Rapport de Stage
Étude des automates cellulaires inversibles

Jean-Christophe Dubacq *
dirigé par Bruno Durand
et Jacques Mazoyer.

Stage effectué du 15 juin au 31 juillet 1993
au Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon, U.R.A. C.N.R.S. n° 1398.

Table des matières

*Élève de première année du magistère d'Informatique de l'École Normale Supérieure de Lyon et de l'Université Claude Bernard (Lyon I).

1 Introduction

Le travail effectué au Laboratoire de l'Informatique du Parallélisme a porté sur l'étude des *automates cellulaires*, sous une forme théorique, et en particulier l'étude de la classe des automates cellulaires *inversibles*. Cette étude aborde des aspects importants et diversifiés de ce sujet.

Dans une première partie de mon travail, j'ai donné de manière complète un algorithme permettant de résoudre le problème de décision de l'inversibilité ou de la surjectivité d'automates cellulaires de dimension 1, en temps quadratique. Cet algorithme est inspiré d'un article de K. Sutner [11].

Dans une deuxième partie, je me suis intéressé aux rapports entre automates cellulaires inversibles et universalité, et j'ai construit un automate cellulaire inversible capable de simuler des machines de Turing non réversibles, ce qui prouvait l'universalité de tels automates. Morita, dans [8], avait prouvé un résultat similaire, mais ne simulait que des machines de Turing réversibles.

Enfin, j'ai modifié un programme pré-existant qui permet de simuler des automates cellulaires plans afin d'augmenter ses fonctionnalités, en particulier en lui permettant de décrire les automates sous forme de fonction écrites en langage C.

2 Rappels théoriques

2.1 Définitions

2.1.1 Automate Cellulaire

Un *automate cellulaire* (AC) est un quadruplet $\mathcal{A} = (\lceil, \mathcal{S}, \mathcal{V}, \{)$
où :

— d désigne la dimension de l'automate ;
— S désigne l'ensemble fini des états de l'automate $\{s_1,\ s_2,\ \ldots, s_m\}$;
— V désigne son vecteur de voisinage $\{v_1,\ v_2,\ \ldots, v_n\}$;
— $f : S^n \to S$ désigne sa fonction de transition locale.

On appelle *cellule* tout élément de $\mathbb{Z}^d$. Une *configuration* de l'automate est une application $c : \mathbb{Z}^d \to S$. Soit C l'ensemble des configurations c. On a $C = S^{\mathbb{Z}^d}$. On définit alors la *fonction de transition globale*, $F : C \to C$ par :

$$\forall x \in \mathbb{Z}^d, F(c)(x) = f(c(x + v_1), c(x + v_2), \ldots, c(x + v_n)).$$

On peut parfois distinguer un état spécial $q \in S$, appelé *état quiescent*, qui ne peut être modifié que par d'autres états, c'est à dire tel que $f(q, q, \ldots, q) = q$.

Un automate cellulaire $\mathcal{A}$ est dit *injectif* si sa fonction de transition globale F est injective, *i.e.*

$$c_1 \neq c_2 \Longrightarrow F(c_1) \neq F(c_2).$$

De même, il sera dit *surjectif* si F est surjective, *i.e.*

$$\forall c \in C, \exists c' \in C \ / \ F(c') = c.$$

On définit, pour tout AC $\mathcal{A} = (\lceil, \mathcal{S}, \mathcal{V}, \{)$, son *voisinage minimal* par :

$$\{i \in \mathbb{Z}^d | \exists c_1, c_2 \in C : c_1(i) \neq c_2(i), \forall j \neq i, c_1(j) = c_2(j), F(c_1)(\overline{0}) \neq F(c_2)(\overline{0})\}$$

où $\overline{0} = (0, 0, \ldots, 0) \in \mathbb{Z}^d$. Le voisinage minimal d'un automate cellulaire correspond au vecteur de voisinage regroupant tous les voisins effectivement nécessaires à la définition cet automate.

On appelle *Jardin d'Eden* de l'automate $\mathcal{A}$ toute configuration $c \in C \setminus F(C)$. En d'autres termes, un *Jardin d'Eden* de l'automate $\mathcal{A}$ est une configuration sans antécédent, elle ne peut donc apparaître, lors de l'évolution d'un AC, que comme configuration initiale. On note JdE l'ensemble des configurations jardin d'Eden de l'automate $\mathcal{A}$.

2.1.2 Configurations finies et configurations périodiques

Une configuration sera dite *finie* si elle a seulement un nombre fini de cellules dans un état autre que l'état quiescent. Soit C_f l'ensemble de toutes les configurations finies. On définit alors la fonction F_f, restriction de F à C_f :

$$\begin{array}{ccc} & F_f & \\ C_f & \longrightarrow & C_f \\ c & \longrightarrow & F(c) \end{array}$$

On appelle *configuration périodique* toute configuration qui est périodique selon toutes les directions de l'espace $\mathbb{Z}^d$. Soit $k = (k_1, k_2, \ldots, k_d) \in \mathbb{Z}^d$, on dit qu'une configuration c est k-périodique si et seulement si pour tout $i \in \{1, \ldots, d\}$, k_i est la période de c dans la i-ème direction canonique de l'espace. On appelle C_p (resp. C_p^k) l'ensemble des configurations périodiques (resp. k-périodiques). On peut tout de suite remarquer que l'image d'une configuration k-périodique est également k-périodique (sa période peut en fait être inférieure à k, mais doit diviser k). Nous pouvons alors définir les fonctions F_p et F_p^k, restrictions de F comme suit :

$$\begin{array}{ccc} & F_p & \\ C_p & \longrightarrow & C_p \\ c & \longrightarrow & F(c) \end{array} \qquad \begin{array}{ccc} & F_p^k & \\ C_p^k & \longrightarrow & C_p^k \\ c & \longrightarrow & F(c) \end{array}$$

2.2 Automate cellulaire inversible

2.2.1 Définition

Un automate cellulaire $\mathcal{A} = (\lceil, \mathcal{S}, \mathcal{V}, \{)$ de fonction de transition globale F est dit *inversible* s'il existe un automate cellulaire $\mathcal{B} = (\lceil, \mathcal{S}, \mathcal{V}', \{')$ de fonction de transition globale F' tel que $F' = F^{-1}$.

S'il existe, un tel AC $\mathcal{B}$ simule le processus inverse de $\mathcal{A}$. $\mathcal{B}$ est alors naturellement appelé *automate inverse* de $\mathcal{A}$ et est noté $\mathcal{B} = \mathcal{A}^{-\infty}$.

On définit la fonction de *décalage* $\sigma : S^{\mathbb{Z}} \longrightarrow S^{\mathbb{Z}}$ qui décale une configuration d'une position vers la gauche dans toutes les directions :

$$\forall c \in S^{\mathbb{Z}}\ , \forall i \in S^d\ ,\ \sigma(c)(i)\ =\ c(i+1).$$

Théorème 1 (Richardson [10]) *Toute fonction continue $\phi : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$ satisfaisant $\phi \circ \sigma = \sigma \circ \phi$ est la fonction globale d'un automate cellulaire.*

Un corollaire très important de ce théorème s'énonce ainsi :

Corollaire 1 (Richardson [10]) *Un automate cellulaire est inversible si et seulement s'il est injectif.*

Ce corollaire prouve qu'il existe toujours un AC qui réalise l'inverse de la fonction globale d'un AC bijectif, et que tout AC injectif est bijectif.

2.2.2 Résultats de décidabilité

Des résultats de décidabilité très importants ont été trouvés pour les automates cellulaires, qui séparent le cas des automates à une dimension et le cas des automates de dimension supérieure.

Théorème 2 (Amoroso et Patt [1]) *Il est décidable de savoir si un automate cellulaire de dimension 1 est inversible.*

Une preuve de ce théorème est donnée dans le paragraphe 3. Ce résultat est en revanche modifié pour les automates de dimensions supérieure, et ce meme avec un "petit" voisinage de Von Neumann (centre plus 4 voisins).

Théorème 3 (Kari [3] et [4]) *Il est indécidable de savoir si un AC de dimension 2 avec un voisinage de Von Neumann est inversible.*

Cette démonstration est effectuée par des liaisons avec d'autres problèmes indécidables, relatifs aux pavages du plan. D'autres résultats ont été démontrés, comme

Théorème 4 (Kari [3] et [4]) *Il est indécidable de savoir si un AC de dimension 2 avec un voisinage de Von Neumann est surjectif.*

2.2.3 Résultats divers

Théorème du jardin d'Eden Ce théorème est considéré comme l'un des plus importants théorèmes sur les automates cellulaires. Il a été démontré en deux parties, d'abord par Moore dans [5], et la réciproque a été montrée par Myhill dans [9].

Théorème 5 (Théorème du Jardin d'Eden) *Un automate cellulaire $\mathcal{A}$ est surjectif si et seulement si F_f, la restriction de sa fonction de transition globale aux configurations finies, est injective.*

Les implications diverses peuvent être résumées par le schéma 1.

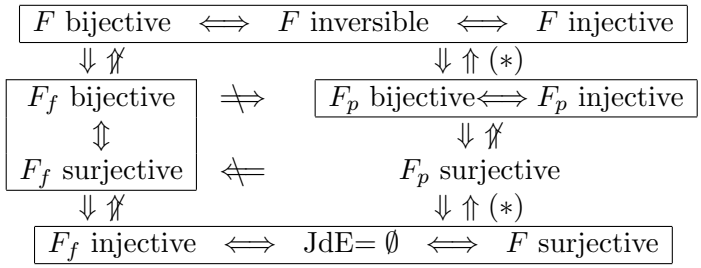

FIGURE 1 – Diagramme des implications.
Les (*) désignent des implications qui ne sont vraies qu'en dimension 1.

3 Automates cellulaires linéaires et graphes de De Bruijn

Cette partie étudie les propriétés globales des automates cellulaires en utilisant des graphes dits de De Bruijn. Il s'inspire de [11] et de [2] qui proposaient des constructions identiques. Dans [1], Amoroso et Patt ont montré qu'il était décidable de savoir si un automate cellulaire linéaire était injectif ou surjectif. On veut trouver un algorithme qui permet de décider de l'injectivité ou de la surjectivité d'un automate.

3.1 Représentation des automates cellulaires à l'aide de graphes de De Bruijn

On considère un automate à une dimension $\mathcal{A} = (1, \Sigma, \mathcal{N}, f)$ où
- Σ est un ensemble (fini) d'états ;
- $r \in \mathbb{N}$ est la taille du voisinage de l'automate ;
- $\mathcal{N} = \{k+1, k+2, \ldots, k+r\}$ est un sous-ensemble fini de $\mathbb{Z}, k \in \mathbb{Z}$ (voisinage de l'automate). On considére ici un voisinage *consécutif*;
- f est une fonction de $\Sigma^{|\mathcal{N}|} \to \Sigma$ (la fonction locale de l'automate).

On va introduire une représentation de ces automates cellulaires sous la forme d'un graphe régulier de degré $\sigma = |\Sigma|$, orienté, comportant $\sigma^{|\mathcal{N}|-1}$ sommets. Pour cela, on considère le graphe $\mathcal{G} = (\mathcal{S}, \mathcal{V})$ où :
- $\mathcal{S} = \Sigma^{|\mathcal{N}|-1} = \{\omega_0\omega_1 \ldots \omega_{|\mathcal{N}|-1}, \forall i, \omega_i \in \Sigma\}$, l'ensemble des sommets du graphe ;
- Une arête $(\omega^{(1)}, \omega^{(2)})$ de $\mathcal{S} \times \mathcal{S}$ appartient à $\mathcal{V}$ si et seulement si
  - $(\omega^{(1)} = a\omega'), a \in \Sigma$, et
  - $(\omega^{(2)} = \omega' b), b \in \Sigma$.

  On étiquette alors l'arête par $f(a\omega' b)$, qui est l'image par la fonction de l'automate $\mathcal{A}$ du mot $a\omega' b$. Un tel graphe est dit de *de Bruijn*.

Dans un graphe de ce type, une configuration peut alors être considérée comme une suite bi-infinie de sommets, de même qu'une configuration périodique est considérée comme un cycle dans le graphe. L'image de cette configuration est facile à obtenir puisqu'il s'agit de la suite des étiquettes des arêtes du graphe. Il est facile d'obtenir à partir de ce graphe un traducteur, qui réalise la transformation d'une configuration par la fonction globale de $\mathcal{A}$: il suffit en effet de parcourir le graphe selon le chemin qui représente la configuration initiale, et de noter les étiquettes des arêtes parcourues pour obtenir l'image de la configuration initiale. C'est cette simplicité de représentation qui va être importante dans la suite. Ce graphe a certaines propriétés intéressantes et utiles par la suite. En particulier, il est connexe et même *hamiltonien* (c'est à dire qu'il existe un chemin passant par tous les sommets une fois et une seule).

3.2 Construction d'un graphe produit $\mathcal{H}$

3.2.1 Définition de $\mathcal{H}$

On s'intéresse au graphe $\mathcal{H} = (\mathcal{S} \times \mathcal{S}, \mathcal{W})$, où

$$\mathcal{W} = \{((s_1, s_2), (s_3, s_4)) | s_1, s_2, s_3, s_4 \in \mathcal{S}, (s_1, s_3) \text{ et } (s_2, s_4) \text{ ont mêmes étiquettes}\}.$$

Le principe de définition de ce graphe est l'étude simultanée de deux configurations. Ce graphe va donc être utile pour étudier l'injectivité et donc l'inversibilité d'un automate cellulaire. De plus, la surjectivité pouvant être déduite de l'injectivité locale, on pourra utiliser ce graphe pour caractériser les automates cellulaires surjectifs. Ceci pourra être effectué de manière très simple en un temps borné par le nombre de sommets et d'arêtes du graphe $\mathcal{H}$, comme nous l'expliquons plus loin.

On définit les composantes fortement connexes de $\mathcal{H}$: une composante fortement connexe est un ensemble maximal pour la relation $R(x, y)$ "Il existe un chemin de x à y". Ces ensembles sont bien évidemment connexes, et du fait que l'on élimine les situations triviales, il est toujours possible de trouver un cycle dans une composante fortement connexe. On construit ensuite un graphe $\mathcal{D}$, dérivé de $\mathcal{H}$ en supprimant tous les points isolés et les points qui ne sont sur aucun chemin bi-infini.

3.2.2 Définition de $\mathcal{D}$

Lemme 1 *Les arêtes de $\mathcal{H}$ qui ne sont pas reliées à une composante fortement connexe dans les deux sens ne peuvent pas faire partie d'un chemin bi-infini.*

Preuve. Il est en effet impossible de prolonger à l'infini un chemin qui contient une telle arête K. Supposons que K appartienne à un chemin bi-infini. Le nombre de sommets de $\mathcal{H}$ étant borné, le chemin passe forcément deux fois par un même point, qui appartient donc à une composante fortement connexe. De même pour les arêtes avant K, et donc K est sur un chemin liant deux composantes fortement connexes (éventuellement confondues). □

On s'attachera donc à construire le graphe $\mathcal{D}$ qui est le graphe résultant de la suppression des points isolés et de ceux qui ne sont pas reliés dans les deux sens à des composantes fortement connexes. Cette construction peut se faire aisément en temps $O(\text{nombre d'arêtes} + \text{nombre de sommets})$. Or, dans $\mathcal{G}$, il y a exactement σ arêtes par sommet. Puisqu'une arête dans $\mathcal{H}$ correspond à l'existence d'une étiquette commune à deux arêtes de $\mathcal{G}$, un sommet de $\mathcal{H}$ a au plus σ^2 arêtes sortantes. Or, le nombre de sommets de $\mathcal{H}$ est de $\sigma^{2(|\mathcal{N}|-1)}$. il est donc possible de déduire $\mathcal{D}$ de $\mathcal{H}$ en temps $O(\sigma^{2|\mathcal{N}|})$.

On notera $\tilde{\mathcal{G}}$ la trace de $\mathcal{G}$ dans $\mathcal{D}$ (*i.e.* l'ensemble des $(\omega, \omega), \omega \in \mathcal{S}$) et $\overline{\mathcal{G}}$ la composante fortement connexe contenant $\tilde{\mathcal{G}}$. Il ressort de la construction même de $\tilde{\mathcal{G}}$ qu'il est connexe et hamiltonien.

Lemme 2 *Il est possible, à partir d'un chemin bi-infini dans $\mathcal{D}$, de construire deux configurations de $\mathcal{A}$ ayant même image.*

Preuve. Notons un chemin de $\mathcal{D}$ sous la forme $(\ldots, (\omega_{i-1}, \theta_{i-1}), (\omega_i, \theta_i), (\omega_{i+1}, \theta_{i+1}), \ldots)$ avec $\forall i \in \mathbb{Z}, \omega_i = a_i\omega'_i, \theta_i = b_i\theta'_i$. Alors les configurations $\mathcal{C}_1 = (\ldots, a_{i-1}, a_i, a_{i+1}, \ldots)$ et $\mathcal{C}_2 = (\ldots, b_{i-1}, b_i, b_{i+1}, \ldots)$ ont même image par $\mathcal{A}$. En effet, l'image de $\mathcal{C}_1$ est la suite des étiquettes du chemin $(\ldots, \omega_{i-1}, \omega_i, \omega_{i+1}, \ldots)$ dans $\mathcal{G}$. De même, l'image de $\mathcal{C}_2$ est la suite des étiquettes du chemin $(\ldots, \theta_{i-1}, \theta_i, \theta_{i+1}, \ldots)$ dans $\mathcal{G}$. Or, par définition de $\mathcal{H}$, ces étiquettes sont justement identiques. Donc, $\mathcal{C}_1$ et $\mathcal{C}_2$ ont bien même image. □

3.3 Algorithme de décision pour l'inversibilité et la surjectivité

3.3.1 Inversibilité

Théorème 6 *La bijectivité de l'automate cellulaire $\mathcal{A}$ est équivalente à $\tilde{\mathcal{G}} = \overline{\mathcal{G}}$ est la seule composante fortement connexe de $\mathcal{D}$.*

Preuve. S'il existe une autre composante fortement connexe, on peut trouver un cycle dans $\mathcal{D}$ qui formera ainsi un chemin bi-infini. D'après le Lemme 2, ce cycle aura deux traces dans $\mathcal{G}$, qui auront la même image, par application de l'automate cellulaire $\mathcal{A}$. Ces deux chemins seront distincts, puisque contenant un point qui n'est pas dans $\tilde{\mathcal{G}}$, donc ayant des traces distinctes dans $\mathcal{G}$. On a donc deux configurations distinctes qui ont une même image, ce qui contredit l'inversibilité de $\mathcal{A}$. Donc $\overline{\mathcal{G}}$ est la seule composante fortement connexe de $\mathcal{D}$. De même, s'il existe un chemin bi-infini qui n'est pas complètement inclus dans $\tilde{\mathcal{G}}$, alors il existe deux chemins distincts en au moins un point dans $\mathcal{G}$, et donc deux configurations distinctes ayant même image par $\mathcal{A}$. Donc,

$$\mathcal{A} \text{ bijectif} \Rightarrow \tilde{\mathcal{G}} = \overline{\mathcal{G}} \text{ est la seule composante fortement connexe de } \mathcal{D}$$

Réciproquement, si l'automate cellulaire $\mathcal{A}$ n'est pas injectif, il existe dans $\mathcal{G}$ deux chemins distincts ayant les mêmes étiquettes. On peut donc construire un chemin bi-infini de $\mathcal{D}$ qui ne soit pas complètement inclus dans $\tilde{\mathcal{G}}$. Or, le nombre de sommets de $\mathcal{G}$ étant fini, ce chemin repasse par un même point après un point hors de $\tilde{\mathcal{G}}$, qui fait donc partie d'une composante fortement connexe, et de même avant. Soit les deux composantes fortement connexes sus-citées sont égales à $\overline{\mathcal{G}}$, et dans ce cas $\overline{\mathcal{G}} \neq \tilde{\mathcal{G}}$, soit ce n'est pas $\overline{\mathcal{G}}$, et dans ce cas il existe une composante fortement connexe différente de $\overline{\mathcal{G}}$. L'équivalence ci-dessus est donc prouvée. □

3.3.2 Surjectivité

Théorème 7 *La surjectivité de l'automate cellulaire $\mathcal{A}$ est équivalente à $\tilde{\mathcal{G}} = \overline{\mathcal{G}}$.*

Preuve. On utilisera pour cette démonstration le Théorème du Jardin d'Eden. Supposons que $\mathcal{A}$ ne soit pas localement injective. Cela implique l'existence de deux configurations qui diffèrent sur un nombre fini de points, mais ayant la même image. Ces configurations sont donc identiques à l'infini. Par conséquent, le chemin généré par leur conjonction dans $\mathcal{D}$ est contenu à l'infini dans $\tilde{\mathcal{G}}$. De plus, il existe des points où elles sont effectivement différentes, et donc des points de $\overline{\mathcal{G}}$ qui ne sont pas dans $\tilde{\mathcal{G}}$. Donc $\overline{\mathcal{G}} \neq \tilde{\mathcal{G}}$. Réciproquement, si $\overline{\mathcal{G}} \neq \tilde{\mathcal{G}}$, il existe un chemin qui correspond à deux configurations identiques à l'infini, mais passant par un point qui est dans $\overline{\mathcal{G}}$ et pas dans $\tilde{\mathcal{G}}$. Ces deux chemins auront même image, et seront identiques presque partout. Donc $\mathcal{A}$ ne sera pas localement injectif, donc pas surjectif. □

3.3.3 Algorithmes de décision

On peut déduire des deux théorèmes qui précèdent un algorithme pour déterminer en temps $O(\sigma^{2|\mathcal{N}|})$ si un automate cellulaire à une dimension est inversible, simplement

surjectif, ou aucun des deux. Il suffit pour cela de construire le graphe $\mathcal{D}$ à partir de la donnée de l'automate $\mathcal{A}$, par exemple sous forme de table, et de tester l'égalité $\tilde{\mathcal{G}} = \overline{\mathcal{G}}$ puis la présence éventuelle d'autres composantes fortement connexes.

Construction de $\mathcal{G}$ à partir de $\mathcal{A}$. Cette construction est faite en temps $O(\sigma^{|\mathcal{N}|})$. Il s'agit en effet de construire $\mathcal{G}$, ce qui se fait en temps $O(\text{nombre de sommets} + \text{nombre d'arêtes})$, soit $O(\sigma^{|\mathcal{N}|-1} + \sigma^{|\mathcal{N}|})$ puisque $\mathcal{G}$ est un graphe régulier de degré σ à $\sigma^{|\mathcal{N}|-1}$ sommets.

Construction de $\mathcal{H}$ à partir de $\mathcal{G}$. Il est possible de déduire $\mathcal{H}$ à partir de $\mathcal{G}$ en examinant tous les couples d'arêtes. Comme expliqué au paragraphe 3.2.2, cette construction peut-être effectuée en temps $O(\sigma^{2|\mathcal{N}|})$, qui est le nombre d'arêtes de $\mathcal{H}$.

Réduction de $\mathcal{H}$ à $\mathcal{D}$. On va pouvoir éliminer en deux temps les points isolés et les arêtes qui ne sont pas dans $\mathcal{D}$ en éliminant d'abord toutes les branches mortes (qui ne mènent pas à une composante fortement connexe), ensuite en inversant les branches, on finit d'éliminer les branches mortes pour obtenir $\mathcal{D}$. Pour cela, on applique un parcours en profondeur au graphe (ou au graphe inversé) en notant avec diverses couleurs les arêtes sur lesquelles on est passé.

blanc pour les sommets non atteints ;

gris pour les sommets déjà atteints en cours de recherche par la récursivité ;

noir pour un sommet qui fait partie d'une composante fortement connexe ;

vert pour un sommet a éliminer.

Le choix des couleurs est : au début tout le monde est blanc ; quand on teste un sommet : on le met gris, puis on teste tous ses fils ; s'il n'a pas de fils ou n'a que des fils verts, on le met en vert ; si au moins l'un de ses fils est gris ou noir, on le met en noir. Ce parcours permet d'éliminer toutes les branches mortes facilement en temps $O(\sigma^{2|\mathcal{N}|} + \sigma^{2(|\mathcal{N}|-1)}) = O(\sigma^{2|\mathcal{N}|})$, c'est à dire un passage sur chaque arête plus un travail sur chaque sommet. Le fonctionnement de cet algorithme est simple : un élément qui mène à une composante fortement connexe restera toujours noir ou gris ; un élément qui ne mène pas à une composante fortement connexe aura forcément tous ses fils qui ne mèneront pas sur des sommets déjà salis... sinon ils mènent à une composante fortement connexe soit dont ils font partie (gris), soit une autre ou éventuellement la même (noir). En appliquant deux fois l'algorithme, on obtient ainsi le graphe $\mathcal{D}$. De plus, on peut noter au passage si l'on a "sali" un point en dehors de $\tilde{\mathcal{G}}$ (test 1). On peut d'ores et déjà éliminer certains automates pour la bijectivité.

Vérifier $\tilde{\mathcal{G}} = \overline{\mathcal{G}}$. On peut faire ceci par un simple parcours en profondeur de $\tilde{\mathcal{G}}$. S'il reste des arêtes qui sortent de $\tilde{\mathcal{G}}$ et qui rentrent, on a $\tilde{\mathcal{G}} \neq \overline{\mathcal{G}}$ et l'automate n'est ni bijectif, ni surjectif ; sinon, il est au moins surjectif. Si, en plus, il remplit la condition énoncée plus haut, il est bijectif.

3.4 Configurations périodiques et bijectivité

Théorème 8 *La bijectivité d'un automate cellulaire $\mathcal{A}$ de dimension 1 sur les configurations périodiques implique la bijectivité de $\mathcal{A}$ sur toutes les configurations.*

Preuve. Rappelons que la bijectivité sur les configurations périodiques implique, quelque soit la dimension de l'automate, la surjectivité sur toutes les configurations. Il reste donc à prouver l'implication suivante : si $\mathcal{A}$ est bijectif pour les configurations périodiques, alors il est injectif pour toutes les configurations. Supposons donc qu'il existe deux configurations non périodiques ayant la même image. Ceci implique au moins l'existence d'un point en dehors de $\tilde{\mathcal{G}}$ dans $\mathcal{D}$. Ce point appartient à un chemin bi-infini, et il appartient donc à une composante fortement connexe, ou bien il est sur un chemin qui mène d'une composante connexe à une autre distincte. Si l'on suppose qu'il existe au moins deux composantes fortement connexes distinctes, alors il en est une qui est différente de $\overline{\mathcal{G}}$, et donc tout cycle dans celle-ci se traduira par deux configurations périodiques ayant même image et n'ayant aucun point commun (donc distinctes). On est donc amené à supposer que l'on est dans le cas où il existe une seule composante fortement connexe, qui est donc $\overline{\mathcal{G}}$, et donc distincte de $\tilde{\mathcal{G}}$. On peut donc construire une suite d'arcs qui sort de $\tilde{\mathcal{G}}$ et qui y rentre, en ayant au moins un point extérieur. Une autre propriété de $\tilde{\mathcal{G}}$ étant qu'il est hamiltonien (on peut passer par tous les sommets sans repasser deux fois par le même), implique qu'il est possible de construire un arc interne à $\tilde{\mathcal{G}}$ joignant la fin et le début de l'arc précédemment cité. Donc, il est possible de construire un cycle qui n'est pas entièrement dans $\tilde{\mathcal{G}}$, et il existe deux configurations périodiques distinctes ayant la même image. La proposition est donc prouvée par contraposée. □

4 Études des automates cellulaires partitionnés

4.1 Définitions et propriétés

4.1.1 Automates Cellulaires partitionnés

Un automate cellulaire partitionné peut être défini formellement comme un quadruplet $(d, \mathcal{S}, \mathcal{N}, f)$ avec :

- d la dimension de l'automate cellulaire ;
- $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \ldots \times \mathcal{S}_r$ avec $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_r$ des ensembles finis d'états ;
- $r \in \mathbb{N}$ le nombre de cellules avec lesquelles la cellule va échanger des informations ;
- $\mathcal{N} = \{z_1, z_2, \ldots, z_r\}$ un sous-ensemble fini de $\mathbb{Z}^d$ (le *voisinage* de l'automate) ;
- f la fonction locale de l'automate définie de $\mathcal{S}^{\mathcal{N}} \to \mathcal{S}$ avec les conditions suivantes :
  - f peut être décomposée en deux fonctions $\Phi \circ \mathcal{G}_{\mathcal{N}}$
  - Φ est une fonction de $\mathcal{S} \to \mathcal{S}$;
  - $\mathcal{G}_{\mathcal{N}}$ est la fonction de $\mathcal{S}^{\mathcal{N}} \to \mathcal{S}$ qui est définie par

$$\mathcal{G}_{\mathcal{N}}(z_1, z_2, \ldots, z_r) = (z_1^{(1)}, z_2^{(2)}, \ldots, z_r^{(r)}),$$

où $z_i^{(j)}$ représente la $j^{ème}$ composante du $i^{ème}$ voisin.

Un automate cellulaire partitionné sert en fait à faire circuler des informations (quantifiées par les divers ensembles d'états) qui se déplacent en ligne droite dans le cas où Φ est l'identité et qui peuvent être transformées localement par l'action de Φ.

4.1.2 Inversibilité

Lemme 3 *L'automate cellulaire partitionné $\mathcal{A}$ est inversible ssi la fonction Φ est une bijection.*

Preuve. Nous allons d'abord montrer que si Φ n'est pas une bijection, alors $\mathcal{A}$ n'est pas inversible. Pour cela, nous allons exhiber deux configurations qui ont même image. Φ n'étant pas bijective, elle n'est pas injective (puisque allant d'un ensemble fini dans le même ensemble fini). Il existe donc deux états de l'automate qui ont même image. Dénotons les par $(\sigma_1, \sigma_2, \ldots, \sigma_r)$ et $(\theta_1, \theta_2, \ldots, \theta_r)$. Prenons ensuite deux configurations identiques en tout points sauf sur les voisins d'une cellule c où la $i^{ème}$ composante du $i^{ème}$ voisin vaut σ_i dans la configuration C_1 et θ_i dans la configuration C_2. Si on applique l'automate $\mathcal{A}$ sur ces deux configurations, il devient évident les deux configurations vont passer dans un état identique puisque $\mathcal{G}_{\mathcal{N}}$ fera que toutes les cellules seront identiques sauf pour la cellule c, et que la fonction Φ appliquée sur celle-ci fera qu'elle sera identique dans les deux configurations. On obtiendra ainsi une configuration ayant deux antécédents distincts, ce qui prouve l'assertion faite plus haut.

On peut maintenant prouver que si Φ est une bijection, alors l'automate cellulaire $\mathcal{A}$ est inversible. Pour cela, on va donner l'automate inverse $\mathcal{A}^{-1}$.

On se donne ainsi $\mathcal{A}^{-1} = (d, \mathcal{S}, \mathcal{N}, g)$ avec g une fonction qui s'écrit $\mathcal{G}_{-\mathcal{N}} \circ \widetilde{\Phi^{-1}}$, où $\widetilde{\Phi^{-1}}$ représente l'extension de Φ^{-1} à $\mathcal{S}^{\mathcal{N}}$:

$$\widetilde{\Phi^{-1}}(z_1, z_2, \ldots, z_r) = (\Phi^{-1}(z_1), \Phi^{-1}(z_2), \ldots, \Phi^{-1}(z_r)).$$

Le calcul de la valeur de la $i^{ème}$ composante d'une cellule c nous donne que sa valeur est celle de $(\Phi^{-1}(c - z_i))^{(i)}$, soit $(\Phi^{-1}(\Phi(\mathcal{G}_{\mathcal{N}}(c - z_i))))^{(i)}$, soit $(\mathcal{G}_{\mathcal{N}}(c - z_i))^{(i)}$, soit enfin $c^{(i)}$. Ceci étant valable pour toute cellule c et pour tout $i, 1 \leq i \leq r$, l'automate $\mathcal{A}$ est inversible et d'inverse $\mathcal{A}^{-1}$. □

4.1.3 Conséquences

On a pu ainsi construire une famille d'automates dont il est aisé de vérifier l 'inversibilité. On utilisera ces automates pour vérifier qu'il existe des automates cellulaires inversibles linéaires universels dans la partie suivante. La complexité théorique de ces automates n'est pas très élevée dans la mesure où si l'on sait que l'on a affaire à ce type d'automates, il est possible de trouver l'inverse facilement. Il convient toutefois de noter que tous les automates cellulaires inversibles ne sont pas partitionnés.

4.2 Simulation d'une machine de Turing par des automates cellulaires partitionnés

On va ici s'intéresser à la possibilité de simuler des processus avec des automates cellulaires partitionnés, et en particulier des machines de Turing. Ceci permet de résoudre le problème de l'universalité des automates cellulaires inversibles, puisque si l'on sait simuler une machine de Turing, alors on peut calculer n'importe quelle fonction calculable. Morita a déjà analysé plusieurs fois les rapports entre universalité et automates cellulaires dans [8], [7] et [6].

4.2.1 Machine de Turing Déterministe

La machine de Turing [1] que nous allons considérer dans ce chapitre est définie comme un triplet $T = (\mathcal{F}, \mathcal{Q}, \tau)$ avec

- $\mathcal{F}$ un ensemble fini de symboles, les caractères de la machine. On distingue un symbole particulier noté 0, dit caractère *blanc*;
- $\mathcal{Q}$ un ensemble fini d'états parmi lesquels on distingue un *état d'acceptation* q_Y et un *état de refus* q_N ;
- τ une fonction de $\mathcal{F} \times \mathcal{Q} \to \mathcal{F} \times \mathcal{Q} \times \{gauche, droite\}$, qui est la fonction de *transition* de la machine de Turing T.

Une configuration de cette machine de Turing est donnée par une application de $\mathbb{Z}$ dans $\mathcal{F}$. A chaque pas de calcul, la tête de lecture initialement en position de départ effectue la transition correspondant à sa fonction τ selon son état actuel et la valeur de la case lue, écrit une nouvelle valeur dans la case, puis se déplace soit à gauche, soit à droite.

1. La machine de Turing donnée ici n'est pas forcément inversible ! Ceci est une amélioration par rapport au travail de Morita dans [8]

Une configuration valide est une application presque nulle de $\mathbb{Z}$ dans $\mathcal{F}$ (c'est à dire nulle partout sauf en un nombre finis de points), qui amène la machine dans un des deux états particuliers q_Y ou q_N.

4.2.2 Définition de $\mathcal{A}_\tau$

Étant donnée une machine de Turing T, on se donne un automate cellulaire partitionné $\mathcal{A}_\tau$:

$$\mathcal{N} = \{0, 1, -1, 2\};$$

$$\Sigma = \mathcal{Q} \cup \{\diamondsuit\};$$

$$\mathcal{A}_\tau = (1, \mathcal{F} \times \Sigma \times \Sigma \times (\mathcal{F} \times \Sigma \times \{gauche, droite\}), \mathcal{N}, \Phi \circ \mathcal{G}_\mathcal{N}\}.$$

On dénote un état d'une cellule par (a, b, c, d) et on définit Φ qui remplit les conditions suivantes :

— Si $b \neq \diamondsuit, c = \diamondsuit, d = (0, \diamondsuit, gauche)$, alors, en notant $(a', b', \delta) = \tau(a, b)$:

$$\begin{array}{rcll} \Phi(a, b, \diamondsuit, d) & = & (a', b', \diamondsuit, (a, b, gauche)) & \text{si } \delta = gauche\,; \\ & = & (a', \diamondsuit, b', (a, b, gauche)) & \text{si } \delta = droite. \end{array}$$

— Si $b = \diamondsuit, c \neq \diamondsuit, d = (0, \diamondsuit, gauche)$, alors, en notant $(a', c', \delta) = \tau(a, c)$:

$$\begin{array}{rcll} \Phi(a, \diamondsuit, c, d) & = & (a', c', \diamondsuit, (a, c, droite)) & \text{si } \delta = gauche\,; \\ & = & (a', \diamondsuit, c', (a, c, droite)) & \text{si } \delta = droite. \end{array}$$

— Si $b = c = \diamondsuit, d = (0, \diamondsuit, gauche)$, alors

$$\Phi(a, \diamondsuit, \diamondsuit, (0, \diamondsuit, gauche)) = (a, \diamondsuit, \diamondsuit, (0, \diamondsuit, gauche)).$$

On va montrer que la restriction de Φ aux configurations précédentes est injective. Pour toutes les autres configurations on peut alors choisir une image qui n'est pas dans les images déjà obtenues de façon à ce que la fonction Φ reste injective (et donc soit bijective).

Lemme 4 *Φ restreinte aux configurations précédentes est injective.*

Preuve. L'injectivité de Φ dans un même groupe de configurations est triviale. Il reste à vérifier simplement qu'il ne peut y avoir de confusions entre les groupes. On ne peut évidemment confondre des éléments appartenant aux deux premiers groupes puisque la valeur attachée à la dernière composante diffère par son dernier élément de manière évidente. De plus, aucun des éléments de ces deux groupes ne peut être assimilé comme venant du troisième puisque la valeur du deuxième élément de la dernière composante est différente de $\diamondsuit$ dans les deux premiers cas et qu'elle vaut $\diamondsuit$ dans le troisième. □

Corollaire 2 *Φ peut être prolongée en une fonction bijective ; donc $\mathcal{A}_\tau$ est un automate cellulaire partitionné inversible.*

4.2.3 Simulation de T par $\mathcal{A}_\tau$

Configurations valides de $\mathcal{A}_\tau$. On définit comme configurations valides de $\mathcal{A}_\tau$ les configurations qui sont de la forme $(0, \diamondsuit, \diamondsuit, (0, \diamondsuit, gauche))$ partout sauf sur un nombre fini de cellules qui peuvent prendre la valeur $(\gamma, \diamondsuit, \diamondsuit, (0, \diamondsuit, gauche)), \gamma \in \mathcal{F}$. De plus, il y aussi une cellule et une seule qui a la forme $(\gamma, a, \diamondsuit, (0, \diamondsuit, gauche)$ avec $a \neq \diamondsuit$. L'ensemble de ces conditions définit une *configuration valide*.

Lemme 5 *Il existe une injection de l'ensemble des configurations valides de T dans l'ensemble des configurations valides de $\mathcal{A}_\tau$.*

Preuve. Soit C une configuration valide de T. On peut construire une configuration valide de $\mathcal{A}_\tau$ comme suit :

- $\forall j \neq 1, \mathcal{C}_C(j) = (C(j), \diamondsuit, \diamondsuit, (0, \diamondsuit, gauche))$;
- $\mathcal{C}_C(1) = (C(0), q_0, \diamondsuit, (0, \diamondsuit, gauche))$, où q_0 est l'état de départ de la machine de Turing T.

Il est clair que la configuration obtenue pour l'automate $\mathcal{A}_\tau$ est valide. L'injectivité résulte naturellement du plongement de C dans $\mathcal{C}_C$. □

Simulation du calcul de T par $\mathcal{A}_\tau$.

Théorème 9 *L'automate $\mathcal{A}_\tau$ simule le calcul de T sur toute configuration C valide de T sans perte de temps.*

Preuve. Il suffit de mettre en évidence une fonction qui permet de transformer des configurations de $\mathcal{A}_\tau$ obtenues par itérations successives à partir d'une configuration valide. Il suffit tout simplement de transformer la ligne d'ordonnée 0 en une configuration de T et de dire que la tête de lecture est dans le seul état différent de $\diamondsuit$ sur les deuxième et troisième composantes de l'état de chaque cellule, et que sa position est l'ordonnée de la cellule précédemment citée, moins un si c'est le deuxième champ qui était différent de $\diamondsuit$, plus un sinon. Il faut garantir que l'on ne peut trouver qu'une seule cellule sur la ligne d'ordonnée 0 qui comprenne un symbole différent de $\diamondsuit$ pour prouver la validité de cette transformation.

On démontre par récurrence la propriété suivante : il existe une et une seule cellule d'abscisse x_0 avec le deuxième champ ou le troisième champ différent de $\diamondsuit$, et $\forall x > x_0$, le quatrième champ est égal à $(0, \diamondsuit, gauche)$. Supposons que ce soit le cas à la $n^{ième}$ itération. La tête de lecture se situe sur la cellule x_0 (symbole différent de $\diamondsuit$). Après l'application de $\mathcal{G}_\mathcal{N}$, la tête de lecture se déplace soit à gauche, soit à droite. Le décalage vers la gauche du seul groupe du quatrième champ différent de $(0, \diamondsuit, gauche)$ implique que celui-ci se trouve en position $x_0 - 2$, et la tête de lecture se retrouvant en position $x_0 - 1$ ou $x_0 + 1$, la quatrième composante vaut bien $(0, \diamondsuit, gauche)$ et la fonction Φ nous apprend alors qu'il ne peut y avoir qu'une seule tête de lecture après l'application de Φ. La condition est donc bien à nouveau vérifiée pour la $(n+1)^{ième}$ application de l'automate.

Il s'agit maintenant de vérifier qu'une itération de l'automate cellulaire correspond bien à un pas de calcul de la Machine de Turing T. Le déplacement de la tête est correct : il suffit de regarder la construction de la fonction. De plus, le ruban est correctement simulé, car il n'y a changement d'état que si la tête de lecture est sur la cellule en train d'opérer, et, au cas où elle opère, l'opération réalisée est précisément celle que ferait la machine de Turing (par définition de Φ). Donc, à tout instant t, il y a une correspondance exacte entre le ruban de T et la rangée d'automates cellulaires. □

On peut donc en tirer le corollaire suivant :

Corollaire 3 *Il existe des automates cellulaires inversibles linéaires universels.*

5 Un simulateur d'automates cellulaires

Une partie du stage a consisté en la programmation d'un logiciel qui simule l'action d'automates cellulaires sur des configurations finies. Le support employé pour ce programme était à l'origine un programme réalisé avec une interface OpenLOOK, réalisé pour son stage de DEA par Olivier Blettery. Le but de mes modifications était de corriger certaines imperfections, et d'ajouter un type d'automate dont la fonction de transition aurait pu être donnée de manière implicite en un langage de programmation, par exemple le langage C.

5.1 Présentation du programme CA

5.1.1 Objectif du programme

Le but du programme CA est d'interpréter des images comme des configurations finies d'automates cellulaires, et de visualiser le résultat de l'application d'un automate cellulaire sur cette configuration sous forme d'une autre image. Les nécessités imposées pour le programme étaient :

— Une interface utilisateur simple d'emploi ;
— Une visualisation des résultats immédiate ;
— La possibilité de comparer l'image originale et l'image obtenue ;
— La possibilité de changer d'image avec un même automate ;
— La possibilité de changer d'automate avec une même image.

Ceci peut en effet être utilisé à des fins de cryptage. Il suffit de posséder un automate inversible pour crypter une image et pour que le destinataire (qui possède l'automate inverse) puisse la décrypter.

5.1.2 Présentation

Ce logiciel est ainsi capable de charger en mémoire une image à partir d'un fichier, et d'afficher celle-ci dans une fenêtre appropriée. Il est également capable de charger un AC, à partir d'un fichier de type *cellauto*, contenant la description d'un automate cellulaire (c'est à dire le quadruplet (d, S, V, f)). Puis, grâce à l'image initiale, à la dimension de l'automate et à son nombre d'états, il crée un tableau d'entiers (voir partie 4.2) qui modélise une configuration périodique de l'AC qui se rapproche le plus possible de l'image de départ.

Grâce à la fonction f en mémoire, et en considérant le tableau représentatif de la configuration comme un tore, l'utilisateur peut itérer l'automate cellulaire sur la configuration le nombre de fois qu'il le désire. A cette nouvelle configuration correspond une unique image (possédant d'ailleurs les mêmes caractéristiques que l'image initiale : mêmes largeur, hauteur, et couleurs définies). Le logiciel traduit alors la configuration courante de l'automate en l'image qu'elle représente, et l'affiche dans une autre fenêtre que l'image initiale. L'opérateur peut ainsi comparer les 2 images, et juger de la qualité *visuelle* du cryptage effectué.

Le logiciel est alors capable de sauvegarder ces résultats dans des fichiers : on peut sauvegarder le cryptotexte (*i.e.* la configuration finale) dans un fichier de type *ca_config* qui contient la donnée entière du tableau de configuration ainsi que ses dimensions et le nombre d'états de cette configuration. On peut également sauvegarder l'image cryptée obtenue.

Le fait de sauvegarder la configuration est indispensable afin de pouvoir décrypter ultérieurement notre message, car le logiciel est capable, au lieu de charger une image, de charger directement en mémoire une configuration d'automate (à partir d'un fichier de type *ca_config*), et d'afficher l'image qu'elle engendre dans la fenêtre réservée à l'image initiale. Si cette configuration correspondait à l'itérée n fois d'un automate cellulaire $\mathcal{A}$ sur une configuration c, il suffira à l'utilisateur de charger le fichier contenant l'automate $\mathcal{A}^{-\infty}$ et d'itérer n fois celui-ci sur la configuration récemment chargée afin de retrouver la configuration c, et ainsi obtenir l'image initiale qui avait été codée.

FIGURE 2 – Exemple d'exécution

Remarque : l'utilisateur peut également faire fonctionner CA en mode *aveugle* (option `-blind`). CA se contente alors d'itérer l'automate désigné sur l'image voulue le nombre de fois spécifié, sans rien afficher, puis il sauvegarde automatiquement l'image cryptée et la configuration de l'automate engendrées, dans deux fichiers distincts, de types *image* et *ca_config*, respectivement.

5.2 Présentation du *loader* de SunOS

5.2.1 Édition de liens dynamique et statique

La formule adoptée par SunOS afin de gérer les exécutables est de proposer aussi bien les éditions de liens *dynamiques* (assemblage final du programme juste avant l'exécution) que les éditions de liens *statiques* (assemblage du programme au moment de la compilation). Un programme passe donc par les étapes indiquées Figure 3.

Cette structure permet une grande souplesse. L'édition de liens dynamique est réalisée en ne faisant, à la compilation, que noter l'emplacement des bibliothèques utilisées et des modules partageables (caractérisés par la terminaison `.so`). Il suffit ensuite, lors de

			Nom du résultat
Programme-Source	$\xrightarrow{Compilateur}$	Programme-assembleur	`*.s`
Programme-assembleur	$\xrightarrow{Assembleur}$	Module-objet	`*.o`
Module-objet	$\xrightarrow{ld\ et\ ld.so}$	Module-exécutable	`a.out` ou `*.so`
Module-exécutable	$\xrightarrow{exec}$	Code-machine	–

FIGURE 3 – Compilation et exécution d'un programme.

l'exécution du programme, de juxtaposer les différentes parties en allant les chercher à l'endroit indiqué dans les informations du `a.out`, et lors de l'appel à une fonction *externe* (d'une fonction qui fait partie d'un module qui n'était pas dans le `a.out`), on appelle en fait une fonction qui réoriente vers la bonne adresse mémoire. De plus, ce système augmente la portabilité d'un programme (c'est à dire sa capacité à fonctionner sur d'autres systèmes que celui du programmeur), puisque les bibliothèques peuvent changer en fonction du système utilisé, tant qu'on les situe à la même place dans l'arborescence.

L'édition de liens statiques, elle, privilégie la rapidité au détriment de l'encombrement mémoire. En effet, si l'option de compilation statique est choisie, les bibliothèques ne sont pas seulement notées, elles sont également intégrées au code du `a.out`. Ainsi, l'appel système `exec` n'a pas à aller chercher les bibliothèques sur le disque au moment de l'exécution. Mieux encore, le code est spécifié avec l'option PIC (Position-Independent Code) qui lui donne un adressage uniquement relatif. Ainsi, la pagination utilisée lors de l'exécution n'a pas d'importance.

La gestion des modules partageables est tout à fait similaire. Ce qui différencie un module partageable d'un programme normal est l'absence d'une fonction `main` qui désigne le début de programme. C'est sous cette forme que nous les utiliserons.

5.2.2 La gestion du gestionnaire de liens dynamiques

On s'attachera ici à la gestion de cette interface en langage C. Ce qui nous intéresse ici, c'est de pouvoir ajouter, en cours d'exécution du programme, un morceau de code qui n'est pas connu au moment de l'exécution du programme lui-même. Ce problème est résolu par l'utilisation des fonctions suivantes :

dlopen permet de donner un accès à un module partageable. Le module est juste recopié dans l'espace adressable du programme. `dlopen` renvoie un numéro d'identification du module rajouté ;

dlclose permet de fermer un accès à un module partageable. Si ce module a été ouvert plusieurs fois, il n'est pas enlevé de l'espace adressable tant que l'on ne l'a pas fermé suffisament de fois ;

dlsym permet de donner l'accès à une fonction [2] contenue dans un des modules objets. On désigne la fonction par son nom [3] et l'identificateur du module-objet où on la cherche. `dlsym` renvoie un pointeur sur la fonction demandée ;

dlerror donne une description de la dernière erreur qui s'est produite en employant les fonctions ci-dessus sous forme d'une chaîne de caractères.

5.3 Forme des automates

La solution sélectionnée initialement était de rentrer l'automate sous la forme la plus simple, c'est-à-dire en donnant explicitement le voisinage, et la table de transition de

2. Il est aussi possible d'accéder à des variables globales préinitialisées. La fonction renvoie alors un pointeur sur la valeur de la variable.

3. Par exemple le nom de la fonction **main** est `_main`. Le "_" est rajouté par le compilateur C. En assembleur, le nom de la fonction correspond directement au "label" utilisé.

```
/* Fichier en-tete pour automate cree automatiquement */
/* Automate : /tmp/sample */
/* Programme realise par Jean-Christophe Dubacq, 22 juillet 1993 */
/* Version raccourcie */
#define NB_ETATS 5
#define NB_VOISINS 2
typedef state unsigned short int;
void debut() {}
state delta(etat)
state *etat; {}
void fin() {}
```

FIGURE 4 – Exemple de squelette d'automate

l'automate. Mais il est un moyen qui semble plus pratique de rentrer la fonction de transition c'est d'en donner une formulation implicite. C'est cela qui a été ajouté.

5.3.1 Interface utilisateur

L'utilisateur n'a qu'à utiliser le programme `make_automata` qui lui génère un squelette de programme générique en C comme montré dans la figure 4. Il suffit pour cela de spécifier :

- le nombre d'états de l'automate ;
- le nombre de voisins de l'automate ;
- la position des voisins. L'ordre dans lequel on les donne est important puisqu'il déterminera l'ordre dans lequel seront rangés les valeurs dans le tableau (voir 5.3.2) ;
- un nom pour l'automate, éventuellement précédé d'un chemin absolu ou relatif pour indiquer le répertoire de travail.

Ceci s'effectue très simplement et de manière entièrement naturelle. L'utilisateur n'a ensuite plus qu'à éditer le code généré pour inclure la fonction de transition et éventuellement des fonctions de début de calcul et de fin de calcul.

Une variable d'environnement, appellée CALIB, est utilisée pour déterminer où ranger les automates dont le nom n'est pas un chemin absolu. Si elle n'est pas présente, le programme utilise par défaut `/tmp`.

Ensuite, il compile son programme avec une commande UNIX de la forme :

```
exemple% cc -c /tmp/sample.c
exemple% ld -o /tmp/sample.so /tmp/sample.o
exemple% rm /tmp/sample.o
```

Un fichier automate se présente sous forme de deux fichiers :

1. le fichier `nom.so` qui représente le code de la fonction de transition

2. le fichier `nom.ca` qui représente les informations nécessaires à l'automate (nombre et position des voisins, nombre d'états, position du fichier `nom.so`, *nombre magique* caractérisant le fichier.)

Pour l'utiliser dans le programme CA, il suffit de sélectionner l'automate exactement comme on le faisait auparavant : la forme de l'automate (table ou programme C) est totalement transparente pour l'utilisateur.

5.3.2 Programmation d'un automate

La programmation d'un automate est très simple. Il suffit de renvoyer à la fin de la fonction `delta` une valeur de type *unsigned short int* qui provient d'un codage numérique des états sous forme d'entiers. Le codage des états est entièrement laissé à l'utilisateur. La forme C de la fonction est très simple ; la fonction `delta`, lorsqu'elle est appellée pour appliquer la fonction de transition locale à un groupe de cellules, reçoit en entrée un tableau d'états (codés numériquement) dont le $n^{ième}$ élément correspond au $n^{ième}$ voisin (tel qu'ils ont été entrés lors de la création du fichier). Sur la base de ce codage, on entre une portion de code C qui calcule la valeur de la cellule-cible en fonction des autres ; toutes les subtilités du langage C pouvant être utilisées. Pour cela, on peut employer :

— des opérateurs classiques (+ - * / % et toutes les opérations habituelles) ;
— des tests, des boucles, des variables locales ;
— des fonctions externes déclarées dans le programme. Toutefois, **il est alors important d'avoir compilé le programme avec l'option -PIC** ;
— des variables globales appartenant à la procédure `debut`, par exemple des descripteurs de fichier ou des entiers. Celles-ci devront avoir étées déclarées sous la forme `static ...` à l'intérieur de cette même procédure, afin que le code engendré soit correct. Les variables globales déclarées *ne devraient pas* être utilisées [4] ;
— des fonctions appartenant à des bibliothèques. Il importe de lier alors ces bibliothèques avec l'option `-Bstatic` lors de la compilation de l'automate pour que ces bibliothèques soient intégrées directement dans le module. En effet, il n'y a pas de liaison automatique effectuée lors du chargement de l'automate en mémoire, et les fonctions ne seraient pas référencées.

L'ensemble de ces possibilités laisse donc une grande souplesse au programmeur pour faire son code. Le code de `delta` s'achève par `return(x);` où x désigne une variable de type *state*, (*i.e. unsigned short int*), qui représente la valeur à affecter à la cellule.

5.3.3 Résultats Divers

Transportabilité. Le problème de ce codage d'automate est que si l'on peut aisément et sans contre-indications déplacer le fichier-automate (fichier de type `nom.ca`), il est nécessaire de le modifier si l'on veut modifier la position du fichier-code (fichier de type `nom.so`). Un utilitaire avec interface graphique est prévu pour rectifier ce problème, et il

4. Il est possible que cela fonctionne avec certains compilateurs si le code généré n'utilise qu'un adressage réellement relatif.

suffira alors de modifier une ligne de texte, mais pour le moment il faut recréer le même automate au bon endroit et simplement remplacer le fichier C par le fichier-code déjà compilé. La transportabilité a toutefois fait l'objet d'une attention particulière puisque le programme s'assure que le fichier-code n'est pas désigné en adressage relatif (ce qu'il ne faut pas puisque cela obligerait à lancer le programme CA depuis un répertoire particulier).

Rapidité. La rapidité n'est pas améliorée par cette présentation des automates. En effet, le calcul est bien plus rapide lorsque l'on dispose de la table déjà toute calculée (automate de la première forme) que lorsque l'on doit faire plusieurs opérations. Le rapport de temps, sur les premières expériences prouve que le temps peut-être au moins multiplié par un facteur 50 (de une seconde à plus d'une minute sur une image de taille raisonnable). Ce résultat est attendu. En revanche, la diminution de l'encombrement mémoire est spectaculaire, puisqu'un programme de 32 kO peut refléter une table de plusieurs MO (notamment dans le cas des automates cellulaires partitionnés où une part importante de la table donne le même résultat). Le choix de ce type d'automate tient donc à une politique d'économie mémoire plutôt que de performances.

Références

[1] S. Amoroso and Y.N. Patt. Decision procedures for surjectivity and injectivity of parallel maps for tesselation structures. *J. Comp. Syst. Sci.*, 6 :448–464, 1972.

[2] T. Head. One-dimensional cellular automata : injectivity from unambiguity. *Complex Systems*, 3 :343–348, 1989.

[3] J. Kari. *Decision problems concerning cellular automata.* PhD thesis, University of Turku (Finland), 1989.

[4] J. Kari. Reversibility and surjectivity problems of cellular automata. *J. Comp. and System. Scien.*, 1991.

[5] E.F. Moore. Machine models of self–reproduction. *Proc. Symp. Apl. Math.*, 14 :13–33, 1962.

[6] K. Morita. Any irreversible cellular automaton can be simulated by a reversible. Technical report, IEICE, 1992.

[7] K. Morita. Computation universal models of two-dimensional 16-state reversible cellular automata. *Transactions of the IEICE*, E-75(1) :141–147, January 1992.

[8] K. Morita and M. Harao. Computation universality of one-dimensional reversible (injective). *Transactions of the IEICE*, E-72(6) :758–762, June 1989.

[9] J. Myhill. The converse to Moore's garden-of-eden theorem. *Proc. Am. Math. Soc.*, 14 :685–686, 1963.

[10] D. Richardson. Tesselations with local transformations. *Journal of Computer and System Sciences*, 6 :373–388, 1972.

[11] K. Sutner. De Bruijn graphs and linear cellular automata. *Complex Systems*, 5 :19–30, 1991.

6 Conclusion

Résultats Le principal résultat théorique obtenu pendant ce stage est une démonstration plus simple de l'universalité des automates cellulaires inversibles que celle de K. Morita. Sa démonstration impliquait des résultats préliminaires sur les machines de Turing réversibles (résultats qu'il avait d'ailleurs prouvés). L'utilisation des automates cellulaires *partitionnés* introduit une grande facilité pour construire des modèlisations de phénomènes réversibles, que ce soient des phénomènes physiques ou biologiques.

Apport du stage Ce stage m'a permis de voir ce qu'était un travail en laboratoire et en équipe. Il a comporté une phase bibliographique, un travail théorique et une partie de programmation, ce qui m'a donné une idée de ce que l'on peut rencontrer dans un travail de recherche.

Remerciements Je tiens à remercier Bruno Durand, qui m'a soutenu dans mon travail, ainsi que Jacques Mazoyer, qui a bien su me conseiller. Je tiens également à remercier Thomas, qui a patiemment relu toutes mes fautes, et Zsuzsanna pour son soutien moral.