Projet de Programmation Fonctionnelle Automates Cellulaires

Gourgoulhon Maxime, Jacquette Pierrick, *Université Paris Diderot*

Ce document est un rapport pour un projet de programmation fonctionnelle, mené en novembre-décembre 2015 par Maxime Gourgoulhon et Pierrick Jacquette, dans le cadre des études en licence d'Informatique à l'Université Paris Diderot.

Table des matières

1 Introduction

Un *automate cellulaire* consiste en une grille régulière de **cellules** contenant chacune un **état** choisi parmi un ensemble fini et qui peut évoluer au cours du temps. L'état d'une cellule au temps $t+1$ est fonction de l'état au temps t d'un nombre fini de cellules appelé son **voisinage**. À chaque nouvelle unité de temps, les mêmes règles sont appliquées simultanément à toutes les cellules de la grille, produisant une nouvelle **génération** de cellules dépendant entièrement de la génération précédente.

(réf. wikipédia)

Ici, on étudie des cellules à 2 états (vivant ou mort) et utilisant le voisinage suivant :

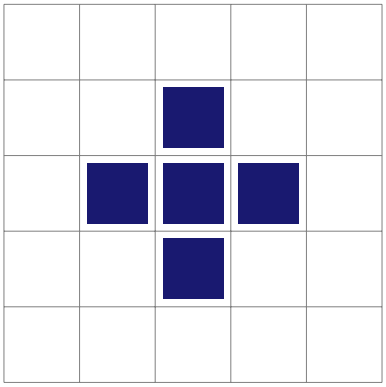


Figure 1 – *Von Neumann de rayon 1*

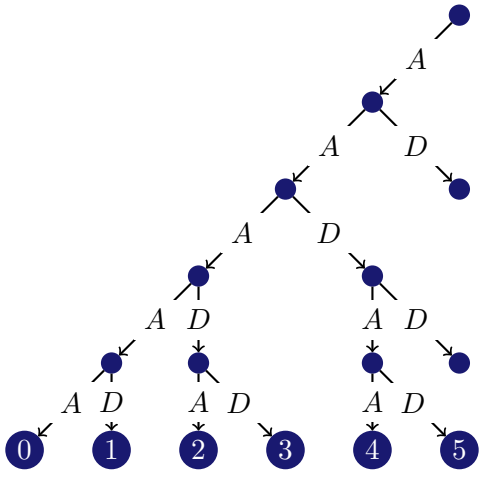
On veut coder une fonction d'évolution d'un automate, pour simuler l'évolution de générations en fonction d'un automate donné.

Une génération G d'un automate cellulaire est dite **stable** si la génération suivante G' produite à partir de G par application des règles d'évolution de l'automate à toutes les cellules est égale à G. On dit aussi qu'elle est stable si elle est un point fixe de la fonction d'évolution de l'automate.

On veut aussi, suivant un automate (ensemble de $2^5 = 32$ règles) et une taille de grille (carrée, torique), trouver toutes les génération stables.

2 Sources

La liste des règles d'un automate est stockée dans un tableau de 32 cases ; chaque case correspond à un état A ou D (par défaut D), à chaque indice correspond un voisinage, celui du parcours de l'arbre suivant (seule une partie de la gauche de l'arbre est dessinée) :



Cela revient à convertir un voisinage $XXXXX$ en binaire, avec $A \to 0$ et $D \to 1$; par exemple $DDADA \to 11010 \to 26$, ainsi à la 27ième case se trouve l'état suivant d'une cellule ayant un tel voisinage.

Le projet contient les sources suivantes :

Table 1 – *Liste des fichiers*

`Makefile`	compilation
`README.md`	explications
`automaton.ml`	automate
`automaton.mli`	
`core.ml`	types
`formula.ml`	
`interfaceGeneration.ml`	ui
`interfaceGeneration.mli`	
`mainShow.ml`	ui
`mainView.ml`	
`minisat`	linux binary
`print.ml`	affichage texte
`print.mli`	
`read.ml`	lecture
`read.mli`	
`show_stable.ml`	calcule des stables
`show_stable.mli`	
`simulate.ml`	simulation
`simulate.mli`	
`stable.ml`	transformation fnc
`stable.mli`	
`test`	config init
`test.ml`	main
`view_stable.ml`	ui
`view_stable.mli`	

3 Compilation

Assurez-vous d'avoir bien installé ocaml, et les librairies : `graphics.cma` et `str.cma`.

Pour compiler les sources :

```
make
make show
make view
```

Ensuite, générer la documentation :

```
make doc
make dot
```

Pour nettoyer :

```
make clean
```

N.B. : Le programme utilise minisat [1] pour calculer les générations stables, une version compilée exécutable pour Linux est inclue avec les sources, si vous utilisez un autre système vous devez inclure votre version compilée à la place.

1. http://minisat.se/MiniSat.html

4 Utilisation

Les règles de l'automate (celles qui donne donnent à une cellule l'état vivant) et la grille de base se trouve dans le fichier `test`. Chaque règle est de la forme EtatNord EtatEst EtatSud EtatOuest EtatCellule, des cinq états valant soit A soit D (ex : AADAD).

Pour simuler graphiquement l'évolution de la générationZéro : `./show`.

Ensuite, l'exécution de la commande `./prog` génère le fichier `gens` (et aussi des fichiers temporaires pour les calculs) contenant toutes les générations stables trouvées.

Après, on peut afficher les générations trouvées avec `./view`.

5 Liens

Github (sources) : https://github.com/XAMEUS/Two-dimensional-Cellular-Automaton
Documentation : http://xameus.github.io/Two-dimensional-Cellular-Automaton/doc/