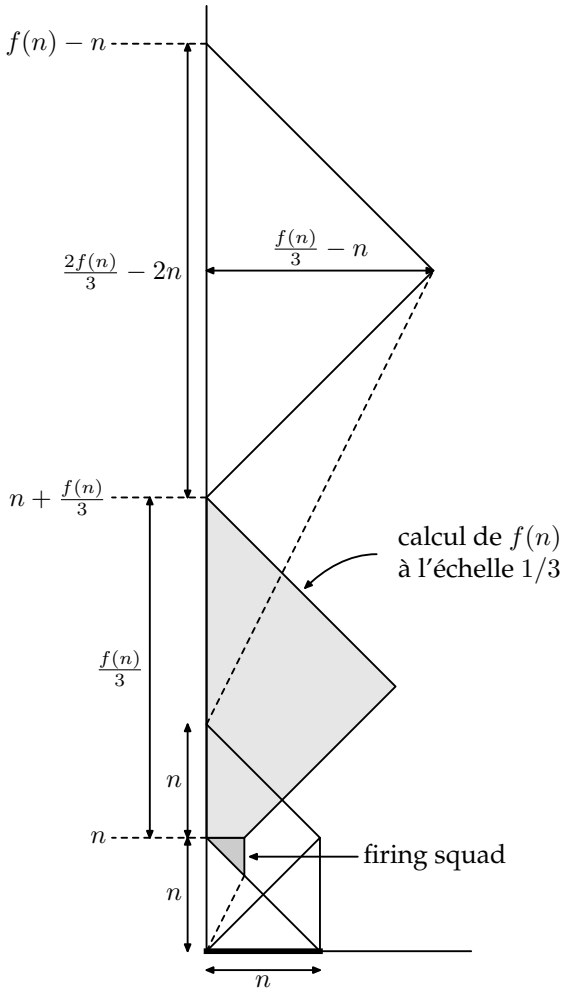

FIGURE 5.1 – Construction de la fonction $n \mapsto f(n) - n$.

Si $f(n) \geq 2kn$ alors $f(n)/k + n \geq 3n$ et donc la fonction $n \mapsto f(n)/k$ est également t-constructible par la proposition 5.1.2. □

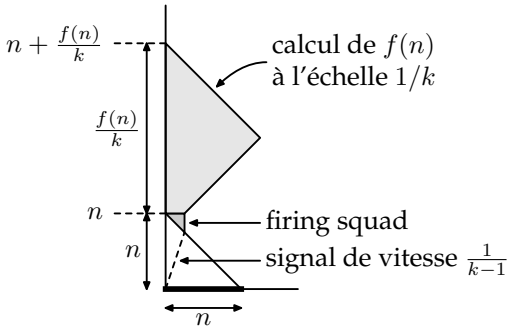

FIGURE 5.2 – Construction de la fonction $n \mapsto f(n)/k + n$.

Proposition 5.1.4
Toute fonction t-constructible f telle que $\forall x, f(x) \geq x$ est s-constructible.

Preuve : Considérons une fonction f t-constructible vérifiant $\forall x, f(x) \geq x$. Soit $\mathcal{A}$ un automate qui, sur l'entrée x en unaire marque l'origine d'un état q_f pour la première fois au temps $f(x)$. Nous allons décrire la construction d'un automate qui, sur l'entrée x en unaire, fait passer l'ensemble des cellules $[\![0, f(x) - 1]\!]$ en même temps dans un état q'_f pour la première fois et tel qu'aucune autre cellule ne participe au calcul.

Initialement, le diagramme espace-temps de $\mathcal{A}$ est de la forme illustrée sur partie en haut à gauche de la figure 5.3 où toutes les cellules en dehors de la zone grise sont dans l'état B. On « replie » alors le diagramme espace-temps de $\mathcal{A}$ au niveau de la colonne $(x - 1)$ (la dernière lettre du mot en entrée) de telle sorte qu'aucune cellule à droite du mot en entrée ne participe au calcul. C'est la transformation que l'on a déjà effectuée de nombreuses fois pour limiter l'espace de calcul à une demi-droite mais cette fois-ci la transformation est effectuée à droite du mot en non à gauche comme on a l'habitude de le faire. Le diagramme espace-temps du nouvel automate que l'on obtient après cette transformation est alors illustré sur la partie en haut à droite de la figure 5.3. On modifie alors notre automate en groupant deux états par cellule de telle sorte que le calcul utilise deux fois moins de cellules sur la partie négative. En effet, si l'on peut mettre deux états de l'automate initial par cellule du nouvel automate le même calcul peut avoir lieu sur un espace plus petit. Pour simplifier le diagramme et la construction, on n'applique pas la compression sur les cellules qui contenaient les lettres du mot initialement. Le diagramme espace-temps de ce nouvel automate est alors représenté sur la partie en bas à gauche de la figure 5.3. Il ne reste plus qu'à effectuer un dernier pliage, au niveau de la colonne 0 pour ramener le calcul sur les cellules positives. On obtient finalement un diagramme espace-temps comme celui qui est représenté sur la partie inférieure droite de la figure 5.3.

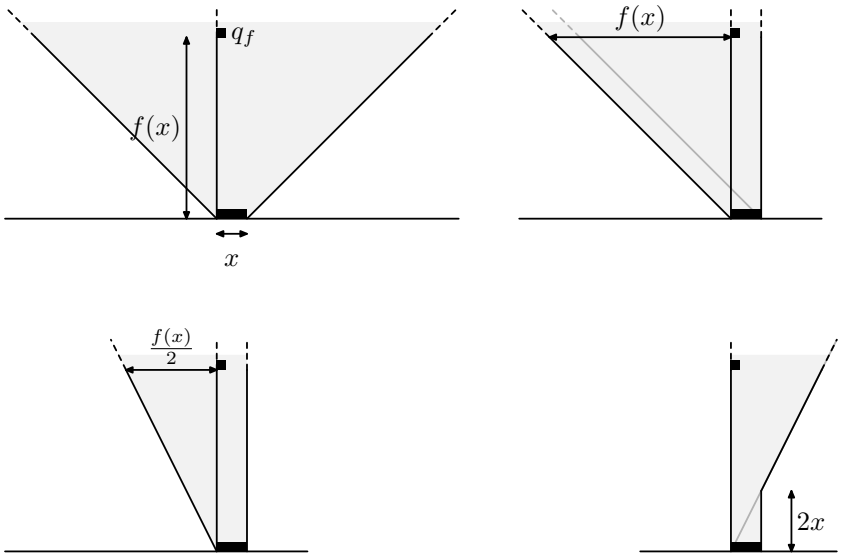

FIGURE 5.3 – Transformation du diagramme espace-temps de $\mathcal{A}$ pour prouver la proposition 5.1.4[2].

Toutes les transformations que l'on a décrites n'affectent pas le fonctionnement de l'origine ce qui signifie qu'elle prend bien l'état q_f pour la première fois au temps $f(x)$.

La fin de la construction est illustrée par la figure 5.4 : au temps 0 l'origine émet un signal qui se déplace vers la droite à vitesse 1/2. Plus tard, lorsqu'elle entre dans l'état q_f (donc au temps $f(x)$), elle émet un second signal qui se déplace vers la droite à vitesse maximale et amorce une synchronisation selon l'algorithme du firing squad (proposition 5.1.1). Si $f(x) \geq x$ les deux signaux se rencontrent sur la cellule $(f(x) - 1)$ au temps $2f(x)$ et cette cellule est alors marquée comme étant la dernière du segment à synchroniser. À cet instant, toutes les cellules au-delà de $(f(x) - 1)$ ont toujours été dans l'état B. La synchronisation peut alors être effectuée sans faire intervenir d'autres cellules (la synchronisation est représentée par une surface hachurée sur la figure 5.4) et la totalité du segment $[\![0, f(x) - 1]\!]$ peut alors passer pour la première fois dans l'état q'_f.

La fonction f est donc bien s-calculable. □

5.2 Théorème de transfert général

Nous sommes maintenant en mesure de montrer le résultat suivant :

Théorème 5.2.1 (Transfert)
Étant données deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$ s-constructibles, si l'on a $\text{CASPACE}(f) \subseteq \text{CATIME}(g)$ *alors pour toute fonction t-constructible $h : \mathbb{N} \to$*

[2]La fin du pliage permettant de transformer la totalité du diagramme espace-temps en cocotte ou en cygne est laissée au lecteur en guise d'exercice.

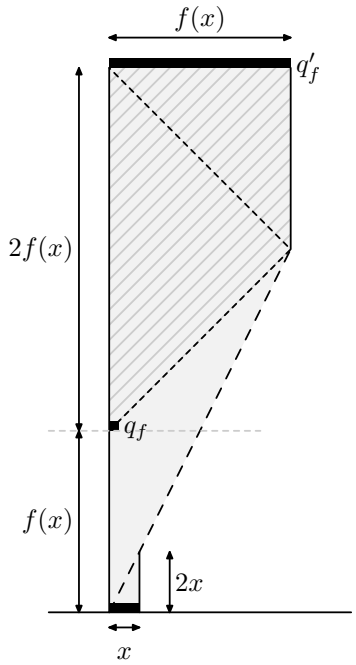

FIGURE 5.4 – L'automate permettant de prouver la proposition 5.1.4

$\mathbb{N}$ *telle que* $\forall x \in \mathbb{N}, h(x) \geq 6x$ *on a*

$$\text{CASPACE}(f \circ h) \subseteq \text{CATIME}(g \circ h)$$

Pour montrer ce théorème, considérons trois fonctions f, g et h vérifiant les hypothèses annoncées et supposons que l'on ait

$$\text{CASPACE}(f) \subseteq \text{CATIME}(g)$$

On se donne par ailleurs un langage L sur un alphabet Σ reconnu en espace $f \circ h$. Montrons alors que L est reconnu en temps $g \circ h$.

5.2.1 Le langage $\widetilde{L}$

Lemme 5.2.1
Le langage $\widetilde{L} = \{w\#^{h(|w|)-|w|} \mid w \in L\}$, *où* $\#$ *est un nouveau symbole n'appartenant pas à* Σ, *est reconnu en temps* g.

Preuve : Montrons tout d'abord que $\widetilde{L}$ est reconnu en espace f en construisant un automate cellulaire. Soit ω un mot en entrée, on cherche à déterminer si ω est dans $\widetilde{L}$.

Si ω n'est pas dans $\Sigma^*\{\#\}^*$, il est très facile de le détecter et de refuser l'entrée. Supposons donc que $\omega = w\#^l$ avec $w \in \Sigma^*$. Puisque f est s-constructible on peut commencer par marquer la cellule $f(|\omega|) - 1$, ce qui nous permet de borner l'espace de calcul de notre automate (on impose que le calcul ne dépasse jamais cette limite).

Ensuite l'automate vérifie que w appartient bien à L. Cela peut se faire en espace $f \circ h(|w|)$ par hypothèse sur L. Si le mot en entrée est dans $\widetilde{L}$ on a donc assez de place pour reconnaître si w appartient à L puisque $f(h(|w|)) = f(|\omega|)$. Si au cours de ce calcul l'automate n'a pas assez de place c'est qu'il n'y avait pas assez de symboles # dans le mot ω et le mot est donc refusé.

Par ailleurs, si w est bien dans L, l'automate doit vérifier qu'il y a exactement le bon nombre de symboles #. Ceci se fait facilement puisque h est s-constructible. On marque donc la cellule $h(|w|) - 1$ sans utiliser de cellules au-delà. Ici encore, si le calcul dépasse de la limite que l'on s'était fixée $(f(|\omega|) - 1)$ c'est que le mot ω n'appartient pas à $\widetilde{L}$. Si par contre le calcul se fait correctement on peut alors vérifier que la dernière lettre du mot ω est exactement sur la cellule $h(|w|) - 1$ ce qui permet de déterminer si ω appartient ou non à $\widetilde{L}$.

Le langage $\widetilde{L}$ est donc reconnu en espace f. Puisque l'on a supposé que $\mathrm{CASPACE}(f) \subseteq \mathrm{CATIME}(g)$, $\widetilde{L}$ est également reconnu en temps g. □

On suppose à partir de maintenant que l'on a un automate cellulaire $\mathcal{A}$ reconnaissant $\widetilde{L}$ en temps g. On va alors construire un automate cellulaire $\mathcal{A}'$ capable de reconnaître L en temps $g \circ h$.

5.2.2 Compression du diagramme espace-temps

Le but de cette section est d'expliquer comment on peut construire l'automate cellulaire $\mathcal{A}'$ qui, sur l'entrée w, simule le fonctionnement de $\mathcal{A}$ sur l'entrée $w\#^{h(|w|)-|w|}$, à vitesse réelle. La configuration initiale de $\mathcal{A}$ est très simple, et la seule information qui manque à $\mathcal{A}'$ est la position exacte de la cellule $h(|w|)-1$.

Observons un diagramme espace-temps typique de l'automate $\mathcal{A}$ sur une entrée $\omega = w\#^{h(|w|)-|w|}$ (figure 5.5). Si l'on utilise la technique de restriction de l'espace on peut considérer qu'aucun calcul n'a lieu à gauche de l'origine. On distingue alors la bande verticale se trouvant immédiatement au dessus du mot w et le reste du diagramme, à partir de la cellule $|w|$ (ces deux zones sont séparées par une ligne pointillée sur la figure). Si l'on suppose que $\mathcal{A}'$ arrive à calculer correctement tous les états sur la colonne se trouvant juste à droite de la ligne pointillée (les états successifs de la cellule $|w|$), $\mathcal{A}'$ est alors capable de calculer correctement tous les états se trouvant dans la bande verticale correspondant au mot w puisque le mot w est sur la configuration initiale de $\mathcal{A}'$ et que les états sur cette bande ne dépendent que des états précédemment sur la bande et de ceux présents sur la colonne $|w|$.

Pour calculer les états sur la colonne $|w|$, nous allons effectuer une transformation géométrique de la partie droite (à droite de la ligne pointillée) du diagramme espace-temps qui conserve cette colonne, et permette à l'automate $\mathcal{A}'$ de construire la portion de la configuration initiale de $\mathcal{A}$ qu'il ne connaît pas encore : le segment de symboles # de longueur $h(|w|) - |w|$.

Définition de la compression

Décalons le système de coordonnées sur le diagramme de telle sorte que l'origine soit maintenant la première cellule à droite du mot w, c'est-à-dire que la cellule

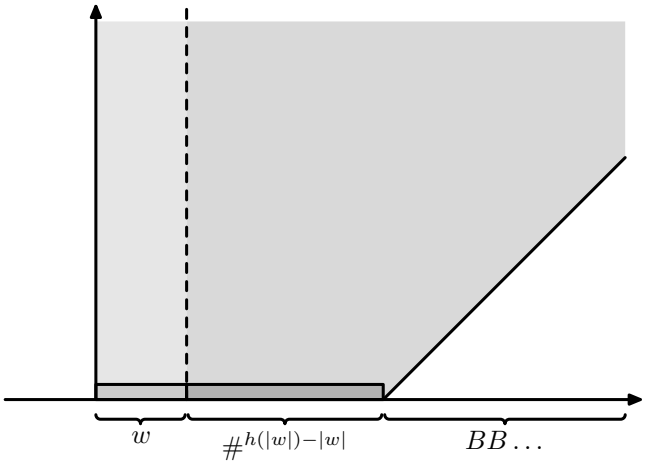

FIGURE 5.5 – Diagramme espace-temps de l'automate $\mathcal{A}$.

c devient maintenant la cellule $(c - |w|)$ (voir figure 5.6). Considérons alors la transformation

$$\sigma : \begin{cases} \mathbb{N}^2 & \to & (\frac{1}{3}\mathbb{N})^2 \\ (x, y) & \mapsto & (\frac{x}{3}, y + \frac{2x}{3}) \end{cases}$$

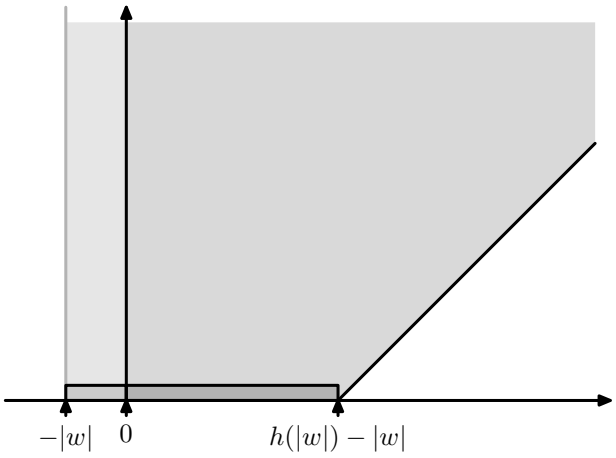

FIGURE 5.6 – Changement de coordonnées.

L'axe vertical est invariant par σ et l'image de l'axe horizontal (la configuration initiale) est un segment de pente 2. La figure 5.7 illustre en détail l'effet de la transformation σ sur les états de la partie droite du diagramme espace-temps de $\mathcal{A}$. On voit que chaque cellule après compression reçoit 3 états du diagramme avant compression (exceptée l'origine qui conserve un unique état). On a également représenté sur la figure le voisinage d'une cellule et son successeur au temps suivant avant et après la transformation.

Montrons alors que l'automate $\mathcal{A}'$ sur l'entrée correspondant au mot w est capable de simuler le fonctionnement de $\mathcal{A}$ sur l'entrée ω, en construisant l'image

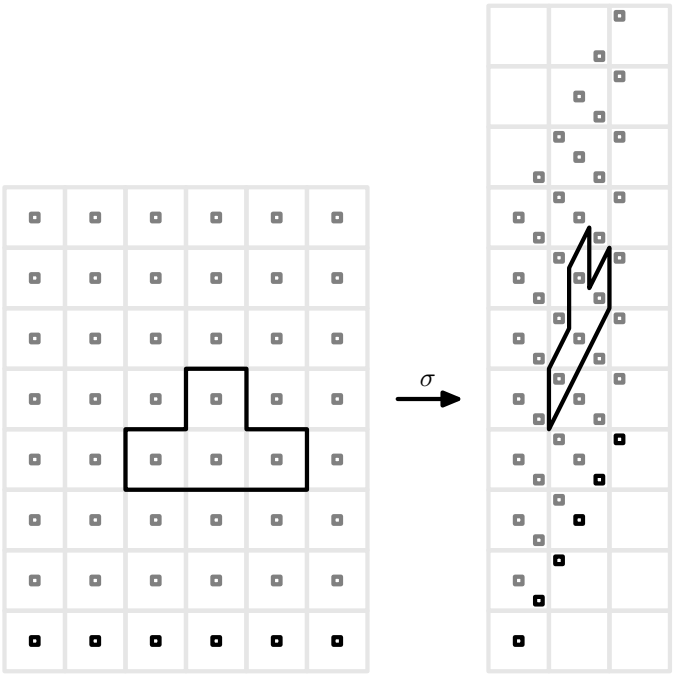

FIGURE 5.7 – Les effets de la compression σ sur la partie droite du diagramme espace-temps.

par σ de la partie droite du diagramme espace-temps de $\mathcal{A}$.

Fonctionnement de la simulation

Puisque l'on a supposé que pour tout $x \in \mathbb{N}$, $h(x) \geq 6x$, la proposition 5.1.3 assure que $h/3$ est t-constructible ce qui signifie que l'origine (toujours dans notre repère translaté) peut être marquée au temps $h(|w|)/3$, ce qui nous permet de marquer la cellule $h(|w|)/3$ au temps $2h(|w|)/3$ en envoyant des signaux comme indiqué sur la figure 5.8 (le trait en pointillés noirs est un signal se déplaçant à vitesse 1 tandis que celui en pointillés gris est un signal se déplaçant à vitesse 1/2). Le point de l'espace-temps $(h(|w|)/3, 2h(|w|)/3)$ est exactement l'image par σ du point $(h(|w|), 0)$ qui est la dernière cellule dans l'état # de la configuration initiale de $\mathcal{A}$.

Ainsi, même si $\mathcal{A}'$ n'a pas l'information complète de ω sur sa configuration initiale, il est capable de reconstituer l'image de cette configuration initiale par σ. En effet, en déplaçant un signal vers la droite à vitesse 1/2 à partir de la nouvelle origine, $\mathcal{A}'$ peut écrire des états # sur le segment qui est l'image du segment de symboles # dans la configuration initiale de $\mathcal{A}$ et l'on vient de voir que $\mathcal{A}'$ est capable de déterminer le point exact où ce segment doit se terminer.

Il reste maintenant à vérifier que l'automate $\mathcal{A}'$ est capable d'appliquer la fonction de transition locale de $\mathcal{A}$ aux informations qui se trouvent sur la partie du diagramme espace-temps transformée selon σ.

Les états du diagramme espace temps de $\mathcal{A}$ peuvent se trouver dans 3 situations différentes après compression par σ : en haut à gauche d'une cellule,

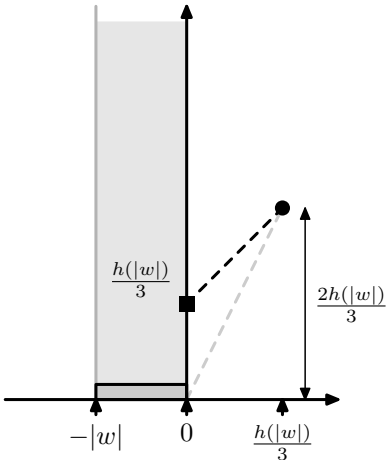

FIGURE 5.8 – Construction du point $(h(|w|)/3, 2h(|w|)/3)$ au temps $2h(|w|)/3$.

au centre d'une cellule ou en bas à droite d'une cellule. La figure 5.9 illustre la situation pour chacune de ces trois positions.

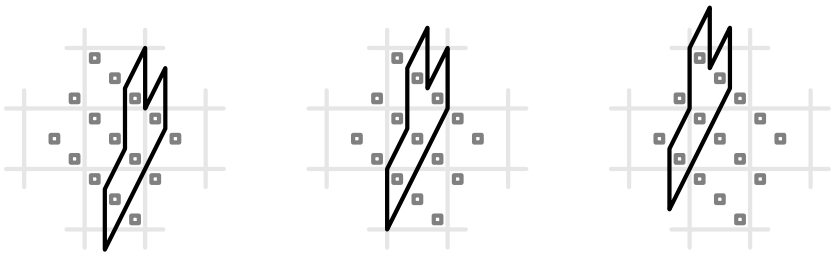

FIGURE 5.9 – Les trois cas de figure possibles lorsque l'on cherche à appliquer la règle de transition de $\mathcal{A}$ après compression du diagramme espace-temps.

Sur la partie gauche de la figure, on s'intéresse à un état se trouvant en bas à droite d'une cellule. La cellule que l'on considère est celle qui se trouve au centre de la figure. On a représenté par un contour noir le voisinage de l'état auquel on s'intéresse ainsi que la position (dans l'espace-temps) où se trouve le successeur de cet état (comme nous l'avions déjà illustré sur la figure 5.7). On voit que l'état successeur se trouvera sur la même cellule au temps suivant et que tous les états que la cellule doit connaître pour appliquer la règle de transition de $\mathcal{A}$ sont sur le voisinage de la cellule que l'on considère : l'état central est sur la cellule elle-même, l'état de droite se trouve sur la voisine de droite tandis que l'état de gauche se trouvait sur la cellule elle-même au temps précédent et l'on peut supposer que les cellules ont une mémoire suffisante pour se souvenir des informations qu'elles contenaient une génération plus tôt. Dans ce cas-ci, la cellule est donc capable d'appliquer la règle de transition de $\mathcal{A}$ à l'information qu'elle porte.

Sur la partie centrale de la figure 5.9, on s'intéresse au cas d'un état se trouvant au centre d'une cellule. La situation est quasiment identique à la précédente à ceci près que l'un des états que la cellule doit connaître (celui de droite) se

trouve dans le futur de la cellule : on sait qu'il se trouvera sur la cellule au temps suivant. Cependant, cet état est un état en bas à droite de la cellule, et l'on a vu que la cellule était bien capable de calculer cet état avec les informations qu'elle voyait. Elle peut donc calculer cet état en bas à droite comme précédemment, puis calculer le successeur de l'état qu'elle porte au centre le tout en une seule génération.

Enfin la partie droite illustre le dernier cas. Ici on voit que la cellule doit d'abord calculer les successeurs des deux autres états qu'elle porte comme décrit précédemment pour être ensuite capable de calculer le successeur de l'état en haut à gauche.

Dans les trois cas, nous avons vu que si le diagramme de $\mathcal{A}'$ contient l'image par σ de la configuration de $\mathcal{A}$ au temps t alors $\mathcal{A}'$ est capable de calculer l'image de la configuration de $\mathcal{A}$ au temps $(t+1)$. Nous avions déjà vu qu'il était possible de reconstituer totalement la configuration initiale de $\mathcal{A}$, ce qui permet donc à $\mathcal{A}'$ de simuler totalement la partie droite du diagramme espace-temps de $\mathcal{A}$ (à droite de la dernière lettre de w dans la configuration initiale).

5.2.3 Recollement des simulations et fin du calcul

Finalement, on a vu que $\mathcal{A}'$ pouvait simuler la bande verticale du diagramme espace-temps de $\mathcal{A}$ se trouvant au dessus de la partie de l'entrée correspondant aux lettres du mot w sans déformation, et la partie droite du diagramme après avoir appliqué une transformation σ, qui ne transforme cependant pas la colonne qui permet de joindre les deux parties. L'automate $\mathcal{A}'$ sur l'entrée w est donc capable de simuler entièrement le fonctionnement de $\mathcal{A}$ sur l'entrée ω, ce qui signifie qu'au temps $g(|\omega|) = g \circ h(|w|)$ il est capable de déterminer si ω appartient ou non à $\widetilde{L}$ et donc si w appartient ou non à L. Finalement, le langage L est bien reconnu en temps $g \circ h$, ce qui achève la preuve du théorème 5.2.1.

5.3 Théorème de transfert temps réel

Si l'on suppose que tout langage reconnu en espace linéaire par un automate cellulaire est également reconnu en temps réel, on a une version légèrement plus forte du théorème précédent :

Théorème 5.3.1 (Transfert temps réel)
Si l'on a CASPACE(Id) $\subseteq$ CATIME(Id) *alors pour toute fonction s-constructible* $h : \mathbb{N} \to \mathbb{N}$ *on a*

$$\text{CASPACE}(h) \subseteq \text{CATIME}(h)$$

Ce théorème est quasiment un cas particulier du théorème 5.2.1 en prenant $f = g = \text{Id}$, à ceci près que l'on ne suppose pas ici que h est t-constructible (uniquement s-constructible) ni que pour tout $x \in \mathbb{N}$ on a $h(x) \geq 6x$. Voyons pourquoi ces deux hypothèses ne sont pas nécessaires ici.

Lemme 5.3.1
Si CASPACE(Id) $\subseteq$ CATIME(Id) *alors toute fonction s-constructible est t-constructible.*

Preuve : Soit f une fonction s-constructible. Considérons le langage

$$L_f = \{1^x\#^{f(x)-x} \mid x \in \mathbb{N}\}$$

Étant donné un mot w, si le mot est bien de la forme $1^a\#^b$, l'automate essaie de marquer la cellule $f(a)-1$. Par hypothèse sur f ce calcul peut être réalisé en ne faisant intervenir que $f(a)$ cellules. Si au cours du calcul l'automate sort du segment délimité par le mot en entrée c'est que b est trop petit. Si par contre le calcul se déroule correctement, l'automate peut comparer $f(a)-1$ à la taille du mot en entrée et conclure sur l'appartenance ou non de w à L_f. Le langage L_f est donc reconnu en espace Id.

Par hypothèse, L_f est donc reconnu en temps réel. Cela signifie qu'il existe un automate cellulaire $\mathcal{A}$ tel que, sur l'entrée $1^x\#^y$, l'origine est dans un état acceptant au temps $(x+y-1)$ si et seulement si $x+y=f(x)$. En particulier, sur l'entrée $1^x\#^\infty$ (la configuration initiale contient un segment initial de 1 puis une infinité de # sur la droite), l'origine n'est dans un état acceptant après le temps $x-1$ qu'au temps $t=f(x)-1$ puisqu'être dans un état acceptant à un temps t signifie accepter le préfixe de taille $(t+1)$ de l'entrée (on rappelle qu'au temps t, seules les $(t+1)$ premières lettres du mot peuvent avoir une influence sur l'état de l'origine), soit donc le mot $1^x\#^{t-x+1}$ pour $t \geq x-1$.

Ainsi, on peut construire un automate qui, sur l'entrée 1^x simule le fonctionnement de $\mathcal{A}$ sur l'entrée $1^x\#^\infty$ et marque l'origine au temps $f(x)$. La fonction f est donc bien t-constructible. □

Sous l'hypothèse CASPACE(Id) $\subseteq$ CATIME(Id), la s-constructibilité implique donc la t-constructibilité. En ce qui concerne la contrainte $\forall x \in \mathbb{N}, h(x) \geq 6x$, rappelons tout d'abord que CASPACE($x \mapsto 6x$) = CASPACE(Id). Ainsi, d'après notre hypothèse initiale, CASPACE($x \mapsto 6x$) = CATIME(Id).

Soit h une fonction s-constructible, L un langage reconnu en espace h et $\mathcal{A}$ un automate cellulaire reconnaissant L en espace h. On définit le langage

$$L_< = \{w \in L \mid \text{ la reconnaissance de } w \text{ par } \mathcal{A} \text{ utilise moins de } 6|w| \text{ cellules}\}$$

Par définition de la complexité en espace, pour tout mot w tel que $h(|w|) \leq 6|w|$ on a $w \in L_< \Leftrightarrow w \in L$. De plus $L_<$ est reconnu en espace $x \mapsto 6x$ et donc, d'après notre hypothèse, en temps réel.

L'automate cellulaire que l'on a construit pour la preuve du théorème 5.2.1 permettait de décider l'appartenance ou non d'un mot w au langage L en temps $h(|w|)$ pour tout mot tel que $h(|w|) \geq 6|w|$. Par ailleurs nous venons de montrer l'existence d'un automate qui accepte tous les mots w de L tels que $h(|w|) \leq 6|w|$, et n'accepte que des mots de L.

Il nous suffit enfin de considérer un automate produit qui effectue en parallèle le travail de ces deux automates. L'automate produit accepte tout mot qu'au moins l'un des deux automates accepte. Cet automate reconnaît alors le langage L en temps h, ce qui prouve le théorème 5.3.1.

5.4 Conséquences en complexité Turing

Nous étudierons ici les conséquences des théorèmes 5.2.1 et 5.3.1 sur les classes de complexité au sens Turing.

Dans toute cette section, pour toute fonction $f : \mathbb{N} \to \mathbb{N}$ on note DTIME(f) (resp. DSPACE(f)) la classe des langages reconnus par une machine de Turing déterministe à un ruban et une tête en temps f (resp. espace f).

On utilisera dans cette section le théorèmes suivants :

Théorème 5.4.1 (Paterson[14])
Pour toute fonction $f : \mathbb{N} \to \mathbb{N}$ telle que $\forall x, f(x) \geq x$ on a

$$\mathrm{DTIME}(f^2) \subseteq \mathrm{DSPACE}(f)$$

Remarque. Dans toute cette section la notation f^2 désigne la fonction $(x \mapsto (f(x))^2)$ et non pas $(x \mapsto f \circ f(x))$.

On rappelle les identités suivantes :

Proposition 5.4.1
Pour toute fonction $f : \mathbb{N} \to \mathbb{N}$ vérifiant $\forall x \in \mathbb{N}, f(x) \geq x$, toute fonction $g : \mathbb{N} \to \mathbb{N}$ vérifiant $\forall x \in \mathbb{N}, g(x)/\log(g(x)) \geq x$ et tout entier $k \geq 1$, on a

$$\begin{array}{rcccccl} & & \mathrm{CASPACE}(f) & = & \mathrm{DSPACE}(f) & & \\ \mathrm{DTIME}(f) & \subseteq & \mathrm{CATIME}(f) & \subseteq & \mathrm{DTIME}(f^2) & \subseteq & \mathrm{DSPACE}(f) \\ & & \mathrm{CASPACE}(k.f) & = & \mathrm{CASPACE}(f) & & \\ & & \mathrm{CATIME}(k.f) & = & \mathrm{CATIME}(2.f) & & \end{array}$$

Bien que nous ne soyons toujours pas en mesure de prouver que la classe des langages reconnus en temps linéaire par un automate cellulaire en dimension 1 est strictement incluse dans la classe des langages reconnus en espace linéaire, nous pouvons relier cette hypothèse à d'autres questions bien connues de complexité.

Proposition 5.4.2
Si $\mathrm{CASPACE}(\mathrm{Id}) \subseteq \mathrm{CATIME}(2\,\mathrm{Id})$ *alors*

$$\mathrm{P} = \mathrm{PSPACE}$$

où $\mathrm{P} = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(x \mapsto x^k)$ *et* $\mathrm{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}(x \mapsto x^k)$ *désignent les classes de complexité bien connues.*

Preuve : Pour tout $k \in \mathbb{N}^*$ la fonction $(x \mapsto x^k)$ est t-constructible. Si $\mathrm{CASPACE}(\mathrm{Id}) \subseteq \mathrm{CATIME}(2\,\mathrm{Id})$ le théorème 5.2.1 indique que pour tout k on a

$$\mathrm{CASPACE}(x \mapsto x^k) \subseteq \mathrm{CATIME}(x \mapsto 2x^k)$$

et donc

$$\mathrm{DSPACE}(x \mapsto x^k) \subseteq \mathrm{DTIME}(x \mapsto 4x^{2k})$$

d'où $\mathrm{PSPACE} \subseteq \mathrm{P}$. L'inclusion réciproque est bien évidemment vraie. □

Remarque. La proposition 5.4.2 découle également directement du fait que le problème QSAT est PSPACE-complet et reconnu en espace linéaire sur un automate cellulaire. Cependant la preuve que nous utilisons ici ne fait pas appel à l'existence de problèmes PSPACE-complets.

L'égalité CASPACE(Id) = CATIME(Id) implique également une très forte équivalence entre les machines de Turing à un ruban et les automates cellulaires en dimension 1 :

Proposition 5.4.3
Si CASPACE(Id) $\subseteq$ CATIME(Id), *pour toute fonction s-constructible* f *telle que* $\forall x \in \mathbb{N}, f(x) \geq x$, *on a*

$$\text{CATIME}(f) = \text{DTIME}(f^2) = \text{CASPACE}(f) = \text{DSPACE}(f)$$

Preuve : On sait d'après le théorème 5.4.1 que

$$\text{CATIME}(f) \subseteq \text{DTIME}(f^2) \subseteq \text{CASPACE}(f)$$

or le théorème 5.3.1 nous donne

$$\text{CASPACE}(f) \subseteq \text{CATIME}(f)$$

□

Les automates cellulaires sont un modèle de calcul simple dont la principale différence avec les machines de Turing est qu'ils permettent de modéliser un calcul massivement parallèle. Le fait qu'une machine de Turing à un ruban puisse simuler en temps f^2 le fonctionnement d'un automate cellulaire fonctionnant en temps f pour toute fonction f indique que le gain en temps apporté par le parallélisme est au mieux quadratique. Réciproquement, savoir si ce gain quadratique peut toujours être obtenu en parallélisant un algorithme séquentiel est encore une question ouverte. La proposition 5.4.3 permet de montrer que s'il existe un algorithme fonctionnant en temps f où f est une fonction s-constructible qui ne peut pas être parallélisé avec un gain de temps quadratique alors il existe un langage reconnu en temps $(x \mapsto x^2)$ par une machine de Turing à un ruban qui n'est pas reconnu en temps réel par un automate cellulaire. S'il est faux que tout algorithme séquentiel peut être accéléré quadratiquement en le parallélisant, il existe donc un contre-exemple dans les classes de complexité les plus basses.

Chapitre 6

Changement de dimension

> In a Blackwood article nothing makes so fine a show as your Greek. The very letters have an air of profundity about them. Only observe, madam, the astute look of that Epsilon ! That Phi ought certainly to be a bishop ! Was ever there a smarter fellow than that Omicron ? Just twig that Tau ! In short, there is nothing like Greek for a genuine sensation-paper.
>
> **Edgar Allan Poe** – *How To Write a Blackwood Article*

Ce chapitre sera consacré à l'étude de la traduction d'un automate cellulaire de dimension 3 en un automate cellulaire de dimension 2 qui simulera le fonctionnement du premier. Comme nous le verrons, cette transformation peut également être réalisée pour passer d'un automate cellulaire en dimension quelconque supérieure à 3 à un automate cellulaire de dimension 2. Elle ne peut toutefois pas être utilisée pour obtenir des automates cellulaires unidimensionnels (pour des raisons qui seront expliquées).

Dans tout ce chapitre, on suppose que l'on s'est donné un automate $\mathcal{A}_3$ de dimension 3 fonctionnant sur le voisinage de Von Neumann tridimensionnel

$$V_3 = \{(0,0,0), \pm(1,0,0), \pm(0,1,0), \pm(0,0,1)\}$$

On note $\mathcal{Q}_3$ l'ensemble des états de $\mathcal{A}_3$.

6.1 Description de la simulation

Nous appellerons $\mathcal{A}_2$ l'automate cellulaire de dimension 2 que l'on va construire à partir de $\mathcal{A}_3$. Pour des raisons de commodité $\mathcal{A}_2$ fonctionnera sur le voisinage de Moore. Notons cependant que la simulation de $\mathcal{A}_3$ par $\mathcal{A}_2$ se fera avec un ralentissement polynomial d'ordre 4 (comme nous le verrons par la suite) et qu'il serait donc « équivalent » de faire fonctionner $\mathcal{A}_2$ sur le voisinage de Von Neumann, ou tout autre voisinage complet, moyennant un ralentissement linéaire (dans le cas de Von Neumann, l'automate serait simplement deux fois plus lent que celui que nous allons construire). Nous ne nous intéresserons dans

ce chapitre qu'aux ordres de grandeurs (degrés des polynômes) des distorsions temporelles et spatiales provoquées par le changement de dimension.

On cherche à obtenir une simulation « cellule par cellule », c'est-à-dire qu'il existe une fonction injective $t : \mathbb{Z}^3 \to \mathbb{Z}^2$ telle que l'évolution de la cellule $c \in \mathbb{Z}^3$ de $\mathcal{A}_3$ *se retrouve* dans l'évolution de la cellule $t(c) \in \mathbb{Z}^2$ de $\mathcal{A}_2$.

Plus précisément, on veut qu'il existe un sous ensemble $\mathcal{Q}_{\text{out}} \subseteq \mathcal{Q}_2$ d'états de $\mathcal{A}_2$ et une « fonction de traduction » $\sigma : \mathcal{Q}_{\text{out}} \to Q_3$ tels que pour toute cellule $c \in \mathbb{Z}^3$ de $\mathcal{A}_3$, si l'on note $(q_i)_{i\in\mathbb{N}}$ la suite des états de c lors de l'évolution de $\mathcal{A}_3$ à partir d'une configuration initiale $\mathfrak{C}$ et $(q'_i)_{i\in\mathbb{N}}$ la suite des états de la cellule $t(c)$ dans l'évolution de $\mathcal{A}_2$ à partir de la configuration initiale en dimension 2 correspondant à $\mathfrak{C}$, l'image par σ de la sous-suite de $(q'_i)_{i\in\mathbb{N}}$ obtenue en ne conservant que les éléments dans $\mathcal{Q}_{\text{out}}$ est exactement $(q_i)_{i\in\mathbb{N}}$ (voir figure 6.1).

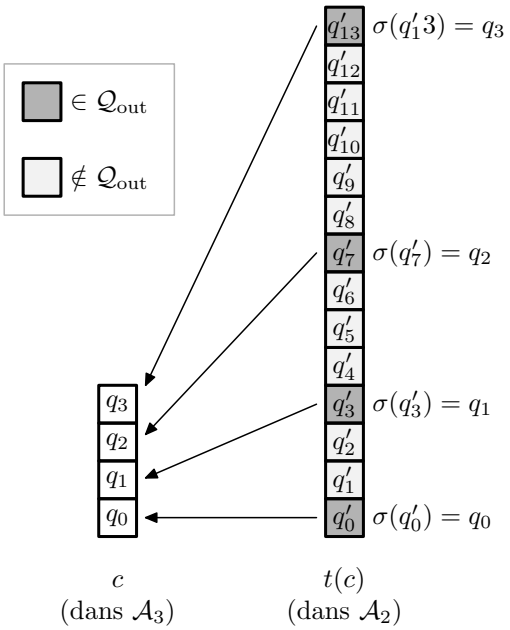

FIGURE 6.1 – Simulation de l'évolution de la cellule c de $\mathcal{A}_3$ par la cellule $t(c)$ de $\mathcal{A}_2$.

Intuitivement cela signifie que si l'on arrive à construire $\mathcal{A}_2$ il suffira, pour connaître toute l'évolution d'une cellule c de $\mathcal{A}_3$, d'observer la cellule $t(c)$ aux temps où elle est dans un état de $\mathcal{Q}_{\text{out}}$.

Dans le fonctionnement normal de $\mathcal{A}_3$, la cellule c doit connaître les états de chacune de ses voisines au temps τ pour calculer son propre état au temps $(\tau + 1)$. Cela signifie que dans l'automate $\mathcal{A}_2$, toute cellule de $t(c) \in t(\mathbb{Z}^3)$ doit, après avoir calculé un nouvel état dans $\mathcal{Q}_{\text{out}}$, le transmettre à toutes les cellules de la forme $t(c')$ où c' est une voisine de c (on appellera ces cellules les t-voisines de $t(c)$) pour que chacune de ces cellules puisse calculer son nouvel état, et ainsi puisque ces t-voisines feront de même $t(c)$ pourra elle aussi calculer le nouvel état de $\mathcal{Q}_{\text{out}}$ correspondant au temps suivant dans l'évolution de $\mathcal{A}_3$.

Choisissons dans $\mathbb{Z}^2$ les 6 directions

$$\{(1,0),(1,1),(0,1),(-1,0),(-1,-1),(0,-1)\}$$

Lorsqu'une cellule de $t(\mathbb{Z}^3)$ calcule un nouvel état dans $\mathcal{Q}_{\text{out}}$ elle émet un signal dans chacune de ces 6 directions contenant le nouvel état calculé. Si l'on fait en sorte que :

- la première cellule de $t(\mathbb{Z}^3)$ que le signal partant vers la droite (direction $(1,0)$) rencontre soit $t(c+(1,0,0))$;
- la première cellule de $t(\mathbb{Z}^3)$ que le signal partant vers le haut (direction $(0,1)$) rencontre soit $t(c+(0,1,0))$;
- la première cellule de $t(\mathbb{Z}^3)$ que le signal partant vers le haut et la droite (direction $(1,1)$) rencontre soit $t(c+(0,0,1))$,

et symétriquement pour les directions opposées, alors il est facile pour les cellules de $t(\mathbb{Z}^3)$ d'échanger leurs informations calculées avec leurs t-voisines, à partir du moment où les cellules de $t(\mathbb{Z}^3)$ sont distinguées des autres cellules de $\mathbb{Z}^2$ (marquées par un certain état par exemple).

6.2 La transformation t de $\mathbb{Z}^3$ dans $\mathbb{Z}^2$

Cherchons maintenant une transformation $t : \mathbb{Z}^3 \to \mathbb{Z}^2$ satisfaisant les spécifications exposées précédemment.

Dans un premier temps, il faut que pour tout point $(a,b,c) \in \mathbb{Z}^3$, la famille $t(x+k,y,z)_{k\in\mathbb{Z}}$ de points de $\mathbb{Z}^2$ soit sur une même ligne horizontale, ordonnée selon k (plus k est grand, plus le point correspondant est à droite). De même, les points $t(x,y+k,z)_{k\in\mathbb{Z}}$ doivent se trouver sur une même colonne ordonnés selon k et enfin les points $t(x,y,z+k)_{k\in\mathbb{Z}}$ doivent se trouver sur une même diagonale, également ordonnés selon k.

Cela se traduit par le fait que la première composante de $t(x,y,z)$ ne dépend pas de y et que la seconde composante ne dépend pas de x. Ainsi il existe deux fonctions $\alpha : \mathbb{Z}^2 \to \mathbb{Z}$ et $\beta : \mathbb{Z}^2 \to \mathbb{Z}$ strictement croissantes en chacune de leurs variables telles que

$$\forall (x,y,z) \in \mathbb{Z}^3, t(x,y,z) = (\alpha(x,z), \beta(y,z))$$

Le fait qu'en ne faisant varier que z on reste sur la même diagonale signifie que $\alpha(x,z) - \beta(y,z)$ ne dépend pas de z et donc finalement il existe 3 fonctions strictement croissantes f, g et h de $\mathbb{Z}$ dans lui-même telles que

$$\forall (x,y,z) \in \mathbb{Z}^3, t(x,y,z) = (f(x)+h(z), g(y)+h(z)) \tag{6.1}$$

Par ailleurs, puisque la première cellule de $t(\mathbb{Z}^3)$ que le signal envoyé par une cellule $t(c)$ dans une certaine direction rencontre doit être la t-voisine de $t(c)$ correspondant à la direction du signal on ne peut pas interposer d'éléments de $t(\mathbb{Z}^3)$ sur un ligne occupée par une famille $t(x+k,y,z)_{k\in\mathbb{Z}}$, ou sur une colonne ou une diagonale occupée.

Cette dernière contrainte se traduit par le fait que si $t(x,y,z)$ est sur la même ligne que $t(x',y',z')$ on a nécessairement $y = y'$ et $z = z'$ (et les deux formulations correspondantes pour les deux autres directions), ce qui revient à dire que les fonctions

$$\left.\begin{array}{rcl} (x,z) & \mapsto & f(x)+h(z) \\ (y,z) & \mapsto & g(y)+h(z) \\ (x,y) & \mapsto & f(x)-g(y) \end{array}\right\} \text{ sont injectives.} \tag{6.2}$$

Il existe de nombreuses fonctions vérifiant les contraintes 6.1 et 6.2. Par exemple

$$f,g,h:\mathbb{Z}\to\mathbb{Z} \qquad \begin{cases} f(k) &=& \mathrm{sgn}(k).2^{3|k|} \\ g(k) &=& \mathrm{sgn}(k).2^{3|k|+1} \\ h(k) &=& \mathrm{sgn}(k).2^{3|k|+2} \end{cases} \qquad (\text{où } \mathrm{sgn}(x) \text{ est le signe de } x)$$

ou encore

$$f,g,h:\mathbb{Z}\to\mathbb{Z} \qquad \begin{cases} f(k) &=& \mathrm{sgn}(k).2^{|k|} \\ g(k) &=& \mathrm{sgn}(k).3^{|k|} \\ h(k) &=& \mathrm{sgn}(k).5^{|k|} \end{cases} \qquad (\text{où } \mathrm{sgn}(x) \text{ est le signe de } x)$$

mais nous nous intéresserons ici à une solution pour laquelle les fonctions f, g et h croissent polynomialement.

On a le résultats suivant :

Proposition 6.2.1
Il existe trois fonctions strictement croissantes f, g et h de $\mathbb{Z}$ dans lui-même, bornées par des polynômes de degré 3 et satisfaisant les contraintes 6.2.

Preuve : On considère les fonctions f, g et h définies inductivement par :

- chacune des trois fonctions est impaire ($f(0) = 0$ et pour tout x, $f(-x) = -f(x)$) ;
- pour tout $n \in \mathbb{N}$, $f(n+1)$ est par définition le plus petit entier naturel n'appartenant pas à
  $\{f(i)+g(j)+g(k),\quad f(i)+h(j)+h(k) \quad | \quad |i|,|j|,|k| \le n\}$;
- pour tout $n \in \mathbb{N}$, $g(n+1)$ est par définition le plus petit entier naturel n'appartenant pas à
  $\{g(i)+h(j)+h(k), g(i)+f(j')+f(k') \quad | \quad |i|,|j|,|k| \le n, |j'|,|k'| \le n+1\}$
- pour tout $n \in \mathbb{N}$, $h(n+1)$ est par définition le plus petit entier naturel n'appartenant pas à
  $\{h(i)+f(j)+f(k),\quad h(i)+g(j)+g(k) \quad | \quad |i|,|j|,|k| \le n+1\}$

Il est immédiat, de par leur définition que les fonctions f, g et h définissent une fonction t satisfaisante, l'injectivité étant assurée étape par étape puisque chaque nouvelle valeur est choisie de manière à ne pas créer de conflit avec les valeurs précédemment définies.

De plus,

$$\forall n \in \mathbb{N}, \quad \begin{cases} \mathrm{Card}\,\{f(i)+g(j)+g(k) \mid -n \le i,j,k \le n\} &\le& (2n+1)^3 \\ \mathrm{Card}\,\{f(i)+h(j)+h(k) \mid -n \le i,j,k \le n\} &\le& (2n+1)^3 \end{cases}$$

Ainsi pour tout n l'union de ces deux ensembles est de cardinal strictement inférieur à $2(2n+1)^3$ puisque 0 appartient aux deux ensembles. D'après les définitions de f, g et h, on obtient alors

$$\forall n \in \mathbb{N}, \quad \begin{cases} f(n+1) &\le& 2(2n+1)^3 &=& O(n^3) \\ g(n+1) &\le& 2(2n+1)(2n+3)^2 &=& O(n^3) \\ h(n+1) &\le& 2(2n+1)(2n+3)^2 &=& O(n^3) \end{cases} \tag{6.3}$$

□

A partir de maintenant on considère que les fonctions f, g et h sont celles qui ont été définies dans la preuve de la proposition 6.2.1. La transformation t est donc définie par

$$\forall (x,y,z) \in \mathbb{Z}^3, \quad t(x,y,z) = (f(x)+h(z), g(y)+h(z))$$

Le tableau 6.2 donne les premières valeurs des fonction f, g et h. On peut voir que les fonctions sont irrégulières mais semblent effectivement croître de manière cubique (l'argument que l'on a donné permet simplement de trouver une borne supérieure mais il est difficile de donner une borne inférieure également cubique). Puisqu'à chaque étape c'est la valeur de f qui est choisie en premier et celle de h qui est choisie en dernier, on pourrait penser que f sera inférieure à g, elle-même inférieure à h. Cependant il semble (au vu des premières valeurs uniquement) que les trois fonctions restent assez proches l'une de l'autre pour que f passe au dessus des deux autres de temps en temps.

Ce comportement apparemment erratique et le fait que la définition des fonctions n'est pas aisée à manipuler font que l'on n'a pas trouvé d'expression simple de ces fonctions. Cependant, il est possible de construire ces trois fonctions à l'aide d'un automate cellulaire en appliquant directement la définition.

n	$f(n)$	$g(n)$	$h(n)$
0	0	0	0
1	1	3	4
2	10	15	17
3	26	38	46
4	57	75	65
5	84	116	128
6	143	119	176
7	152	223	193
8	270	276	340
9	327	380	386
10	510	553	579
11	616	646	775
12	724	862	803
13	1010	1041	1163
14	1197	1145	1388
15	1448	1365	1610
16	1528	1649	1773

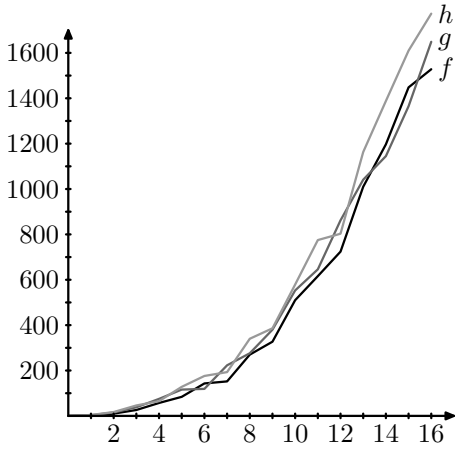

FIGURE 6.2 – Les premières valeurs de f, g et h.

6.3 La construction de $t(\mathbb{Z}^3)$

Dans toute la suite de ce chapitre la norme considérée sera la norme 1 sur $\mathbb{Z}^3$ définie par

$$\|(a,b,c)\| = |a| + |b| + |c|$$

L'objectif de cette section est de montrer qu'il est possible de construire l'ensemble $t(\mathbb{Z}^3)$ à l'aide d'un automate cellulaire de dimension 2 en temps

polynomial, au sens où il existe un polynôme $P \in \mathbb{Z}[X]$ tel que pour toute cellule $c \in \mathbb{Z}^3$, la cellule $t(c) \in \mathbb{Z}^2$ est marquée comme appartenant à $t(\mathbb{Z}^3)$ en temps au plus $P(\|c\|)$. On se propose donc de montrer dans un premier temps le théorème suivant :

Théorème 6.3.1
Il existe un automate cellulaire de dimension 2 travaillant sur le voisinage de Moore qui, partant d'une configuration où toutes les cellules sont dans un même état quiescent exceptée l'origine, va marquer d'un état particulier toutes les cellules de $t(\mathbb{Z}^3)$ en temps polynomial de degré 4 : pour toute cellule $c \in \mathbb{Z}^3$ la cellule $t(c)$ est marquée en temps $O(\|c\|^4)$.

6.3.1 Les calculs sur les axes

Pour construire la totalité des cellules de $t(\mathbb{Z}^3)$ nous allons commencer par calculer les valeurs des fonctions f, g et h définies précédemment sur l'axe des abscisses. Puisque les définitions de ces fonctions sont récursives et très liées l'une à l'autre il est nécessaire de calculer les valeurs des trois fonctions en parallèle et sur le même axe, afin de pouvoir utiliser toutes les valeurs précédemment calculées pour déterminer les valeurs suivantes. Nous calculerons donc successivement les valeurs $f(1)$, $g(1)$, $h(1)$, $f(2)$, $g(2)$, etc.

Dans toute cette section nous associerons l'axe des abscisses à l'ensemble $\mathbb{Z}$ de manière canonique, la cellule $(n, 0)$ étant alors simplement notée n.

L'idée de la construction est de marquer, après avoir construit le point $f(k)$, les points $f(k)\pm g(i)\pm g(j)$, $f(k)\pm h(i)\pm h(j)$, $g(i)\pm f(j)\pm f(k)$ et $h(i)\pm f(j)\pm f(k)$ pour tout $i, j < k$, et une construction similaire après les calculs de $g(k)$ et $h(k)$.

Si l'on suppose par récurrence que tous les points cités précédemment ont été marqués après le calcul de $f(k+1)$, et que de plus le point $g(k)$ est déjà construit, il suffit alors de déplacer un signal vers la droite depuis la cellule $g(k)$ jusqu'à trouver la première valeur qui n'est pas « interdite » par la définition de g et de la marquer comme étant la valeur suivante de la fonction g.

Par ailleurs, pour la suite de la construction, l'axe vertical ($y = 0$), l'axe horizontal ($x = 0$) et « l'axe diagonal » ($x = y$) seront marqués par des signaux partant de l'origine et se déplaçant à vitesse maximale, de telle sorte qu'il sera facile par la suite de savoir si une cellule est ou non sur l'un de ces axes (et sur lequel).

Nous ne décrirons ici que la construction des fonctions f, g et h sur la partie positive de l'axe des abscisses. La construction symétrique sur la partie négative de l'axe a cependant lieu en même temps.

Additions et soustractions sur l'axe

On suppose ici que notre automate marque certains points de l'axe au cours de son évolution d'un état q. On veut alors faire en sorte que toutes les sommes possibles de deux points q soient marquées (par un autre état) après un certain temps.

Pour ce faire, l'automate va envoyer un signal se déplaçant vers le haut dès qu'une cellule de l'axe est marquée q. Lorsque ce signal rencontre la diagonale ($y = x$) il change de direction pour se déplacer vers la droite. Au cours de son déplacement, le signal laisse une « trace », c'est-à-dire que toute cellule par laquelle il est passé reste marquée (voir figure 6.3).

Lorsqu'un signal se déplaçant vers le haut (issu de la cellule $(b, 0)$) ou sa trace rencontre un signal se déplaçant vers la droite (issu de la cellule $(a, 0)$ et donc ayant rencontré la diagonale en (a, a)) ou sa trace, un nouveau signal est émis qui se déplace en diagonale vers la droite et le bas. La collision entre les deux signaux a lieu sur la cellule (b, a), le signal résultant rencontrera l'axe des abscisses au point $(a + b, 0)$ et le marquera donc.

Cette construction est illustrée sur la figure 6.3.

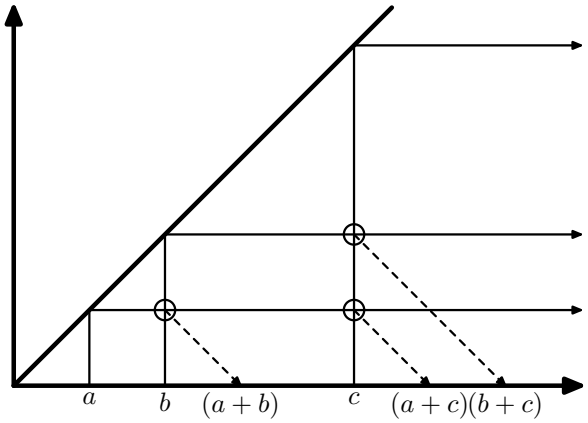

FIGURE 6.3 – Additions sur les axes.

Si l'on veut maintenant construire toutes les différences positives entre des points marqués q, il suffit que chaque point lorsqu'il est marqué de l'état q émette deux signaux se déplaçant dans chacune des deux diagonales orientées vers le haut (voir figure 6.4). Lorsqu'un signal se déplaçant vers le haut et la gauche rencontre la diagonale $(y = x)$ il s'arrête.

Lorsque qu'un signal diagonal allant vers la gauche (issu du point $(a, 0)$) rencontre un signal diagonal allant vers la droite (issu du point $(b, 0)$) un signal se déplaçant vers la gauche est émis. Lorsque ce dernier signal rencontre la diagonale $(y = x)$ il est redirigé vers le bas et la droite. Lorsqu'il atteint l'axe des abscisses il marque le point.

Si l'on suppose que $b > a$, la collision entre les deux signaux diagonaux a lieu au point $((a + b)/2, (b - a)/2)$, le signal résultant rencontre alors la diagonale au point $((b - a)/2, (b - a)/2)$ et le point ainsi marqué sur l'axe des abscisses est $(b - a, 0)$.

Cette dernière construction est illustrée sur la figure 6.4.

Si l'on suppose que les points a et b $(b > a)$ étaient marqués sur l'axe aux temps respectifs $\tau(a)$ et $\tau(b)$, alors les constructions précédentes permettent de marquer les points $(a + b)$ et $(b - a)$ aux temps

$$\begin{array}{rcl} \tau(a+b) & = & \max(\tau(a) + a + b, \tau(b) + 2a) \\ \tau(b-a) & = & \max(\tau(a) + b, \tau(b) + b) \end{array} \tag{6.4}$$

Calcul des fonctions f, g et h sur l'axe

Pour marquer les ensembles $f(\mathbb{N})$, $g(\mathbb{N})$ et $h(\mathbb{N})$ sur l'axe en utilisant la définition des trois fonctions, il est nécessaire de construire parallèlement les ensembles

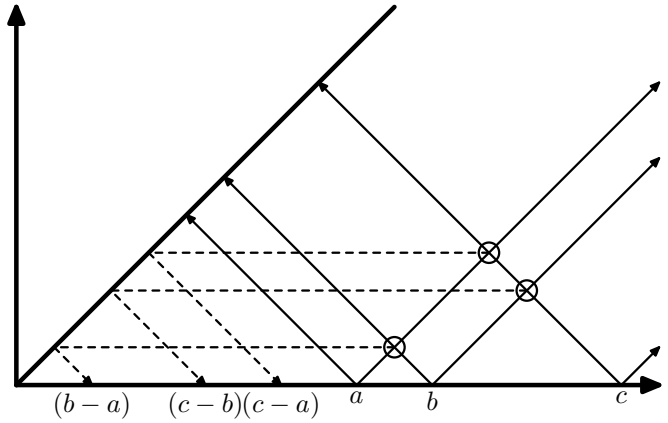

FIGURE 6.4 – Soustractions sur les axes.

des valeurs « interdites » pour chaque fonction. Supposons que l'on est dans une configuration telle que les ensembles suivants de cellules soient marqués sur l'axe (les ensembles ne sont pas nécessairement d'intersection vide, et une même cellule peut être marquée comme appartenant à plusieurs ensembles à la fois) :

$$\begin{array}{l} F = \{f(k) \mid 0 \leq k \leq n\}, \\ 2F = \{f(k) \pm f(k') \mid 0 \leq k, k' \leq n\} \cap \mathbb{N} = (F \pm F) \cap \mathbb{N} \quad \text{et} \\ \overline{F} = (\pm F \pm 2G \cup \pm F \pm 2H) \cap \mathbb{N} \end{array}$$

ainsi que les ensembles correspondants, en remplaçant f par g et h.

On suppose également que $f(n)$, $g(n)$ et $h(n)$ sont marquées différemment des autres éléments de F, G et H afin d'indiquer que ce sont les dernières valeurs qui ont été calculées, et que tout point qui a été marqué a envoyé des signaux verticalement et en diagonale afin de pouvoir effectuer des additions et des soustractions comme il a été expliqué dans la sous-section précédente. Chacun des signaux envoyé par un point de l'axe porte l'information de l'ensemble auquel le point appartient ce qui permet par la suite de savoir, lorsque l'on effectue une addition ou une soustraction, de quel ensemble provenaient les deux éléments de l'opération pour pouvoir marquer correctement le point qui en résulte.

À partir de cette situation, construire $f(n+1)$ est un jeu d'enfant puisqu'il suffit de déplacer un signal à partir de $f(n)$ vers la droite jusqu'à trouver la première cellule qui n'est pas dans $\overline{F}$. Lorsque ce signal part, la cellule $f(n)$ perd son marquage qui indiquait que c'était la dernière valeur de f calculée, et c'est $f(n+1)$ qui le reprend lorsqu'elle est marquée. Dès qu'il est construit, le point $f(n+1)$ est marqué comme appartenant aux ensembles F, $2F$, $\overline{F}$, $\overline{G}$ et $\overline{H}$. De plus de nouveaux signaux sont envoyés pour effectuer des additions et des soustractions afin de marquer les cellules des ensembles suivants :

$$\begin{array}{l} f(n+1) \pm F \subseteq 2F, \\ 2f(n+1) \in 2F, \\ \pm f(n+1) \pm 2G \subseteq \overline{F} \text{ et} \\ \pm f(n+1) \pm 2H \subseteq \overline{F} \end{array}$$

Les nouveaux points de $2F$ émettent à leur tour des signaux pour marquer les nouvelles cellules des ensembles $2F + G \subseteq \overline{G}$ et $2F + H \subseteq \overline{H}$.

Pour que le point $g(n+1)$ soit ensuite construit correctement par une méthode similaire, il faut être sûr que l'ensemble $\overline{G}$ est complètement construit lorsque le signal part de $g(n)$. Si l'on note $\tau_f(n+1)$ le temps auquel la cellule $f(n+1)$ est marquée comme appartenant à F, les inégalités (6.4) indiquent que tous les points de $2F$ sont correctement marqués au temps $\tau_f(n+1)+2f(n+1)$ et donc que tous les points de $\overline{G}$ sont correctement marqués au temps

$$\begin{array}{rl} & [\tau_f(n+1)+2f(n+1)]+[2f(n+1)+2g(n)] \\ = & \tau_f(n+1)+4f(n+1)+2g(n) \end{array}$$

Ainsi, si l'on attend $4f(n+1)+2g(n)$ générations avant de faire partir le signal de $g(n)$ qui doit construire $g(n+1)$, nous sommes sûrs que $\overline{G}$ est correctement marqué. Pour pouvoir attendre le temps nécessaire, la cellule $f(n+1)$ va émettre deux nouveaux signaux lorsqu'elle est marquée. Le premier se déplace vers la gauche à vitesse $1/4$ tandis que le second se déplace vers la droite à vitesse maximale. L'évolution de ces signaux ne peut être affectée que par les cellules $f(n+1)$, $g(n)$ et l'origine 0. On est dans l'un des deux cas suivants :

- $0 < g(n) < f(n+1)$ (voir figure 6.5). Dans ce cas, le signal se déplaçant vers la droite ne rencontrera aucune des cellules qu'il cherche et continuera donc vers la droite indéfiniment. Puisqu'il se déplace à vitesse maximale, il n'y a aucun risque qu'il soit rattrapé par la construction et donc qu'il rencontre par la suite un autre point marqué de $g(\mathbb{N})$. Le signal se dirigeant vers la gauche quant à lui va rencontrer $g(n)$ et continuer sa route, toujours à vitesse $1/4$ tout en se souvenant qu'il a croisé $g(n)$. Puis il arrivera à l'origine où il fera demi-tour et repartira vers la droite à vitesse $1/2$. Lorsqu'il atteint $g(n)$ pour la seconde fois il a parcouru toute la distance séparant $f(n+1)$ de 0 à vitesse $1/4$ puis la distance de 0 à $g(n)$ à vitesse $1/2$. Il atteint donc $g(n)$ après $4f(n+1)+2g(n)$ étapes et peut donc lancer la construction de $g(n+1)$.

- $0 < f(n+1) < g(n)$ (voir figure 6.6). Cette fois-ci, le signal partant vers la gauche ne rencontrera pas $g(n)$ avant d'atteindre l'origine. Il disparaît alors en arrivant en 0. De son côté, le signal se déplaçant vers la droite rencontre $g(n)$ après quoi il fait demi-tour et repart vers la gauche à vitesse 1. Lorsqu'il repasse par $f(n+1)$ il ralentit et continue vers l'origine à vitesse $1/4$. Lorsqu'il atteint l'origine il fait demi-tour et repart vers la droite à vitesse $1/2$. Lorsqu'il passe à nouveau par $f(n+1)$ il accélère et se dirige finalement vers $g(n)$ à vitesse maximale. Lorsqu'il atteint $g(n)$ pour la seconde fois, il a parcouru la distance de $f(n+1)$ à 0 à vitesse $1/4$ une première fois, puis à vitesse $1/2$ une seconde fois. Par ailleurs il a parcouru la distance de $f(n+1)$ à $g(n)$ 3 fois à vitesse 1. Il s'est donc écoulé

  $$\begin{array}{rlcl} & 6f(n+1)+3(g(n)-f(n+1)) & & \\ = & 3f(n+1)+3g(n) & > & 4f(n+1)+2g(n) \end{array}$$

  étapes et le calcul de $g(n+1)$ peut démarrer sans risque.

Dans chacun des cas, nous avons vu que le signal indiquant à la cellule $g(n)$ de partir chercher la première cellule libre pour marquer $g(n+1)$ arrive assez tard pour que la totalité de l'ensemble $\overline{G}$ soit marqué. Ainsi, $g(n+1)$ peut être marqué correctement. À son tour la cellule $g(n+1)$ va émettre des signaux

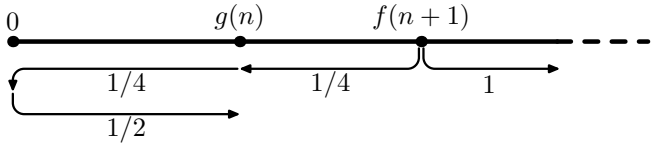

FIGURE 6.5 – Le cas $0 < g(n) < f(n+1)$.

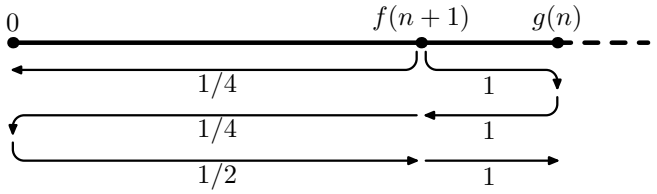

FIGURE 6.6 – Le cas $0 < f(n+1) < g(n)$.

afin d'effectuer les additions et soustractions nécessaires afin de mettre à jour les ensembles $\overline{F}$, $\overline{G}$ et $\overline{H}$ (ainsi que tous les intermédiaires), et deux signaux qui permettront d'attendre suffisamment longtemps avant de prévenir la cellule $h(n)$ qu'elle peut à son tour calculer la valeur suivante de h. Finalement, lorsque $f(n+1)$ reçoit le signal provenant de $h(n+1)$, on est à nouveau dans les hypothèses que l'on s'était données pour construire $f(n+1)$. La construction peut donc continuer inductivement, pour marquer la totalité des points de $f(\mathbb{N})$, $g(\mathbb{N})$ et $h(\mathbb{N})$.

On a par ailleurs les inégalités suivantes (ici encore $\tau_f(n+1)$ désigne le temps auquel la cellule $f(n+1)$ est marquée comme appartenant à F, et symétriquement pour les fonctions g et h) :

$$\begin{array}{lcl} \tau_g(n+1) & \geq & \tau_f(n+1) + 4f(n+1) + 2g(n) \\ \tau_h(n+1) & \geq & \tau_g(n+1) + 4g(n+1) + 2h(n)) \\ \tau_f(n+2) & \geq & \tau_h(n+1) + 4h(n+1) + 2f(n+1)) \end{array} \tag{6.5}$$

Ce qui permet notamment d'obtenir en combinant les trois inégalités précédentes :

$$\begin{array}{rcl} \tau_h(n+1) & \geq & \tau_h(n) + 4(h(n) + f(n+1) + g(n+1)) \\ & & +2(f(n) + g(n) + h(n)) \\ & \geq & \tau_h(n) + 6f(n) + 6g(n) + 6h(n) \end{array} \tag{6.6}$$

Bornes polynomiales

Nous allons donner ici une majoration du temps nécessaire à la construction des fonctions f, g et h sur l'axe des abscisses telle qu'elle a été décrite précédemment. On a

$$\tau_f(0) = \tau_g(0) = \tau_h(0) = 0$$

puisque l'état de l'origine est fixée sur la configuration initiale. Après avoir marqué $f(n+1)$ on attend au plus $4f(n+1) + 3g(n)$ étapes avant que le signal parte de $g(n)$ pour marquer $g(n+1)$ (selon le cas le signal met $4f(n+1)+2g(n)$ ou $3f(n+1)+3g(n)$ étapes à revenir en $g(n)$). Ce signal met exactement $g(n+1) - g(n)$ étapes à atteindre son but. D'après les inégalités (6.3) on sait que les

fonctions f, g et h sont bornées par des polynômes de degré 3, on a donc pour tout $n \in \mathbb{N}$,

$$\tau_g(n+1) = \tau_f(n+1) + O(n^3)$$

On a bien évidemment des résultats similaires pour les fonctions f et h. On obtient donc

$$\tau_g(n+1) = \tau_g(n) + O(n^3) = \sum_{k=0}^{n} O(k^3) = O(n^4) \tag{6.7}$$

et de même pour les deux autre fonctions.

L'automate que l'on a décrit construit donc les fonctions f, g et h en temps polynomial (de degré 4) sur l'axe des abscisses.

Construction sur les autres axes

Lorsque la cellule $g(n)$ est marquée sur l'axe des abscisses, elle émet un signal se propageant vers le haut et la gauche (direction $(-1,-1)$) qui atteint l'axe des ordonnées au point $(0, g(n))$ après $g(n)$ étapes et le marque. De même, lorsque la cellule $h(n)$ est marquée sur l'axe des abscisses elle émet un signal vertical qui va marquer la cellule $(h(n), h(n))$ en rencontrant la diagonale $(y = x)$ après $h(n)$ générations.

Ainsi, en temps polynomial (également de degré 4), les points de la forme $t(0, n, 0) = (0, g(n))$ sont marqués sur l'axe des ordonnées et les points de la forme $t(0, 0, n) = (h(n), h(n))$ sont marqués sur la diagonale.

Par ailleurs, on rappelle que la construction symétrique de celle que nous avons décrite précédemment a lieu sur la partie négative de l'axe des abscisses, et les valeurs ainsi calculées sont également reportées sur les parties négatives de l'axe des ordonnées et de la diagonale.

La construction exposée jusqu'ici permet donc de construire les images par t des trois axes de $\mathbb{Z}^3$.

6.3.2 Construction du reste de $t(\mathbb{Z}^3)$

Soit p un point de $\mathbb{Z}^3$, avec $\|p\| = k$. On dira que $p' \in \mathbb{Z}^3$ est un *voisin supérieur* (resp. *voisin inférieur*) de p si p' est un voisin de p et que $\|p'\| = k+1$ (resp. $\|p'\| = k-1$). Dans $\mathbb{Z}^2$ on dira que $t(p')$ est un *t-voisin supérieur* (resp. *t-voisin inférieur*) de $t(p)$.

Le nombre de voisins supérieurs et inférieurs d'un point $p \in \mathbb{Z}^3$ et leur position par rapport à p ne dépendent que du signe des coordonnées de p. Tout point a ainsi un voisin supérieur dans la direction du signe de chacune de ses coordonnées non nulles et deux voisins supérieurs dans chacun des deux sens correspondant à une de ses coordonnées nulles. Inversement, un point n'a de voisin inférieur que dans le sens inverse du signe de ses coordonnées non nulles.

Par exemple, le point $(1, 2, -4)$ a 3 voisins supérieurs, un dans chacune des directions $\vec{x}$, $\vec{y}$ et $-\vec{z}$, et 3 voisins inférieurs dans les directions $-\vec{x}$, $-\vec{y}$ et $\vec{z}$. Par contre le point $(2, 0, -4)$ a 4 voisins supérieurs, dans les directions $\vec{x}$, $\vec{y}$, $-\vec{y}$ et $-\vec{z}$, tandis qu'il n'a que deux voisins inférieurs, dans les directions $-\vec{x}$ et $\vec{z}$.

Ainsi, tout point n'étant pas sur l'un des trois axes a au moins deux voisins inférieurs (un pour chaque coordonnée non nulle).

La construction par induction

Pour tout $i \in \mathbb{N}$, on notera S_i la sphère de rayon i dans $\mathbb{Z}^3$ relativement à la norme 1 ($S_i = \{c \in \mathbb{Z}^3 \mid \|c\| = i\}$).

Nous allons voir ici comment il est possible de construire inductivement les images par t de toutes les sphères S_i. Soit $k \in \mathbb{N}$, on suppose que l'on est dans une configuration de l'automate telle que :

- Pour tout $p \in S_k$, la cellule $t(p)$ est marquée d'un état S_{+0} et connaît le signe de chacune des coordonnées de p, son antécédent par t. Par ailleurs, seuls les points de $t(S_k)$ sont marqués S_{+0}. L'état S_{+0} signifie ici que ce point appartient à l'avant dernière sphère qui a été calculée. Il n'est en effet pas possible de donner à chaque point de $t(\mathbb{Z}^3)$ le numéro de la sphère à laquelle il appartient puisque cela demanderait une infinité d'états différents. On se contentera donc de marquer les points des deux dernières sphères construites, et l'on mettra à jour ces marquages au cours de la construction de la sphère suivante.

- Pour tout $p \in S_{k+1}$, la cellule $t(p)$ est marquée par un état S_{+1} et connaît le signe de chacune des coordonnées de p. Ici encore, chaque cellule marquée S_{+1} est l'image par t d'un point de S_{k+1}.

Une telle situation est illustrée sur la figure 6.7 pour $k = 1$.

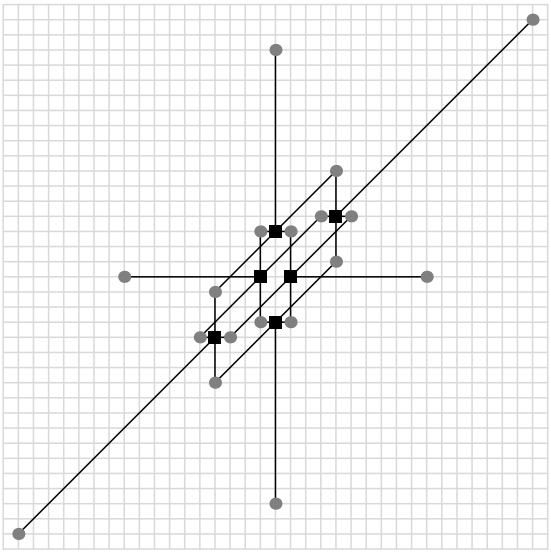

FIGURE 6.7 – Les images par t des sphères S_k (carrés noirs) et S_{k+1} (points gris) pour $k = 1$.

Des signaux s_0 sont alors émis à partir de l'origine, se propageant dans toutes les directions de manière à atteindre toute cellule de $\mathbb{Z}^2$. Il existe un signal s_0 pour chaque paire de directions (distinctes) parmi $\{\uparrow, \nearrow, \rightarrow, \downarrow, \swarrow, \leftarrow\}$ exceptées les paires de directions opposées. Nous verrons plus en détail les différents types de signaux s_0 et la façon dont chacun se propage dans la sous-section suivante, et nous expliquerons également ce qui provoque l'apparition de ces signaux.

Lorsqu'une cellule de S_{+0} reçoit un signal s_0 correspondant à une paire de directions $\{d_1, d_2\}$, si cette cellule a des t-voisines supérieures dans les deux directions d_1 et d_2, elle émet un signal s_1 dans ces deux directions. Le signal émis dans la direction d_1 porte l'information d_2 et réciproquement. Lorsqu'une cellule S_{+0} a reçu tous les signaux s_0 possibles elle efface son marquage S_{+0}.

Lorsqu'une cellule $t(c_1)$ de S_{+1} reçoit un signal s_1 se déplaçant dans la direction d_1 et portant l'information d_2 provenant d'une de ses t-voisines inférieures $t(c_0)$, elle émet un signal s_2 portant l'information $\{d_1, d_2\}$ dans la direction d_2. Il est important de noter qu'il existe bien une t-voisine supérieure de $t(c_1)$ dans la direction d_2 car $t(c_0)$ a des t-voisines supérieures dans les deux directions d_1 et d_2, donc

$$\|c_0 + d_1\| = \|c_0 + d_2\| = \|c_0\| + 1$$

ce qui signifie que

$$\|c_0 + d_1 + d_2\| = \|c_1 + d_2\| = \|c_0\| + 2 = \|c_1\| + 1$$

Lorsqu'une cellule de S_{+1} a reçu tous les signaux s_1 de chacune de ses t-voisines inférieures (elle sait combien elle en attend) elle devient une cellule de S_{+0} et n'appartient plus à S_{+1}.

Afin que ces nouvelles cellules S_{+0} n'interagissent pas avec le signal s_0 lors de l'étape k (on veut qu'elles attendent la prochaine série de signaux s_0), on utilise une information de parité. Ainsi, les cellules de S_{+0} ne réagissent en réalité qu'aux signaux s_0 qui sont de même parité, ce qui permet de ne pas confondre les différentes étapes.

Les émissions de signaux s_1 et s_2 correspondant à l'étape $k = 1$ sur le quart de plan supérieur droit sont illustrées sur la figure 6.8.

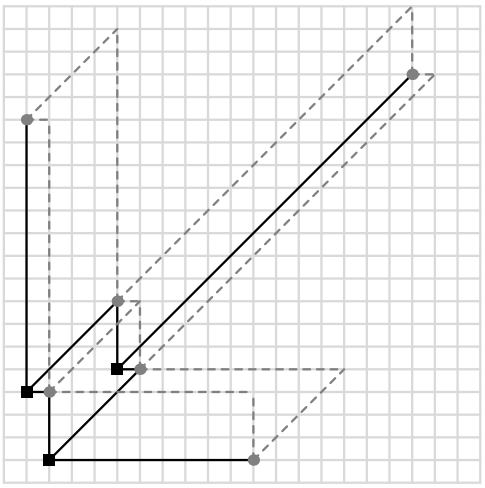

FIGURE 6.8 – Les signaux s_1 (en noir) et s_2 (en gris discontinu) dans le quart de plan supérieur droit pour $k = 1$.

Lorsque deux signaux s_2 portant la même informations (les deux mêmes directions) se rencontrent sur une cellule, les deux signaux disparaissent et la cellule où la collision a eu lieu est marquée comme appartenant à S_{+1} (cette cellule ne recevra aucun signal s_1 de ses t-voisines inférieures avant l'étape suivante,

elle n'influera donc pas sur la construction à l'étape k). De plus, les directions des signaux s_2 arrivant sur la cellule indiquent les positions des t-voisines inférieures de la nouvelle cellule marquée, ce qui permet lui permet de déduire, lorsque tous les signaux s_2 sont arrivés, les positions de toutes ses t-voisines supérieures et les signes de chacune des coordonnées de son antécédent par t.

Les signaux s_0

La validité de la construction que nous avons décrite dans la sous-section précédente repose sur le fait que si deux signaux s_2 semblables se rencontrent sur une cellule, cette cellule est bien dans $t(\mathbb{Z}^3)$. Plus précisément, il faut que les deux signaux proviennent de deux cellules qui sont elles-mêmes deux t-voisines supérieures d'une même cellule de S_{+0}. La figure partie gauche de la figure 6.9 illustre ce scénario. Il est aisé de vérifier que, puisque les signaux s_1 et s_2 se déplacent à vitesse maximale, si les deux signaux s_1 partent en même temps, ce qui est le cas, les deux signaux s_2 se trouveront bien sur la cellule c_2 au même instant, ce qui est nécessaire pour marquer c_2.

Cependant, il est *a priori* possible que deux signaux s_2 de même type, provenant initialement de deux cellules distinctes c_0 et c'_0 de $t(S_k)$ se rencontrent en un point donné, comme illustré sur partie droite de la figure 6.9. Dans ce cas le signal s_2 se déplaçant verticalement émis par la cellule c'_{1b} rencontre le signal s_2 se déplaçant horizontalement émis par c_{1a}. Dans ce cas la cellule où a lieu la collision (indiqué par une croix sur le schéma) est marquée à tort comme appartenant à $t(S_{k+2})$, les signaux ayant pris part à la collision disparaissent donc et ne marqueront pas les cellules c_2 et c'_2 qui auraient dû l'être, et finalement les deux signaux restants vont se rencontrer en la seconde cellule marquée d'une croix sur la figure et la marquer également comme appartenant à $t(S_{k+2})$.

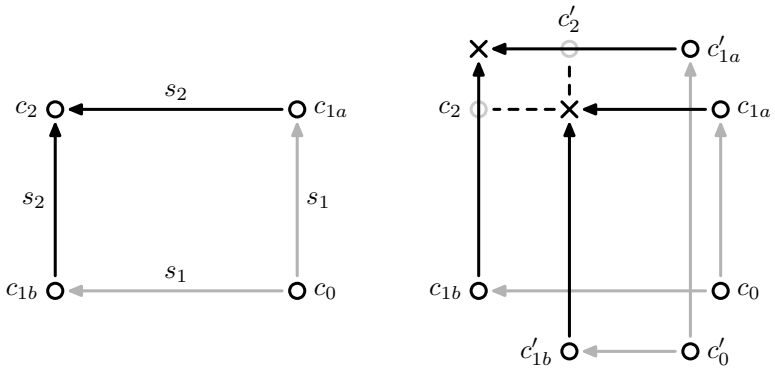

FIGURE 6.9 – Possibilité de conflit.

Observons précisément les conditions dans lesquelles une telle collision peut arriver, dans le cas de signaux dont les directions sont $\{\uparrow, \leftarrow\}$ comme le sont ceux de la figure (notons que si les signaux ne sont pas de mêmes directions l'automate sait alors qu'ils ne sont pas destinés à se rencontrer et chacun poursuit donc sa route sans incident).

Si l'on choisit de repérer chaque cellule de $\mathbb{Z}^2$ par la diagonale $(y-x=d)_{d\in\mathbb{Z}}$ à laquelle il appartient (voir figure 6.10), il est immédiat que si un signal se déplaçant dans l'une des directions $\uparrow$ ou $\leftarrow$ se trouve sur une diagonale à un

temps donné, il se trouvera sur la diagonale suivante au temps suivant. Ainsi, si les cellules c_0 et c'_0 se trouvent sur les diagonales d_0 et d'_0 respectivement et reçoivent le signal s_0 correspondant aux directions $\{\uparrow, \leftarrow\}$ aux temps respectifs τ_0 et τ'_0 (on suppose $\tau'_0 \geq \tau_0$) de telle sorte que les signaux s_2 engendrés se rencontrent après un certain temps sur la même cellule de la diagonale d_2, cela signifie alors que

$$\tau'_0 - \tau_0 = d'_0 - d_0 \tag{6.8}$$

En effet, puisque les signaux s_2 se rencontrent sur la même case, c'est qu'ils sont sur la même diagonale lors de la collision. Or chacun des signaux (qu'il soit s_1 ou s_2) avançait d'exactement une diagonale par génération et donc cela signifie que lorsque la cellule c'_0 a émis son signal s_1 au temps τ'_0, elle se trouvait sur la même diagonale que le signal s_1 déjà envoyé par c_0. Puisque ce signal s_1 se déplaçait à raison d'une diagonale par génération, il a été émis exactement $d'_0 - d_0$ générations avant τ'_0, d'où le résultat.

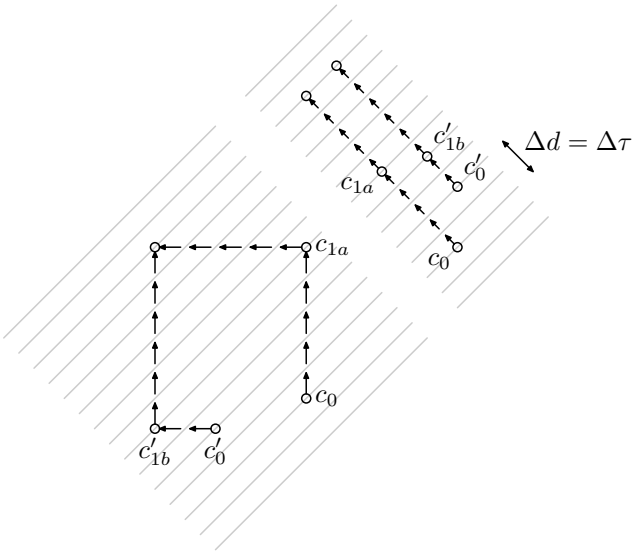

FIGURE 6.10 – Projection selon les diagonales.

La partie en haut à droite de la figure 6.10 illustre ce raisonnement, en projetant les déplacements des signaux en une unique dimension qui correspond à l'indexation des diagonales. Le résultat (6.8) est alors évident.

On se place maintenant dans le quart de plan supérieur droit, et l'on suppose que le signal s_0 avance selon les deux directions $\uparrow$ et $\rightarrow$ (en formant une ligne de front diagonale, orthogonale à la diagonale $(y = x)$). Pour que les deux cellules c_0 et c'_0 produisent des signaux qui créent un conflit comme nous l'avons vu précédemment, il faut que la différence $\tau'_0 - \tau_0$ soit égale à la différence $d'_0 - d_0$ des diagonales sur lesquelles sont c_0 et c'_0, ce qui signifie ici que les deux cellules sont sur la même colonne (voir figure 6.11).

Or, d'après la définition de la transformation $t : \mathbb{Z}^3 \rightarrow \mathbb{Z}^2$, il n'existe qu'au plus deux points de $t(S_k)$ sur une même colonne, qui sont les deux points images de points symétriques l'un de l'autre dans $\mathbb{Z}^3$ par rapport au plan $y = 0$ (ce sont les deux points ayant les mêmes première et troisième coordonnée). Si

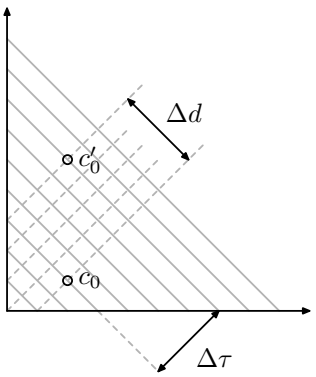

FIGURE 6.11 – les lignes de front de s_0 (en gris plein) et deux cellules qui créent un conflit.

l'on donne aux signaux s_1 et s_2 les signes des coordonnées du point dont c_0 est l'image par t, les deux signaux ne créeront plus de conflit puisqu'en cas de collision incorrecte l'automate a assez d'information pour déterminer que les deux signaux ne proviennent pas de la même cellule c_0.

Il faut de plus que les signaux s_1 et s_2 « sachent » qu'ils proviennent d'un signal s_0 qui se propageait dans le quart de plan supérieur droit, ce qui est également possible.

Nous avons ainsi expliqué en détail comment le choix de la propagation du signal s_0 permettait de résoudre le problème des conflits potentiels en ce qui concerne le signal s_0 correspondant aux directions $\uparrow$ et $\leftarrow$, dans le quart de plan supérieur droit. La figure 6.12 illustre les solutions correspondantes pour toutes les autres zones du plan, et pour toute paire de directions possible à symétrie près, puisque toute paire peut être obtenue à partir des cas illustrés, par application d'une symétrie centrale ou d'une symétrie axiale d'axe $(y = x)$, le diagramme correspondant est alors obtenu à l'aide de la même transformation.

Dans chacun des schémas le plan a été séparé en un nombre fini de parties par des traits noirs gras (par exemple dans la partie en haut à gauche de la figure 6.12 correspondant aux directions $\uparrow$ et $\leftarrow$ le quart de plan supérieur droit sur lequel nous avons travaillé précédemment est représenté). Dans chacune de ces parties, les lignes noires indiquent les lignes de front correspondant à la propagation du signal s_0 (dans le cas étudié, les lignes sont bien des diagonales orthogonales à la diagonale $(y = x)$), et les pointillés gras indiquent la direction dans laquelle doivent se trouver deux cellules qui créent un conflit (dans le cas traité, ces cellules étaient à la verticale l'une de l'autre). Cette direction étant toujours une des trois directions correspondant aux projections des axes de $\mathbb{Z}^3$ dans $\mathbb{Z}^2$, elles sont toujours telles que si deux cellules de l'ensemble $t(S_k)$ créent un conflit, elles peuvent être distinguées par le signe des coordonnées du point de $\mathbb{Z}^3$ dont elles sont l'image et donc le conflit peut être évité.

Finalement, si l'on définit le *type* d'un signal s_0 comme étant la donnée de la paire de directions lui correspondant ainsi que la portion du plan dans laquelle il se propage comme indiqué sur la figure 6.12, il existe un nombre fini de types de signaux s_0 (62 au total). Si chaque signal s_1 ou s_2 émis par une cellule c_0 porte le type du signal s_0 qui est arrivé en c_0 ainsi que les signes des coordonnées de l'antécédent de c_0 par t on peut garantir que lorsque deux signaux s_2 ayant

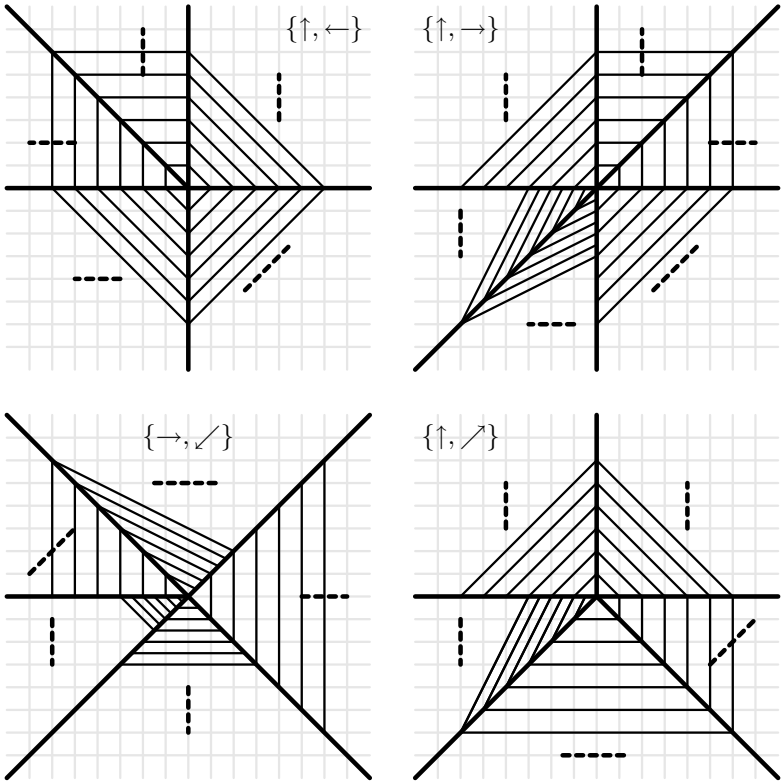

FIGURE 6.12 – La propagation du signal s_0 en fonction des directions associées.

la même information se rencontrent ils proviennent bien de la même cellule c_0 comme illustré sur la partie gauche de la figure 6.9, ce qui nous assure que les points de $t(S_{k+2})$ ayant au moins deux t-voisins inférieurs seront correctement construits.

Synchronisation

La construction inductive des points de $t(S_{k+2})$ que nous avons décrite précédemment repose sur le fait que le signal s_0 n'apparaît que lorsque les points de $t(S_k)$ et $t(S_{k+1})$ sont correctement marqués lorsque le signal s_0 apparaît. Puisqu'il est immédiat que les points de $t(S_{k+1})$ seront marqués après ceux de $t(S_k)$, nous devons juste nous assurer que le signal s_0 correspondant à l'étape k apparaît après la construction complète de $t(S_{k+1})$.

Pour obtenir ce résultat, faisons apparaître un signal en $h(k+1)$ au temps $\tau_h(k+1)$ (c'est-à-dire au moment où cette cellule est marquée comme étant dans H). Ce signal se déplace alors à vitesse maximale vers l'origine et provoque l'apparition des signaux s_0 lorsqu'il atteint l'origine. Le signal s_0 correspondant à l'étape k (celui qui servira à construire les points de $t(S_{k+1})$) apparaît donc au temps

$$\tau_{s_0}(k) = \tau_h(k+1) + h(k+1)$$

Pour montrer par induction sur k que la construction est correcte il suffit donc de montrer que les points de $t(S_{k+1})$ sont marqués avant le temps $\tau_{s_0}(k+1)$.

Les signaux s_0 tels que nous les avons décrits dans la sous-section précédente et tels qu'ils sont illustrés sur la figure 6.12, se déplacent selon chacun des axes principaux (vertical, horizontal et diagonal) à une vitesse supérieure à $1/4$ puisque le signal le plus lent est celui qui correspond aux directions $\rightarrow$ et $\swarrow$ et à la portion « supérieure gauche » du quart de plan inférieur gauche et qu'il se déplace à vitesse $1/4$ le long de la diagonale. Ainsi, tout point de $t(S_k)$ a reçu tous les signaux s_0 au temps $\tau_{s_0}(k) + 4(f(k) + g(k) + h(k))$.

Lorsqu'un point de $t(S_k)$ reçoit un signal s_0, les signaux s_1 et s_2 mettent tous deux au plus $\max(f(k+1) - f(k), g(k+1) - g(k), h(k+1) - h(k))$ étapes à atteindre leur cible, les point de $t(S_{k+2})$ sont donc correctement marqués au temps

$$\begin{aligned}
\tau_S(k+2) &\leq \tau_{s_0}(k) + 4(f(k) + g(k) + h(k)) \\
&\quad + 2f(k+1) - 2f(k) \\
&\quad + 2g(k+1) - 2g(k) \\
&\quad + 2h(k+1) - 2h(k) \\
&\leq \tau_{s_0}(k) + 2f(k+1) + 2f(k) \\
&\quad + 2g(k+1) + 2g(k) \\
&\quad + 2h(k+1) + 2h(k) \\
&\leq \tau_{s_0}(k) + 4f(k+1) + 4g(k+1) + 4h(k+1)
\end{aligned}$$

Enfin, d'après l'inégalité (6.6), on a

$$\begin{aligned}
\tau_{s_0}(k+1) &= \tau_h(k+2) + h(k+2) \\
&\geq \tau_h(k+1) + 6f(k+1) + 6g(k+1) + 7h(k+1) \\
&\geq \tau_{s_0}(k) + 6f(k+1) + 6g(k+1) + 5h(k+1) \\
&\geq \tau_S(k+2)
\end{aligned}$$

ce qui nous assure que lorsque le signal s_0 correspondant à l'étape de construction k apparaît à l'origine tous les points de $t(S_k)$ et $t(S_{k+1})$ sont correctement marqués, ce qui achève la preuve de la correction de la construction.

Par ailleurs en utilisant le résultat (6.7) et à l'aide des inégalités précédentes nous avons montré que tous les points de S_{k+1} étaient marqués au temps

$$\begin{array}{rcl} \tau_S(k+2) & \leq & \tau_{s_0}(k) + 4f(k+1) + 4g(k+1) + 4h(k+1) \\ & \leq & \tau_h(k+1) + O(k^3) \\ & \leq & O(k^4) \end{array}$$

ce qui conclut la preuve du théorème 6.3.1.

6.4 La simulation de $\mathcal{A}_3$ par $\mathcal{A}_2$

Le théorème 6.3.1 que nous venons de montrer montre que l'on peut construire l'image de $\mathbb{Z}^3$ par t en temps polynomial. Revenons maintenant à la simulation d'un automate $\mathcal{A}_3$ fonctionnant sur le voisinage de Von Neumann en dimension 3 par un automate $\mathcal{A}_2$ fonctionnant sur le voisinage de Moore en dimension 2.

Comme nous l'avons expliqué dans la section 6.1 pour tout $c \in \mathbb{Z}^3$ c'est la cellule $t(c) \in \mathbb{Z}^2$ qui doit calculer les états successifs de c. Pour cela, elle doit recevoir les états que les cellules $\{t(c+v) \mid v \in V_3\}$ (ses t-voisines) ont calculés pour appliquer la règle de transition de $\mathcal{A}_3$, puis envoyer le nouvel état qu'elle a calculé à ses t-voisines pour qu'elles puissent effectuer à leur tour leur calcul, et ainsi de suite.

La fonction de projection t a été construite pour que les t-voisines de c se trouvent dans des directions bien définies par rapport à c de telle sorte qu'il soit facile d'échanger des informations par envoi de signaux. Observons maintenant en détail comment la simulation se déroule. Il y a deux cas à considérer selon que la configuration initiale est finie ou infinie.

6.4.1 Configurations infinies avec entrée parallèle

Supposons ici que l'on donne intégralement la configuration initiale $\mathfrak{C}_0$ de $\mathcal{A}_3$ à $\mathcal{A}_2$ c'est-à-dire que pour chaque cellule $c \in \mathbb{Z}^3$, la cellule $t(c) \in \mathbb{Z}^2$ reçoit l'état $\mathfrak{C}_0(c)$ et toutes les autres cellules de $\mathbb{Z}^2$ reçoivent un état quiescent n'appartenant pas à Q_3.

Dans ce cas il n'est pas nécessaire d'utiliser la construction de $t(\mathbb{Z}^3)$ par un automate cellulaire que l'on a décrite précédemment puisque l'ensemble $t(\mathbb{Z}^3)$ est déjà construit dans la configuration initiale de l'automate.

On considère ensuite que chaque cellule de $\mathcal{A}_2$ peut être de l'une des trois couleurs *blanc*, *rouge* ou *noir* (on augmente donc le nombre d'états pour que chaque cellule connaisse sa couleur). Initialement toute cellule de $t(\mathbb{Z}^3)$ est rouge, les autres cellules étant blanches (les cellules blanches ne changeront jamais de couleur). L'automate fonctionne ensuite ainsi :

- lorsqu'une cellule de $t(\mathbb{Z}^3)$ est rouge, elle émet un signal portant l'état de la cellule de $\mathcal{A}_3$ qu'elle simule vers chacune de ses t-voisines (c'est-à-dire dans les directions $\uparrow$, $\nearrow$, $\rightarrow$, $\downarrow$, $\swarrow$ et $\leftarrow$). En particulier, au temps 0 chacune de ces cellules émet l'état qu'elle a reçu initialement. Après avoir émis ses signaux la cellule devient noire ;

- lorsqu'une cellule de $t(\mathbb{Z}^3)$ reçoit un signal portant un état d'une de ses t-voisines elle le mémorise et continue à attendre les autres signaux. Lorsqu'elle a reçu les 6 signaux qu'elle attend elle est capable d'appliquer la fonction de transition de $\mathcal{A}_3$ et donc de calculer un nouvel état de sa simulation. Lorsque le nouvel état est calculé, la cellule redevient rouge.

Remarque. Les couleurs (blanc, rouge et noir) que nous avons introduites ne servent ici qu'à indiquer aux cellules voisines d'une cellule rouge quand les signaux sont émis (pour que les voisines prennent l'état correspondant et propagent ainsi le signal). Dans ce cas-ci les couleurs n'ont pas d'autre intérêt mais elles auront plus de sens dans le cas correspondant aux configurations finies.

Notons que le fonctionnement de $\mathcal{A}_2$ que nous avons décrit ici assure que le décalage de simulation entre une cellule de $t(\mathbb{Z}^3)$ et l'une de ses t-voisines ne peut pas dépasser 1, c'est-à-dire qu'en notant $\langle c\rangle_t$ l'état de la cellule $c \in \mathbb{Z}^3$ au temps t dans le fonctionnement de $\mathcal{A}_3$, si à un moment donné la cellule $t(c)$ a calculé l'état $\langle c\rangle_t$ alors le dernier état calculé par sa t-voisine $t(c+v), v \in V_3$ est $\langle c+v\rangle_{t-1}$, $\langle c+v\rangle_t$ ou $\langle c+v\rangle_{t+1}$. Cette propriété est due au fait que pour calculer l'état $\langle c\rangle_{t+1}$ la cellule $t(c)$ doit attendre de recevoir un signal portant l'état $\langle c+v\rangle_t$ provenant de sa t-voisine $(c+v)$, ce signal n'étant bien sûr émis qu'une fois que $(c+v)$ a calculé l'état $\langle c+v\rangle_t$. Ainsi, par une relation de causalité on assure le décalage de simulation entre deux t-voisines de $\mathcal{A}_2$ est d'au plus 1 à tout instant. C'est cette propriété qui nous permet alors de justifier qu'une cellule $t(c)$ ne reçoit jamais trop de signaux de la part de ses t-voisines puisqu'elle est capable de traiter les 6 signaux qu'elle reçoit avant que de nouveaux signaux arrivent (et donc la construction décrite ne nécessite qu'un nombre fini d'états).

L'inconvénient de cette méthode est qu'elle nécessite que la totalité de l'ensemble $t(\mathbb{Z}^3)$ soit calculé pour entrer la configuration initiale de l'automate $\mathcal{A}_2$. Cependant puisque l'on manipule des configurations initiales, il est raisonnable d'envisager que l'écriture de la configuration initiale demande une quantité de calcul infinie (mais la fonction t est bien calculable et ce en temps polynomial).

Nous verrons plus tard comment on peut envisager une entrée séquentielle pour la configuration initiale qui ne requiert alors pas de calcul préalable.

6.4.2 Configurations finies

Il est naturel de demander à ce que les configurations finies de $\mathcal{A}_3$ soient transformées en des configurations finies de $\mathcal{A}_2$ avant d'effectuer la simulation, de telle sorte qu'il faille effectuer un nombre fini d'opérations pour « écrire » la configuration initiale de $\mathcal{A}_2$.

Étant donnée une configuration finie $\mathfrak{C}$ de $\mathcal{A}_3$, on calcule alors l'image $t(c)$ de chacune des cellules qui ne sont pas dans l'état quiescent dans la configuration $\mathfrak{C}$ (il n'y en a qu'un nombre fini) et l'on donne à chacune de ces cellules l'état $\mathfrak{C}(c)$ qui correspond. Toutes les autres cellules de $\mathcal{A}_2$ sont alors dans un état quiescent (qui n'est pas le même que celui de $\mathcal{A}_3$ afin d'éviter les confusions). On marque également l'origine d'un état particulier qui permet alors d'amorcer la construction de $t(\mathbb{Z}^3)$ que nous avons décrite précédemment.

Les cellules de $\mathcal{A}_2$ peuvent maintenant être de l'une des quatre couleurs *blanc*, *gris*, *rouge* ou *noir*. Initialement toutes les cellules sont blanches, exceptée l'origine qui est grise.

À partir de cet instant (et donc sur une configuration initiale finie), l'automate fonctionne de la manière suivante :

- l'origine initialise la construction de $t(\mathbb{Z}^3)$ telle qu'elle a été expliquée précédemment. Lorsqu'une nouvelle cellule de $t(\mathbb{Z}^3)$ est marquée, elle devient grise et émet des signaux vers chacune de ses t-voisines inférieures pour leur indiquer qu'elle a été marquée. Les cellules de $t(\mathbb{Z}^3)$ qui portaient un état de $\mathcal{A}_3$ depuis la configuration initiale conservent cet état lorsqu'elles deviennent grises (c'est alors leur premier état de $\mathcal{A}_3$ simulé. Si une cellule initialement quiescente devient grise, elle prend l'état quiescent de l'automate $\mathcal{A}_3$;
- lorsqu'une cellule grise a reçu un signal de chacune de ses t-voisines supérieures cela signifie que toutes ses t-voisines ont été marquées, et que les signaux qu'elle leur enverra ne seront donc pas perdus. La cellule devient alors rouge ;
- une cellule rouge fonctionne comme précédemment (dans le cas des configurations finies) c'est-à-dire qu'elle envoie son état à toutes ses t-voisines et devient noire ;
- une cellule noire attend de recevoir les états de chacune de ses t-voisines pour appliquer la règle de transition de $\mathcal{A}_3$, lorsqu'elle calcule son nouvel état elle devient rouge.
- il est possible qu'une cellule grise reçoive des états provenant de ses t-voisines inférieures avant de devenir noire. Dans ce cas elle mémorise ces états mais n'effectue aucun calcul (puisqu'elle n'a pas encore reçu tous les états de ses t-voisines.

L'avantage de ce fonctionnement est que l'automate construit lui-même l'ensemble $t(\mathbb{Z}^3)$ au cours du calcul, ce qui permet bien de partir d'une configuration initiale finie. Ici encore on peut montrer que le décalage de simulation entre deux cellules t-voisines de $t(\mathbb{Z}^3)$ est d'au plus une génération. Les t-voisines d'une cellule grise sont bloquées dans leur fonctionnement puisqu'elles doivent attendre que la cellule devienne noire pour qu'elle leur envoie son état.

6.4.3 Configurations infinies avec entrée séquentielle

Nous avons vu qu'il était possible de simuler le fonctionnement de $\mathcal{A}_3$ sur des configurations infinies mais il fallait alors donner toute la configuration initiale au temps 0, ce qui nécessite d'avoir calculé la totalité de $t(\mathbb{Z}^3)$ au préalable. Nous allons ici donner une autre façon de simuler le comportement de $\mathcal{A}_3$ sur une configuration infinie en donnant cette fois-ci la configuration initiale de $\mathcal{A}_3$ de manière séquentielle (un état à la fois). De plus cette méthode sera plus proche de la méthode employée pour les configurations finies et elle permettra donc de ne pas avoir à construire $t(\mathbb{Z}^3)$ puisque c'est l'automate $\mathcal{A}_2$ qui s'en chargera au cours de la simulation.

L'énumération de $\mathbb{Z}^3$

Considérons une méthode permettant à une machine simple de parcourir l'espace $\mathbb{Z}^3$ en se déplaçant selon le voisinage de Von Neumann et de manière à ne passer

qu'un nombre de fois borné en chaque point. Il existe par exemple de nombreux algorithmes permettant à un automate à galets de parcourir tout $\mathbb{Z}^3$ en ne passant jamais plus qu'un nombre borné α de fois en chaque point.

On supposera de plus, pour simplifier la construction à venir, que l'automate parcourt la sphère de rayon 0 puis la sphère de rayon 1 et ainsi de suite (où l'on considère comme précédemment les sphères au sens de la norme 1, la sphère de rayon r étant $S_r = \{(x,y,z) \mid \|(x,y,z)\| = |x|+|y|+|z| = r\}$). Un tel algorithme définit alors une bijection ϕ de $\mathbb{N}$ dans $\mathbb{Z}^3$ où $\phi(n)$ est la n-ième nouvelle cellule visitée par l'automate. On a par hypothèse

$$\begin{array}{rrcl} \forall p, p' \in \mathbb{Z}^3, & \|p\| < \|p'\| & \Rightarrow & \phi^{-1}(p) < \phi^{-1}(p') \\ \forall p \in \mathbb{Z}^3, & \phi^{-1}(p) & \leq & (2\|p\|+1)^3 = O(\|p\|^3) \end{array}$$

et puisque l'automate ne passe jamais plus de α fois par une même cellule la cellule p est visitée au plus tard au temps $\alpha\phi^{-1}(p)$.

De plus un automate cellulaire de dimension 3 fonctionnant sur le voisinage de Von Neumann est facilement capable de reconstruire en temps réel le parcours de l'automate à galets (au temps t l'automate a construit la totalité du parcours de l'automate entre les temps 0 et t).

Placer les états de la configuration initiale

Revenons à notre automate cellulaire $\mathcal{A}_2$ en dimension 2 qui simule le fonctionnement d'un automate $\mathcal{A}_3$ de dimension 3. Ce que l'on veut faire maintenant c'est laisser l'automate construire $t(\mathbb{Z}^3)$ comme nous l'avons déjà décrit, puis donner un par un les états de la configuration initiale de $\mathcal{A}_3$ à l'origine de $\mathcal{A}_2$. L'automate déplace ensuite les informations reçues sur l'origine vers les cellules de $t(\mathbb{Z}^3)$ correspondantes.

Lors de la construction de $t(\mathbb{Z}^3)$ l'origine reçoit un signal lorsque toute l'image de la sphère de rayon r est construite (c'est le signal qui fait partir les signaux s_0). On peut alors donner à l'automate tous les états se trouvant initialement sur la sphère S_r de $\mathcal{A}_3$ dans l'ordre correspondant à la bijection ϕ de telle sorte que l'automate $\mathcal{A}_2$ soit capable de transmettre les états aux cellules concernées en suivant le parcours de l'automate à galets parcourant $\mathbb{Z}^3$ (cette fois-ci il faut suivre le parcours après transformation par t ce qui est facile à faire puisque chacune des directions du voisinage de Von Neumann en dimension 3 correspond à une direction du voisinage de Moore en dimension 2 et que toutes les images des cellules par lesquelles l'automate à galets passe ont été marquées).

Les états initiaux de $\mathcal{A}_3$ peuvent donc être facilement transmis aux cellules de $\mathcal{A}_2$ correspondantes.

La simulation

Pour effectuer la simulation, on utilise le marquage des cellules de $t(\mathbb{Z}^3)$ par des couleurs comme dans le cas de la simulation sur les configurations initiales. Cette fois-ci une cellule $t(c) \in \mathbb{Z}^2$ ne devient grise que lorsqu'elle a été marquée et qu'elle a reçu l'état de la cellule c dans la configuration initiale de $\mathcal{A}_3$. La simulation se déroule alors exactement comme dans le cas décrit pour les configurations finies.

Cette méthode évite donc d'avoir à calculer $t(\mathbb{Z}^3)$ au préalable et permet de donner les états de la configuration initiale de l'automate un par un. La simulation se déroule alors autour de l'origine au fur et à mesure que de nouveaux états sont donnés à l'automate $\mathcal{A}_2$. Le codage par couleurs nous permet ici encore de prouver que deux cellules t-voisines n'ont jamais plus d'un temps simulé d'écart et donc d'assurer le bon déroulement de la simulation.

6.5 Vitesse de simulation

Il ne reste plus maintenant qu'à estimer le temps mis par l'automate $\mathcal{A}_2$ à simuler l'automate $\mathcal{A}_3$. Dans le cas d'une simulation sur les configurations finies telle qu'elle a été décrite dans la sous-section 6.4.2 ou sur les configurations infinies avec entrée séquentielle, le facteur limitant est l'activation des cellules de $t(\mathbb{Z}^3)$ (dans le cas des configurations initiales c'est le temps qu'il faut pour les construire et dans le cas des configurations infinies c'est le placement des états de la configuration initiale sur la bonne cellule). Ensuite la construction est telle que la simulation a lieu « aussi vite que possible » c'est-à-dire que chaque cellule effectue son calcul dès que toutes les informations nécessaires lui sont parvenues puis qu'elle envoie alors le nouvel état calculé immédiatement, en ligne droite vers ses t-voisines.

Cela signifie que pour une cellule $c \in \mathbb{Z}^3$ le temps qu'il faut à la cellule $t(c)$ pour calculer les k premiers états de c dans l'évolution de $\mathcal{A}_3$ est exactement le temps qu'il faut pour que tout le voisinage $V^k(c)$ soit marqué et actif (noir) puis que l'information portée par la plus éloignée des cellules de $V^k(c)$ atteigne c.

À $c \in \mathbb{Z}^3$ fixé nous avons vu que l'ensemble $t(V^k(c))$ était construit en temps $O(k^4)$ (théorème 6.3.1). Dans le cas des configurations infinies avec entrée séquentielle, les états se trouvant initialement sur le voisinage $V^k(c)$ sont donnés à l'origine au temps $O(k^4)$ et atteignent leur destination $O(k^3)$ générations plus tard puisque la distance à parcourir est cubique (équations 6.3). Par ailleurs le rayon de l'ensemble $V^k(c)$ est cubique ce qui signifie que l'état porté par n'importe quelle cellule de $t(V^k(c))$ atteint la cellule c en temps $O(k^3)$ même s'il ne se déplace que selon les six directions correspondant aux six vecteurs du voisinage de Von Neumann en 3 dimensions. La cellule $t(c)$ est donc capable de calculer les k premiers états de la cellule c dans l'évolution de $\mathcal{A}_3$ en $O(k^4)$ générations. La simulation de $\mathcal{A}_3$ par $\mathcal{A}_2$ est donc réalisée en temps polynomial de degré 4.

Remarque. S'il est vrai que toute cellule de $t(\mathbb{Z}^3)$ calcule k états de $\mathcal{A}_3$ en temps $O(k^4)$ il faut cependant noter que la borne polynomiale n'est pas la même pour toutes les cellules de $t(\mathbb{Z}^3)$. En effet, plus une cellule est éloignée de l'origine et plus elle mettra de temps à calculer les premiers états de la simulation (la borne en $O(k^4)$ n'a donc de sens qu'à c fixée). Toutefois le fait que deux cellules t-voisines n'aient jamais plus d'un temps simulé d'écart nous assure que pour une cellule c telle que $\|c\| = r$, le temps nécessaire pour que $t(c)$ ait simulé k états est inférieur au temps qu'il faut à l'origine pour simuler $(k + r)$ états ce qui donne une majoration de la croissance des polynômes majorant les temps de calcul d'une cellule c au fur et à mesure que l'on s'éloigne de l'origine par rapport au temps que met l'origine à simuler k générations.

6.6 Conclusion

Le travail que nous avons réalisé ici peut également être utilisé pour simuler des automates cellulaires de dimension $n \geq 3$ quelconque à l'aide d'automates cellulaires en dimension 2. Cependant il n'est pas possible d'utiliser cette méthode pour simuler des automates cellulaires bidimensionnels à l'aide d'automates unidimensionnels car il n'est pas possible de plonger le voisinage de Von Neumann de dimension 2 dans $\mathbb{Z}$ sans que les vecteurs soient colinéaires.

De nombreux points pourraient encore être approfondis concernant cette simulation. Ainsi, les fonctions f, g et h utilisées pour définir la transformation $t : \mathbb{Z}^3 \to \mathbb{Z}^2$ sont bornées par des polynômes de degré 3. En observant la croissance des sphères dans $\mathbb{Z}^3$ et $\mathbb{Z}^2$ on peut aisément montrer qu'il n'existe pas de solution bornée par des fonctions polynomiales de degré inférieur à 3/2. Une solution de degré exactement 3/2 est très peu vraisemblable car elle ne laisse pas de place pour déplacer les informations (dans notre construction la plupart des cellules ne servent qu'à transmettre les signaux ce qui nous permet de n'utiliser que des communications en ligne droite entre deux cellules t-voisines), toutefois on peut se demander s'il n'est pas possible de trouver d'autres fonctions plus « compactes » pour réaliser la transformation t. De plus, bien que leur définition soit naturelle et assez simple à exprimer, les trois fonctions f, g et h que nous utilisons sont difficiles à calculer et cette difficulté de calcul augmente le temps de la simulation (le degré 4 du temps de simulation ne provient que du temps mis à construire $t(\mathbb{Z}^3)$, toutes les autres étapes sont réalisées en temps cubique). Une étude plus approfondie pourrait aboutir à une expression plus simple (une formule close par exemple) qui pourrait peut-être permettre une construction directe et plus rapide (en temps cubique).

Enfin, la transformation t nous permet de représenter l'espace discret $\mathbb{Z}^3$ dans le plan discret $\mathbb{Z}^2$ tout en conservant assez bien la notion de voisins. Il serait intéressant de voir quelles sont les images d'objets tridimensionnels simples et bien connus par t (des polyèdres, des surfaces, etc.) ou encore de savoir si des propriétés géométriques des configurations d'un automate en dimension 3 peuvent facilement être identifiées en observant leur image par t.

Chapitre 7

Conclusion et perspectives

> "And would, if you could turn that mighty clock back to that long, fateful night, now think carefully Jack, would you do the whole thing all over again, knowing what you know now, knowing what you knew then ?" And he smiled, like the old Pumpkin King that I knew, then turned and asked softly of me, "Wouldn't you ?"
>
> **Tim Burton** – *The Nightmare Before Christmas*

Dans ce dernier chapitre nous reviendrons sur le travail effectué (principalement dans les chapitres 2, 3 et 4) et présenterons certaines questions restées ouvertes qui nous paraissent les plus intéressantes à développer par la suite.

Les voisinages et le temps réel

L'objectif de cette thèse était de comprendre l'influence que pouvait avoir le choix du voisinage sur les capacités algorithmiques d'un automate cellulaire.

Puisque des études précédentes avaient déjà montré que le voisinage n'avait pas d'incidence si l'on s'intéresse à des classes de complexité supérieure ou égale au temps linéaire, nous avons choisi de nous concentrer sur les classes de très faible complexité et tout particulièrement le temps réel.

Le temps réel est une notion très riche dans le cadre des automates cellulaires, n'ayant pas de véritable équivalent dans le cas Turing, qui est très fortement liée à l'idée de voisinage. La complexité en temps réel présente en effet la caractéristique très exceptionnelle d'être « équitable » envers les différents voisinages.

Ainsi si l'on considère deux voisinages V et V' tels que V' soit plus petit que V (au sens de l'inclusion ou du rayon) il est immédiat qu'un automate cellulaire fonctionnant sur le voisinage V aura plus de « puissance » en termes de calcul qu'un automate fonctionnant sur V' car d'une part les informations peuvent se déplacer plus rapidement (importance du rayon) et d'autre part l'automate est capable de manipuler plus d'informations en une unique étape de temps (importance du cardinal du voisinage).

Toutefois le temps réel est une fonction de temps qui correspond à une idée naturelle et intuitive, qui prend en compte la forme du voisinage et permet ainsi

de compenser les désavantages évidents des petits voisinages en leur accordant plus de temps de calcul. Ainsi, la comparaison des différents voisinages au sens du temps réel permet de s'affranchir des contraintes de cardinal et d'envergure pour donner toute son importance à la structure que le voisinage V apporte à l'espace $\mathbb{Z}^d$.

En dimension 1

Nous avons tout naturellement commencé notre étude dans le cadre de la dimension 1. Très peu de voisinages unidimensionnels avaient été étudiés préalablement et la quasi-totalité des résultats connus ne concernaient que le voisinage standard $\{-1, 0, 1\}$ et sa version très affaiblie qu'est le voisinage one-way $\{0, 1\}$.

En travaillant sur des voisinages qui n'étaient pas nécessairement connexes nous avons réussi à montrer un résultat d'équivalence qui permet de ramener le cas de tout voisinage assez complet pour pouvoir effectuer correctement de la reconnaissance de langages à celui du voisinage standard ou du voisinage one-way.

Ce résultat est à la fois très satisfaisant et décevant. Sa très grande portée en fait un résultat très important mais donne également l'impression de rendre obsolètes toutes les questions que l'on pouvait se poser sur les différents voisinages en dimension 1. Il donne l'impression non seulement d'avoir répondu à la question initialement posée mais également d'avoir clos cette problématique trop tôt sans soulever de nouvelles questions comme c'est habituellement le cas en sciences.

Bien évidemment il reste toujours beaucoup à faire en ce qui concerne la reconnaissance de langages par des automates cellulaires en dimension 1, mais il semblerait que cette étude n'ait pas de raison d'être faite sur des voisinages autres que les deux voisinages déjà bien étudiés.

Signalons toutefois que l'équivalence que l'on a obtenue ne concerne que les voisinages complets et que peu de choses sont connues concernant les voisinages incomplets. Comme nous l'avons signalé en conclusion du chapitre 1 nous savons que deux voisinage qui ne permettent pas d'atteindre la même partie de $\mathbb{Z}$ ne sont pas équivalents, mais la réciproque est inconnue :

> **Question :** *En dimension* 1*, étant donnés deux voisinages* V_1 *et* V_2 *tels que* $V_1^\infty = V_2^\infty$*, est-il toujours vrai que tout langage reconnu en temps réel sur* V_1 *l'est également sur* V_2 *?*

On conjecture que cette implication est vraie et donc que la classe des langages reconnus en temps réel sur un voisinage V ne dépend que de la couverture V^∞ de $\mathbb{Z}$ par V.

L'enveloppe convexe en dimension 2

Après avoir répondu à notre question initiale en dimension 1 nous nous sommes intéressés aux dimensions supérieures.

L'ensemble des résultats que l'on a alors obtenus fait très nettement ressortir une caractéristique fondamentale des voisinages : leur enveloppe convexe.

En effet, la grande majorité des propriétés d'un voisinage en dimension 2 vis-à-vis de sa puissance de calcul en temps réel semble être déterminée par la

forme de son enveloppe convexe. La proposition 2.3.1 qui indique qu'un automate fonctionnant sur un voisinage V peut fonctionner exactement comme un automate fonctionnant sur l'enveloppe convexe de V (tant que l'on ne passe pas en dessous du temps réel) est donc capitale.

Cette proposition est d'ailleurs très surprenante car nous n'avons pas réussi à obtenir de réciproque. D'une certaine manière cela signifie donc que tout ce qui peut être réalisé en temps réel sur l'enveloppe convexe d'un voisinage V peut également être réalisé en temps réel sur V (alors que V est « plus petit » que son enveloppe convexe) mais qu'il est possible que certains calculs soient réalisables sur V en temps réel alors qu'ils ne le seraient pas sur $\mathrm{CH}(V)$ (ce qui met particulièrement en évidence la « compensation » que le temps réel apporte aux petits voisinages).

Question : *Existe-t-il un voisinage complet V et un langage L en dimension 2 tels que L soit reconnaissable en temps réel par un automate cellulaire fonctionnant sur V mais par aucun automate cellulaire fonctionnant sur $\mathrm{CH}(V)$?*

Une réponse affirmative à cette question permettrait d'obtenir des résultats d'accélération constante totale pour de nombreux voisinages à l'aide de la proposition 2.3.1 comme il a déjà été remarqué.

Le fait que l'enveloppe convexe d'un voisinage ait autant d'impact sur les possibilités algorithmiques de celui-ci explique en partie pourquoi les voisinages en dimension 1 présentent si peu de diversité puisque les enveloppes convexes en dimension 1 sont réduites à des segments (la différence entre les voisinages one-way et les autres est d'ailleurs également une propriété de l'enveloppe convexe).

Il n'est pas exclu que l'on puisse obtenir une équivalence des voisinages en dimension 2 et au-delà qui soit une généralisation naturelle du théorème 2.2.3 et qui sépare les voisinages en fonction de la forme de leur enveloppe convexe.

On conjecture ainsi l'équivalence très forte suivante de laquelle nous sommes encore loin, bien que plusieurs des propositions présentées dans ce texte semblent nous en rapprocher significativement :

Conjecture : *Étant donnés deux voisinages complets V et V' en dimension 2*

- *si les enveloppes convexes de V et V' sont semblables sur le quart de plan positif (il existe k et k' tels que $\mathrm{CH}(V)^k \cap \mathbb{N}^2 = \mathrm{CH}(V')^{k'} \cap \mathbb{N}^2$) alors V et V' permettent de reconnaître les mêmes langages en temps réel ;*
- *sinon, il existe un langage reconnu en temps réel sur V qui ne l'est pas sur V' (et réciproquement).*

Von Neumann, Moore et les autres

En dimension 2, deux voisinages ont été particulièrement étudiés : le voisinage de Moore (la cellule et ses 8 plus proches voisines) et celui de Von Neumann (la cellule et ses 4 plus proches voisines).

Bien que tous les deux présentent la même simplicité apparente (convexité, petit rayon, symétrie) ils s'avèrent être très différents. Tandis que le voisinage de Moore permet de généraliser sans effort les résultats principaux connus en

dimension 1 sur le voisinage standard tels que l'accélération par une constante ou l'accélération linéaire il faut, dans le cas du voisinage de Von Neumann, utiliser des techniques très différentes.

La question suivante n'a toujours pas de reponse satisfaisante :

> **Question :** *(très informelle) Quelle est la différence fondamentale entre les voisinages de Moore et de Von Neumann qui affecte si fortement les capacités algorithmiques ?*

Remarquons par ailleurs que l'on sait qu'il existe des langages reconnus en temps réel sur le voisinage de Von Neumann qui ne le sont pas sur le voisinage de Moore mais que l'on ignore si l'inverse est vrai également.

Notre travail sur l'accélération linéaire s'est également heurté à une hypothèse inattendue et très contraignante sur la forme de l'enveloppe convexe du voisinage dans le quart de plan positif... Même en considérant des voisinages apparemment très simples, symétriques et convexes, l'accélération linéaire ne semble pas réalisable telle qu'elle est obtenue habituellement.

> **Question :** *Le voisinage en octogone (voir figure 4.10) permet-il une accélération linéaire ?*

Pour conclure la conclusion...

Les travaux effectués au cours de cette thèse, les différentes avancées et les diverses questions ouvertes nouvelles ou anciennes nous ont permis de mettre en évidence une problématique très riche centrée sur les concepts de voisinage et de temps réel qui sont fondamentaux dans le cas des automates cellulaires et n'ont pas d'équivalent sur les autres modèles de calculs.

Modifier le voisinage c'est modifier l'espace de travail car s'il est vrai que l'ensemble des cellules reste le même, sa structure peut être radicalement altérée (les distances et les géodésiques par exemple). Cela peut alors perturber considérablement les possibilités algorithmiques des automates cellulaires considérés.

En imposant un voisinage aux automates que l'on manipule on peut alors obtenir des modèles de calcul pour lesquels il est très difficile (peut-être même impossible) d'obtenir des résultats aussi simples et fondamentaux que des théorèmes d'accélération par une constante ou par un facteur multiplicatif.

Mieux connaître le fonctionnement de tels objets (pour des voisinages divers) pourrait alors permettre de mieux comprendre ce qu'apporte la structure de l'espace sur les modèles déjà bien étudiés (les modèles tels que les machines de Turing ou les automates cellulaires fonctionnant sur le voisinage de Moore ou Von Neumann par exemple) et de séparer les capacités inhérentes au modèle (dans le cas des automates cellulaires c'est ce que l'on peut réaliser sur tout voisinage) des possibilités offertes par la structure de l'espace.

Bibliographie

[1] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for your Mathematical Plays*, volume 2. Academic Press, ISBN 0-12-091152-3, 1982. chapter 25.

[2] C. Choffrut and K. Čulik II. On real-time cellular automata and trellis automata. *Acta Informatica*, 21 :393–407, 1984.

[3] S. N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machines. *IEEE Transactions on Computers*, C-18(4) :349–365, 1969.

[4] C. Čulik II, J. Gruska, and A. Salomaa. Systolic treillis automaton (for VLSI). Res. Rep CS-81-34, Dept of Computer Science, Univ. of Waterloo, 1981.

[5] K. Čulik II and S. Yu. Undecidability of CA classification schemes. *Complex Syst.*, 2(2) :177–190, 1988.

[6] M. Delorme, J. Mazoyer, and L. Tougne. Discrete parabolas and circles on 2d cellular automata. *Theoritical Computer Science*, 218(2) :347–417, 1999.

[7] C. R. Dyer. One-way bounded cellular automata. *Information and Control*, 44(3) :261–281, 1980.

[8] P. C. Fischer. Generation of primes by one-dimensional real-time iterative array. *Journal of the Assoc. Comput. Mach.*, 12 :388–394, 1965.

[9] G. A. Hedlund. Endomorphisms and Automorphisms of the Shift Dynamical Systems. *Mathematical Systems Theory*, 3(4) :320–375, 1969.

[10] O. Ibarra. *Cellular Automata : a Parallel Model*, chapter 6, pages 181–197. Kluwer, Dordrecht, mathematics and its applications edition, 1999.

[11] C. Iwamoto, T. Hatsuyama, K. Morita, and K. Imai. On time-constructible functions in one-dimensional cellular automata. In *FCT*, pages 316–326, 1999.

[12] S. La Torre, M. Napoli, and D. Parente. Synchronization of a line of identical processors at a given time. *Fundamenta Informaticae*, 34(1-2) :103–128, 1998.

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

[14] M. S. Paterson. Tape bounds for time bounded Turing machines. *JCSS*, 6 :116–124, 1972.

[15] Z. Róka. *Automates cellulaires sur les graphes de Cayley.* PhD thesis, Universit Lyon I et École Normale Sup rieure de Lyon, 1994.

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

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

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

Abstract: In this thesis we have worked on the impact of the choice of a neighborhood on the algorithmic abilities of cellular automata. We have specifically studied the lower complexity classes such as the real time (that corresponds to the shortest time necessary for a cellular automaton to read all the letters of the input word) and the real time plus a constant. It is indeed known that neighborhoods are equivalent in linear time and it is therefore necessary to consider shorter times.

We have obtained neighborhood equivalence results with respect to the real time (neighborhood classes such that cellular automata working on any of those neighborhoods can recognize the same languages in real time) and linear or constant speed-up theorems for many classes of neighborhoods.
Keywords: Cellular automata, neighborhoods, language recognition, any dimension, real time, convex hull, linear speed-up, constant speed-up.

Résumé : Dans cette thèse nous nous sommes intéressés à l'importance du choix du voisinage sur les capacités algorithmiques des automates cellulaires. Nous avons travaillé en dimension quelconque en nous concentrant sur les classes de complexité correspondant au temps réel (plus petit temps nécessaire pour que l'automate ait lu le mot en entrée) et temps réel plus une constante. En effet il est connu que les voisinages sont équivalents en temps linéaire et il est donc nécessaire de considérer des temps inférieurs.

Nous avons obtenu plusieurs résultats d'équivalences de voisinages au sens du temps réel (des classes de voisinages tels que les automates fonctionnant sur ces voisinages reconnaissent les mêmes langages) et des résultats d'accélérations linéaires ou constantes selon les voisinages.
Mots-clés : Automates cellulaires, voisinages, reconnaissance de langages, dimension quelconque, temps réel, enveloppe convexe, accélération constante, accélération linéaire.