

Automates cellulaires

Projet de programmation fonctionnelle - PF5

Maxime Gourgoulhon, Pierrick Jacquette

29 décembre 2015

Université Paris Diderot

Table des matières

1. Introduction
2. Simulation
3. Automate -> FNC
4. MiniSat

Introduction

Automate cellulaire

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.

wikipédia

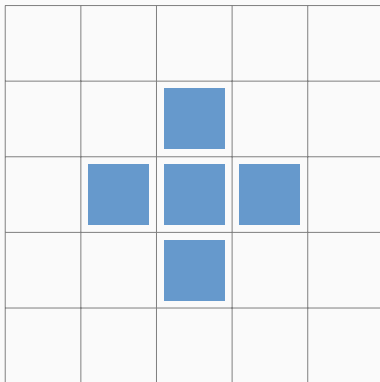


FIGURE 1: Von Neumann de rayon 1

FIGURE 2 : Fichier texte de description

7
Regles
AAAAA
AAAAD
AAADA
AAADD
GenerationZero
DAAAAAA
ADAAAAA
AADAAAA
AAADAAA
AAAADAA
AAAAADA
AAAAAAD

FIGURE 3 : Core.ml

```
type state = A | D;;  
type generation = state array array;;  
type automaton = state array;;
```

FIGURE 4 : Structure des règles (type automaton)

00	:	AAAAA	:	00000	→	A
01	:	AAAAD	:	00001	→	A
02	:	AAADA	:	00010	→	A
03	:	AAADD	:	00011	→	A
04	:	AADAA	:	00100	→	D
—						
—						
—						
31	:	DDDDD	:	11111	→	D

Simulation

FIGURE 5: stable.ml

```
let next_generation (aut:automaton) (gen:generation) =  
  let size = length gen in  
  let next_gen = make_matrix size size D in  
  for i = 0 to size - 1 do  
    for j = 0 to size - 1 do  
      next_gen.(i).(j) <- next i j gen aut  
    done;  
  done; next_gen  
;;
```

FIGURE 6: stable.ml

```
let next i j (gen:generation) (aut:automaton) =  
  if is_rule aut [  
    gen.(i).(j);  
    gen.(i).(if j>0 then j-1 else length gen -1);  
    gen.((i+1) mod (length gen)).(j);  
    gen.(i).((j+1) mod (length gen));  
    gen.(if i>0 then i-1 else (length gen -1)).(j)  
  ] then A else D  
;;
```

Automate -> FNC

- trouver toutes les règles "instables"

ex : $DADDA \rightarrow D$ est instable

$$(\neg x(i-1, j) \wedge x(i, j+1) \wedge \neg x(i+1, j) \wedge \neg x(i, j-1) \wedge x(i, j))$$

- trouver toutes les règles "instables"

ex : $DADDA \rightarrow D$ est instable

$$(\neg x(i-1, j) \wedge x(i, j+1) \wedge \neg x(i+1, j) \wedge \neg x(i, j-1) \wedge x(i, j))$$

- négation, toutes les règles "stables"

$$(x(i-1, j) \vee \neg x(i, j+1) \vee x(i+1, j) \vee x(i, j-1) \vee \neg x(i, j))$$

- trouver toutes les règles "instables"
ex : $DADDA \rightarrow D$ est instable
 $(\neg x(i-1, j) \wedge x(i, j+1) \wedge \neg x(i+1, j) \wedge \neg x(i, j-1) \wedge x(i, j))$
- négation, toutes les règles "stables"
 $(x(i-1, j) \vee \neg x(i, j+1) \vee x(i+1, j) \vee x(i, j-1) \vee \neg x(i, j))$
- lier toutes ces disjonctions pour obtenir la formule sous FNC.

MiniSat

FIGURE 7: fnc.dimacs

p cnf 9 9
6 7 3 8 9 0
6 7 3 -8 -9 0
...
6 -7 -3 -8 -9 0

...

7 2 4 3 1 0
7 2 4 -3 -1 0
...
-7 -2 -4 -3 -1 0

FIGURE 8 : fnc.dimacs

```
let rec show_stable () =  
  let _ = command "./ minisat_ fnc.dimacs_ tmp_gen_ >> _log" in  
  let gen = is_stable "tmp_gen" in  
  if gen = "" then  
    print_string "\ndone\n"  
  else  
    begin  
      add_line (inv (split (regexp "_+") gen)) "fnc.dimacs";  
      add_line gen "gens";  
      show_stable ()  
    end  
;;
```

Questions?