Automates cellulaires

Projet de programmation fonctionnelle - PF5

Maxime Gourgoulhon, Pierrick Jacquette

21 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

Voisinage

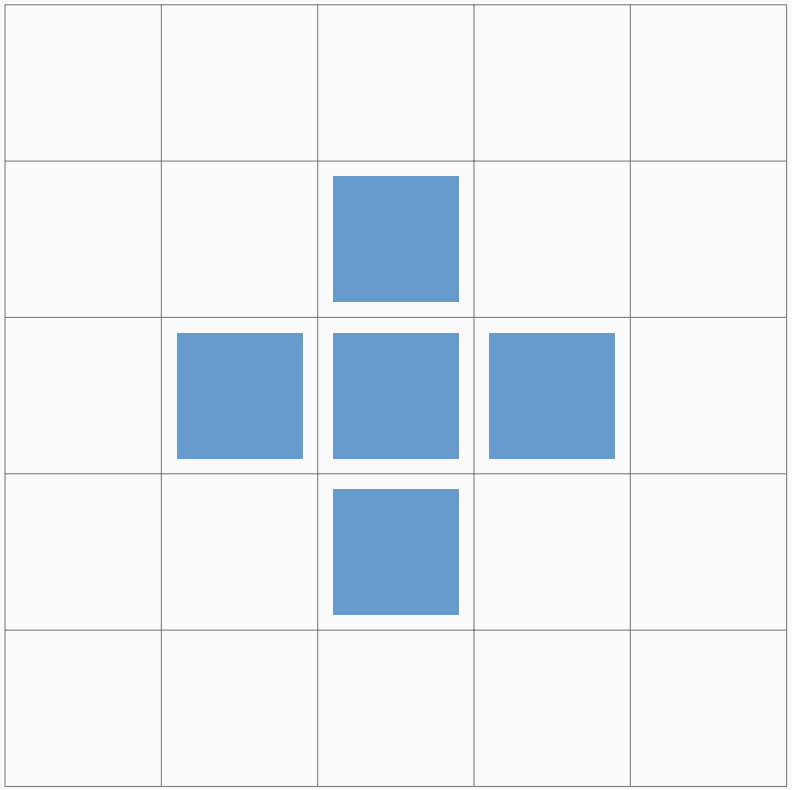

Figure 1 : Von Neumann de rayon 1

Fichier texte

Figure 2 : Fichier texte de description

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

Types

Figure 3 : Core.ml

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

Règles

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

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
;;
```

Simulation - next

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

Automate -> FNC

- trouver toutes les règles "instables"
  ex : $DADDA \rightarrow D$ est instable
  $(\neg x(i-1, j) \wedge x(i, j+1) \wedge \neg x(i+1, j) \wedge \neg x(i, j-1) \wedge x(i, j))$

Automate -> FNC

- trouver toutes les règles "instables"
  ex : $DADDA \rightarrow D$ est instable
  $(\neg x(i-1, j) \wedge x(i, j+1) \wedge \neg x(i+1, j) \wedge \neg x(i, j-1) \wedge x(i, j))$
- négation, toutes les règles stables
  $(x(i-1, j) \wedge \neg x(i, j+1) \wedge x(i+1, j) \wedge x(i, j-1) \wedge \neg x(i, j))$

Automate -> FNC

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

MiniSat

Figure 7 : fnc.dimacs

```
p cnf 25 25
20 21 5 24 −25 20 21 5 −24 −25 ... −20 −21 5 −24 25 0
19 25 4 23 −24 19 25 4 −23 −24 ... −19 −25 4 −23 24 0
18 24 3 22 −23 18 24 3 −22 −23 ... −18 −24 3 −22 23 0
  ...
22 3 7 1 −2 22 3 7 −1 −2 ... −22 −3 7 −1 2 0
21 2 6 5 −1 21 2 6 −5 −1 ... −21 −2 6 −5 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 ?