Rapport de Stage L3

Yoann Corde–Drouet
Encadré par Victor Poupet
LIRMM, équipe ESCAPE

Juin-Juillet 2025

Contexte général

Depuis leur introduction dans les années 1940 par J. von Neumann et S. Ulam, les automates cellulaires ont été largement étudiés, à la fois en tant que systèmes dynamiques discrets et en tant que modèles de calcul massivement parallèles.

Beaucoup de constructions complexes d'automates cellulaires combinent plusieurs mécanismes mais la définition classique d'automate cellulaire rend fastidieux leur description formelle et ne permet pas de séparer clairement leurs différents mécanismes.

On s'intéresse donc aux automates cellulaires à signaux, une représentation équivalente qui permet de clairement séparer les différentes parties d'une construction complexe et de décrire de manière incrémentale la dynamique de l'automate.

Problème étudié

Pendant mon stage j'ai exploré cette représentation par signaux à travers plusieurs directions : la réécriture de constructions précédemment existantes à l'aide de signaux, la preuve de propriétés sur des constructions à bases de signaux et la construction d'automates à signaux ayant des propriétés naturelles sur les signaux

Contribution proposée

J'ai décrit et prouvé en Rocq une construction qui prend un automate cellulaire à signaux à une dimension et en crée un autre qui fait, en une étape de temps, k étapes du premier.

J'ai également prouvé en Rocq la correction d'une construction de l'automate de Fischer (automate permettant de calculer les nombres premiers) traduite avec des signaux par mon tuteur.

Champ et limites de la contribution

Ma contribution représente près de 6000 lignes de preuves en Rocq (elles sont sur ce *GitHub*.) et a permis d'illustrer l'apport de ce nouveau modèle dans la simplification des preuves formelles sur les automates cellulaires.

La preuve de l'automate de Fischer n'a pas d'application directe mais elle a permis de voir des mécanismes sur les automates à signaux qui ne sont pas propres à cette preuve, comme la preuve de propriétés seulement sur des sous-ensembles de signaux interdépendants.

Bilan et perspectives

Que ce soit la traduction en signaux d'anciennes constructions, l'adaptation de transformations directement sur les automates à signaux (théorèmes d'accélération), les soucis d'implémentation de tels automates ou même la recherche de principes propres au modèle avec signaux (conservation des signaux) ; cette nouvelle définition offre encore de nombreux sujets de recherche.

Table des matières

1 Introduction

Mon stage s'est déroulé du 2 juin au 18 juillet 2025 au sein de l'équipe ESCAPE au LIRMM à Montpellier. J'étais encadré par Victor Poupet et accompagné d'une autre stagiaire de M1. Le sujet était large et nous n'avons pas travaillé sur les mêmes choses mais c'était agréable de partager cette expérience avec elle. A mon arrivée au LIRMM j'ai pu assister aux journées SDA2 2025 (Systèmes Dynamiques, Automates et Algorithmes) qui m'ont introduit à l'étude des systèmes dynamiques et en particulier des automates cellulaires. Pendant mon stage j'ai également pu assister à des conférences d'autres équipes du LIRMM (ALGCO et ECO).

Mon stage explorait les automates cellulaires à signaux, une nouvelle description équivalente des automates cellulaires qui se concentre sur la transmission de l'information entre les cellules. Cette idée est déjà présente dans les preuves de constructions complexes : *Fischer*(1) parle de "pulse" et dans *Discrete Parabolas and Circles on 2D Cellular Automata*(2) sont définis des "signals". Cette approche permet de découper les différents mécanismes d'une construction complexe, la rendant plus simple à décrire et à prouver formellement.

2 Automates Cellulaires à signaux

2.1 Définitions

Un automate cellulaire à signaux peut être vu comme un quadrillage de cellules chacune contenant certaines informations (les signaux) parmi un nombre fini d'informations. A chaque étape de temps (temps discret et synchrone), chaque cellule reçoit de nouveaux signaux en fonctions des signaux qu'elle et ses voisines portaient.

Définition 1 *Un automate cellulaire à signaux est un triplet $\mathcal{A} = (d, S, R)$ où :*

- *$d \in \mathbb{N}$ est la dimension*
- *$S = (s_0, ..., s_k)$ est un ensemble fini de signaux tel que chaque cellule de l'automate contient un sous ensemble de S*
- *R est une liste de règles où une règle est la donnée d'une condition booléeene sur les signaux portés par la cellule et ses voisines et de productions de signaux sur la cellule.*

Étant donné un automate cellulaire à signaux $\mathcal{A} = (d, S, R)$, on appelle configuration toute application c de $\mathbb{Z}^d$ dans $\mathcal{P}(S)$. Ainsi pour $z \in \mathbb{Z}^d$, $c(z)$ est l'ensemble des signaux portés par la cellule à la position z dans la configuration c.

De plus, on appelle δ la fonction de transition de l'automate qui à une configuration associe la configuration obtenue en appliquant toutes les règles de R possibles.

2.2 Logique sur les signaux

Les règles demandent de tester une condition booléenne sur les signaux portés par la cellule et ses voisines, on définit pour cela une logique sur les siganux.

Définition 2 *Pour $d \in \mathbb{N}$, on appelle SL_d l'ensemble des formules sur les signaux, il est définit comme étant le plus petit ensemble tel que :*

- $\top \in SL_d$ *et* $\bot \in SL_d$
- *Si* $k \in \mathbb{Z}^d$ *et* $s \in S$ *alors* $k.s \in SL_d$
- *Si* $\phi \in SL_d$ *alors* $\neg\phi \in SL_d$
- *Si* $\phi \in SL_d$ *et* $\psi \in SL_d$ *alors* $\phi \wedge \psi \in SL_d$ *et* $\phi \vee \psi \in SL_d$

Définition 3 *Pour c une configuration, $z \in \mathbb{Z}^d$ et $\phi \in SL_d$ on définit $(c, z) \models \phi$ ("c satisfait ϕ en z") par récurrence sur la structure de ϕ :*

- $(c, z) \models \top$ *est vrai et* $(c, z) \models \bot$ *est faux*
- $(c, z) \models k.s$ *ssi* $s \in c\,(z + k)$
- $(c, z) \models \neg\phi$ *ssi* $(c, z) \not\models \phi$
- $(c, z) \models \phi \wedge \psi$ *ssi* $(c, z) \models \phi$ *et* $(c, z) \models \psi$
- $(c, z) \models \phi \vee \psi$ *ssi* $(c, z) \models \phi$ *ou* $(c, z) \models \psi$

Ainsi les règles sont de la forme (ϕ, s) telles que si $(c, z) \models \phi$ alors $s \in \delta(c)(z)$.

2.3 Implémentation en Rocq

J'ai travaillé sur des automates cellulaires à signaux à une dimension, j'ai représenté les signaux par des entiers et les ensembles de siganux par des listes d'entier. Définitions en Rocq :

```
Definition signal := nat.
Inductive signal_form :=
  | f_true : signal_form
  | f_false : signal_form
  | f_atome : (Z * signal) -> signal_form
  | f_and : signal_form -> signal_form -> signal_form
  | f_or : signal_form -> signal_form -> signal_form
  | f_not : signal_form -> signal_form.
Definition rule : Type := signal_form * signal.
```

```
Definition rules := list rule.
Definition config := Z -> list signal.
Definition empty_config : config := fun x => [].
```

On définit également la satisfiabilité :

```
Fixpoint signal_sat_z (c : config) (z : Z) (phi : signal_form) :
   bool :=
  match phi with
  | f_true => true
  | f_false => false
  | f_atome (k,s) => existsb (fun x => Nat.eqb x s) (c (z+k))
  | f_and phi1 phi2 => (signal_sat_z c z phi1) && (signal_sat_z c
     z phi2)
  | f_or phi1 phi2 => (signal_sat_z c z phi1) || (signal_sat_z c z
      phi2)
  | f_not psi => negb (signal_sat_z c z psi)
  end.
```

Enfin, on définit la fonction δ et son itération (delta_one_rule applique une règle à toutes les positions et delta l'appelle pour chaque règle) :

```
Definition delta_one_rule (r : rule) (c_pred : config) (c : config
   ) : config :=
  fun z =>
    match r with (phi,s) =>
     if (signal_sat_z c_pred z phi) then s::(c z) else (c z)
     end.
Fixpoint delta (rs : rules) (c : config) : config :=
  match rs with
    | [] => empty_config
    | t::q => delta_one_rule t c (delta q c)
  end.
Fixpoint delta_n (rs : rules) (c : config) (n : nat) :=
  match n with
  | 0%nat => c
  | S m => delta_n rs (delta rs c) m
  end.
```

Ici on a $\delta^{n+1} = \delta^n \circ \delta$ mais on aurait pu la dèfinir comme $\delta^{n+1} = \delta \circ \delta^n$ (cela aurait été plus arrangeant dans la preuve de l'automate de Fischer).

3 k étapes en une

J'ai passé tout le début de mon stage à étudier des constructions et transformations d'automates cellulaires dans le but de les traduire pour les automates cellulaires à signaux. Parmi elles se trouvaient les théorèmes d'accélérations constante

et linéaire à une dimension qui demandaient tous les deux de pouvoir créer un automate cellulaire à partir d'un autre, tel que le premier fasse en un temps ce que l'autre fait en k. J'ai choisi dans cette partie, contrairement à celle pour Fischer, de décrire la construction et sa preuve sans faire référence au code Rocq (trouvable dans le fichier *k_steps.v*.) car toutes les étapes se traduisent relativement bien.

Dans la suite on considère un automate cellulaire à signaux $\mathcal{A} = (S, R)$ à une dimension et on note SL l'ensemble SL_1.

3.1 Transformation des formules

Définition 4 *Pour $\phi \in SL$ et $p \in \mathbb{Z}$ on définit $\overline{\phi}^p$ par récurrence sur ϕ telle que :*

- $\overline{\top}^p = \top$ *et* $\overline{\bot}^p = \bot$
- *Si $k \in \mathbb{Z}$ et $s \in S$ alors* $\overline{k.s}^p = (p+k).s$
- *Si $\phi \in SL$ alors* $\overline{\neg\phi}^p = \neg\overline{\phi}^p$
- *Si $\phi \in SL$ et $\psi \in SL$ alors* $\overline{\phi \wedge \psi}^p = \overline{\phi}^p \wedge \overline{\psi}^p$ *et* $\overline{\phi \vee \psi}^p = \overline{\phi}^p \vee \overline{\psi}^p$

$\overline{\phi}^p$ est ϕ dans laquelle tous les (k.s) ont été décalés de p.

Lemme 1 *Pour tout $\phi \in SL$, $p \in \mathbb{Z}$, c configuration et $z \in \mathbb{Z}$,*

$$(c, z) \models \overline{\phi}^p \text{ ssi } (c, z+p) \models \phi.$$

Démonstration 1

Définition 5 *Pour $s \in S$ on note $R_s = \{\phi : (\phi, s) \in R\}$. Pour $\phi \in SL$ on définit $\phi^{2\text{-etapes}}$ par récurrence sur ϕ telle que :*

- $\top^{2\text{-etapes}} = \top$ *et* $\bot^{2\text{-etapes}} = \bot$
- *Si $k \in \mathbb{Z}$ et $s \in S$ alors* $(s.k)^{2\text{-etapes}} = \bigvee_{\psi \in R_s} \overline{\psi}^k$
- *Si $\phi \in SL$ alors* $(\neg\phi)^{2\text{-etapes}} = \neg\phi^{2\text{-etapes}}$
- *Si $\phi \in SL$ et $\psi \in SL$ alors* $(\phi \wedge \psi)^{2\text{-etapes}} = \phi^{2\text{-etapes}} \wedge \psi^{2\text{-etapes}}$ *et* $(\phi \vee \psi)^{2\text{-etapes}} = \phi^{2\text{-etapes}} \vee \psi^{2\text{-etapes}}$

Lemme 2 *Pour tout $\phi \in SL$, c configuration et $z \in \mathbb{Z}$,*

$$(\delta(c), z) \models \phi \text{ ssi } (c, z) \models \phi^{2\text{-etapes}}.$$

Démonstration 2

3.2 Construction du nouvel automate

Définition 6 *Pour $k \in \mathbb{N}$ on définit R_k par récurrence tel que :*

- $R_0 = R$
- $R_{k+1} = \{(\phi^{2\text{-}etapes}, s) : (\phi, s) \in R_k\}$

On définit $\mathcal{A}_k = (S, R_k)$ et on note δ_k la fonction de transition de cet automate.

On veut maintenant montrer que $\mathcal{A}_k$ est un automate qui effectue (k+1) étapes de $\mathcal{A}$ en un temps.

Théorème 1 *Soit $k \in \mathbb{N}$, $\delta^{k+1} = \delta_k$* *Démonstration 3*

Ainsi pour tout $k \in \mathbb{N}^*$, $\mathcal{A}_{k-1}$ est un automate cellulaire à signaux qui effectue k étapes de $\mathcal{A}$ en une étape de temps. On peut remarquer que ces démonstrations restent inchangées en dimensions plus élevées : le résultat est généralisable pour tout automate cellulaire à signaux.

4 Automate de Fischer

Fort d'avoir prouvé le résultat précédent en Rocq j'ai voulu continuer dans cette direction en prouvant la correction de l'automate de Fischer, la construction que mon maître de stage avais utilisée comme exemple introductif aux automates cellulaires à signaux. Cette preuve m'a pris 3 semaines : prouver des propriétés sur une construction précise était plus fastidieux que de prouver une transformation générique.

4.1 Présentation de la construction

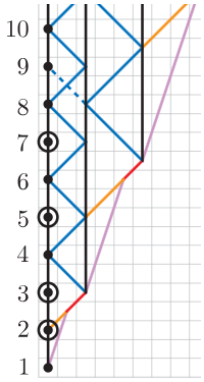

FIGURE 1 – Diagramme espace-temps de l'automate de Fischer

L'automate de Fischer est un automate cellulaire à une dimension attribuée Patrick C. Fischer (1).

Tous les temps $t = 2n$, il marque la cellule initiale si, et seulement si, n n'est pas premier. Il suit une logique similaire au crible d'Ératosthène pour marquer tous les multiples d'un nombre à partir de son carré.

4.2 Description avec des signaux

La description que j'utilise a 13 signaux : ce nombre n'est pas minimal mais il permet de simplifier les dépendances entre les signaux et donc les preuves.

FIGURE 2 – Les signaux utilisés (Init est appelé Init_1 en Rocq)

Règles sur les signaux :

La configuration initiales est composée du signal Init sur la cellule 0 et d'aucun signal sur les autres. On notera ϕ -> s au lieu de (ϕ, s).

Initialisation : 0.Init -> Init_2
0.Init -> slow_1
0.Init -> composite
0.Init -> wall
0.Init_2 -> Init_3
0.Init_3 -> build_r

Signal lent : -1.slow_3 -> slow_1
0.slow_1 $\wedge$ $\neg$ 0.build_r -> slow_2
0.slow_2 -> slow_3

Construction : 0.wall -> wall
-1.build_r $\wedge$ -1.slow_1 -> gen
-1.build_r $\wedge$ -1.slow_1 -> slow_1
0.gen -> wall
1.gen -> build_l
1.build_l $\wedge$ $\neg$ 1.wall -> build_l
-1.build_l $\wedge$ -1.wall -> build_r
-1.build_r $\wedge$ $\neg$ -1.slow_1 -> build_r

Rebonds : 1.gen -> bounce_l
1.bounce_l $\wedge$ $\neg$ 1.wall -> bounce_l
-1.bounce_l $\wedge$ -1.wall -> bounce_r
-1.bounce_r $\wedge$ $\neg$ -1.wall -> bounce_r
1.bounce_r $\wedge$ 1.wall -> bounce_l

Multiple : 1.bounce_l -> composite
1.composite -> composite

On définit ces règles en Rocq, par exemple :

```
Definition init_1 : signal := 0.
Definition init_2 : signal:= 1.
Definition phi_init_1 : signal_form := f_atome (0,init_1).
Definition rule_init_1_to_2 : rule := (phi_init_1,init_2).
```

4.3 Preuve de la construction

Dans cette partie je m'appuie sur certains lemmes faits en Rocq pour décrire les étapes principales de la preuve, mais je ne montre pas directement ces lemmes : ils peuvent être trouvés dans le fichier *fischer.v*.

Les figures avec des points ont été réalisées par un *logiciel* codé par mon maître de stage.

4.3.1 Prétraitement

Après les déclaration des objets, des règles de Fischer et de quelques lemmes utilitaires, la première étape de la preuve a été de traduire les règles en condition simples sur l'appartenance de signaux à des cellules.

Par exemple :

```
Lemma in_slow_1 (c : config) (z : Z) :
In slow_1 ((delta regles c) z) <->
In init_1 (c z)
\/ In slow_3 (c (z-1))
```

4.3.2 Position des générateurs

L'étape importante de la preuve est de décrire la position des générateurs. Pour cela on s'intéresse à la sous-construction suivante :

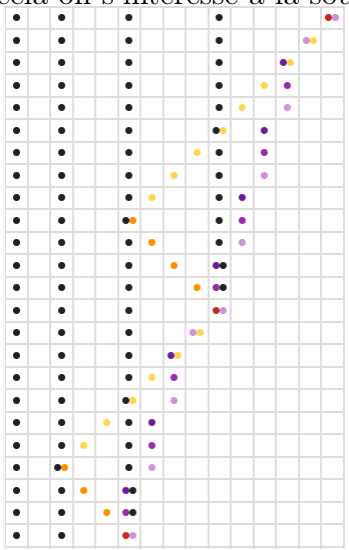

FIGURE 3 – Fischer avec seulement les signaux gen, wall, build_l, build_r et les slow

Au dela de la configuration 3, les signaux qui composent cette sous-construction sont indépendants de ceux non présents.

Partant d'une configuration avec un slow_1 et un gen en $z \in \mathbb{Z}$, un wall en $z - n$ (le plus proche) et aucun autre slow ou build dans les autres cellules on va montrer que $3n + 1$ temps plus tard on a un wall en z, un gen et un slow_1 en $z + n + 1$ et aucun slow ou build dans les autres cellules.

Or la configuration au temps 5 rescpecte cette propriété (Lemme Rocq 1) et on pourra ainsi construire les générateurs un à un et décrire complètement leur position en espace et en temps.

Pour cela on décrit d'abbord la propagation des signaux slow et build et leur condition :

- les signaux slow se propagent à vitesse $\frac{1}{3}$ vers la droite du temps que slow_1 ne recontre pas un build_r (Lemme Rocq 2)
- le signal build_l se propage à vitesse 1 vers la gauche du temps qu'il ne rencontre pas de mur (Lemme Rocq 3)
- le signal build_r se propage à vitesse 1 vers la droite du temps qu'il ne rencontre pas slow_1 (Lemme Rocq 4)

Ensuite il faut s'assurer que les signaux build_r et slow_1 se rencontrent au bon endroit (et non pas avant). Pour cela :

- On montre que si une configuration n'a pas de signaux d'initialisation et possède un seul signal entre build_r, build_l et gen alors toute configuration suivante conserve cette propriété (Lemme Rocq 5) et de même pour les slow. Comme cette propriété est vérifiée à la cinquième configuration, on n'a besoin que de décrire l'èvolution de deux signaux pour toute la construction (on ne risque pas d'avoir un nouveau signal problématique qui apparait).
- On montre des bornes sur la propagation des signaux slow et build_r dues à leurs vitesses respectives (Lemme Rocq 6 et Lemme Rocq 7). Ces bornes permettent d'utiliser les lemmes de propagation.

Ainsi on peut enfin montrer ce qui a été décrit à côté de la Figure 3, et avoir une relation de récurrence sur la position des générateurs (Lemme Rocq 8). On décrit ensuite les configurations jusqu'à la cinquième (Lemme Rocq 1) et montre qu'elle respecte les conditions énoncées (Lemme Rocq 9).

On peut ensuite décrire la position de tous les générateurs en remarquant que :

$$\forall n \in \mathbb{N} \qquad \underbrace{(2n)^2 - \sum_{i=0}^{n} i}_{\text{configuration du générateur}} \quad + \quad \underbrace{3n+1}_{\text{temps entre les deux}} \quad = \quad \underbrace{(2(n+1))^2 - \sum_{i=0}^{n+1} i}_{\text{configuration du générateur suivant}}$$

```
Lemma condition_gen (z : Z) (n : nat) :
  In gen (delta_n regles config_0 n z) <->
  exists m, (m > 1)%nat /\ (n = 2*m*m-(sum_nat m))%nat /\ (Z.
     of_nat (sum_nat m) - 1 = z).
```

4.3.3 Position des wall

Les wall ne peuvent être créés que par un gen, un Init_1 (qui ne se trouve que dans la configuration 0) ou un wall. Connaitre la position des gen permet donc d'obtenir une condition similaire sur les wall :

```
Lemma condition_wall (z : Z) (n : nat) :
  In wall (delta_n regles config_0 n z) <->
  (exists m, (m > 1)%nat /\ (n > 2*m*m-(sum_nat m))%nat /\ (Z.
     of_nat (sum_nat m) - 1 = z))
  \/ ((n > 0)%nat /\ (z = 0)).
```

4.3.4 Position des signaux bounce

Maintenant qu'on a la position des wall et des gen on n'a plus besoin de considérer les siganux slow et build. On s'intéresse maintenant à la sous-construction :

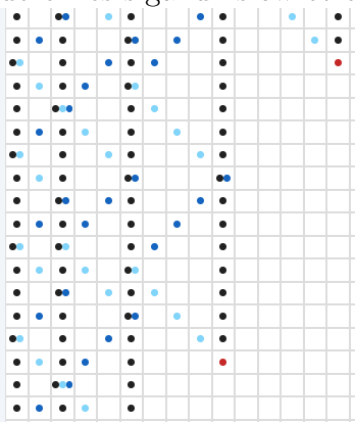

FIGURE 4 – Fischer avec seulement les signaux gen, wall, bounce_l et bounce_r

Les générateurs créent un mur et envoient un signal qui va rebondir entre les deux wall en revenant à sont point de départ tous les $2n$ temps (où n est l'espace entre les deux murs).

Lors de son parcours vers la gauche, ce signal va créer un composite qui va continuer son chemin et marquer la cellulle initiale tous les $2n$ temps (marquant les multiples de n).

De même que pour build_l, on montre que les siganux bounce_l et bounce_r se déplacent respectivement vers la gauche et vers la droite jusqu'à rencontrer un wall (Lemme Rocq 10).

Ensuite on distingue deux cas pour la description des signaux bounce :

- On montre que pour qu'une cellule ait un signal bounce il faut nécessairement qu'elle ou une cellule à sa droite ait eu un signal générateur à un moment antérieur et on traduit cela avec la condition des signaux gen (Lemme Rocq 11).
- On caractérise entièrement les signaux bounce après qu'une cellule à droite ait eu un gen (Lemme Rocq 12)

Avec ces deux lemmes on caractérise entièrement la position des signaux bounce_l :

```
Lemma condition_bounce_l (n m : nat) :
  In bounce_l (delta_n regles config_0 n (Z.of_nat m)) <->
  exists p, (p > 1)%nat /\
```

```
    (Z.of_nat (sum_nat (p - 1)) - 1 <= Z.of_nat m < Z.of_nat (
       sum_nat p) - 1) /\
    (exists q, (n = 2*p*p - sum_nat p + q*2*p + sum_nat p - 1 - m)
       %nat).
```

4.3.5 Position des composite

Les signaux composite ne peuvent être créés que par les signaux Init_1, bounce_l et composite. On s'intéresse donc à la sous-construction suivante :

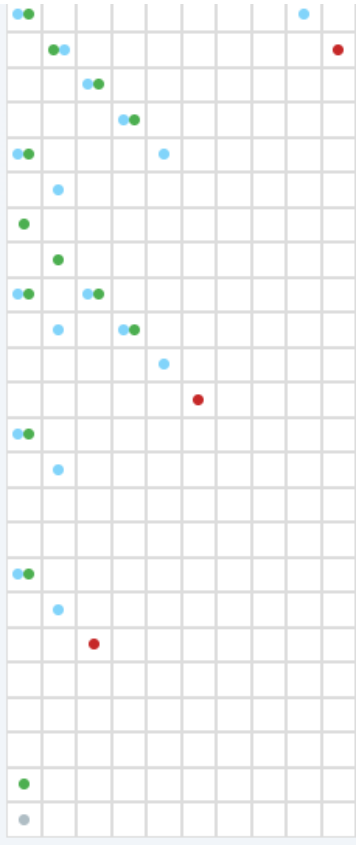

FIGURE 5 – Fischer avec seulement les signaux gen, Init_1, bounce_l et composite

On prouve tout d'abbord un lemme de propagation sur composite (qui se propage vers la gauche sans condition : Lemme Rocq 13).

On peut alors montrer une condition nécessaire et suffisante à la présence d'un composite sur une cellule :

```
Lemma condition_composite (c :
   config) (n : nat) : forall z,
    In composite (delta_n regles c n
        z) <->
    (exists m, (0 < m <= n)%nat /\
    ((In init_1 (delta_n regles c (n
       -m) (z + Z.of_nat m - 1))) \/
    (In bounce_l (delta_n regles c (
       n-m) (z + Z.of_nat m))))
    \/ (In composite (c (z + Z.
       of_nat n))).
```

Lorsqu'on traduit ces conditions avec les règles et la configuration initiale de l'automate de Fischer, à l'aide des lemmes précédents on obtient :

```
Lemma correction_fischer (n : nat) :
    In composite (delta_n regles
       config_0 n 0) <->
    (exists m q, (m > 1)%nat /\ (q
       >= m)%nat /\ (n = 2*q*m - 1)%
       nat)
    \/ (n = 1)%nat.
```

Ainsi pour tout temps $k+1 = 2n$ (qui correspond à k applications de δ), la cellule initiale est marquée si, et seulement si, $2n = 2qm$ (q et m respectant les conditions ci dessus) si, et seulement si, n possède deux diviseurs entiers strictement supérieurs à 1 si, et seulement si, n n'est pas premier.

Références

[1] P. Fischer, *Generation of Primes by a One-Dimensional Real-Time Iterative Array*, Journal of the ACM, 1965.

[2] Marianne Delorme, Jacques Mazoyer, Laure Tougne, *Discrete Parabolas and Circles on 2D Cellular Automata*, Theoretical Computer Science, 1999.

[3] Victor Poupet, *Automates cellulaires : temps réel et voisinages*, PhD thesis, École Normale Supérieure de Lyon, 2006.

[4] Jacques Mazoyer, Nicolas Reimen, *A Linear Speed-Up Theorem for Cellular Automata*, Theoretical Computer Science, 1992.

A - Démonstrations k étapes en une

Démonstration 1 *Par récurrence sur ϕ :*

- $(c,z) \models \overline{\top}^p$ *ssi* $(c,z) \models \top$ *ssi* $(c,p+z) \models \top$ *car* $(c,z) \models \top$ *est vrai pour tout z.*
  De même $(c,z) \models \overline{\bot}^p$ *ssi* $(c,p+z) \models \bot$
- $(c,z) \models \overline{k.s}^p$ *ssi* $(c,z) \models (p+k).s$ *ssi* $s \in c\,(z+p+k)$ *ssi* $(c,z+p) \models k.s$
- $(c,z) \models \overline{\neg\phi}^p$ *ssi* $(c,z) \not\models \overline{\phi}^p \underset{(rec)}{ssi} (c,z+p) \not\models \phi$ *ssi* $(c,z+p) \models \neg\phi$
- $(c,z) \models \overline{\phi \wedge \psi}^p$ *ssi* $((c,z) \models \overline{\phi}^p$ *et* $(c,z) \models \overline{\psi}^p) \underset{(rec)}{ssi} ((c,z+p) \models \phi$ *et* $(c,z+p) \models \psi)$ *ssi* $(c,z+p) \models \phi \wedge \psi$.
  De même $(c,z) \models \overline{\phi \vee \psi}^p$ *ssi* $(c,z+p) \models \phi \vee \psi$

Démonstration 2 *Par récurrence sur ϕ :*

- $(\delta\,(c),z) \models \top$ *ssi* $(c,z) \models \top^{2\text{-}etapes}$ *car* $(c,z) \models \top$ *est vrai pour tout (c,z) et* $\top = \top^{2\text{-}etapes}$.
  De même $(\delta\,(c),z) \models \bot$ *ssi* $(c,z) \models \bot^{2\text{-}etapes}$.
- $(\delta\,(c),z) \models k.s$ *ssi* $s \in \delta\,(c)\,(z+k)$ *ssi* $\exists \psi \in R_s,\ (c,z+k) \models \psi \underset{(Lemme\ 1)}{ssi}$
  $\exists \psi \in R_s,\ (c,z) \models \overline{\psi}^k$ *ssi* $(c,z) \models (s.k)^{2\text{-}etapes}$
- $(\delta\,(c),z) \models \neg\phi$ *ssi* $(\delta\,(c),z) \not\models \phi \underset{(rec)}{ssi} (c,z) \not\models \phi^{2\text{-}etapes}$ *ssi* $(c,z) \models (\neg\phi)^{2\text{-}etapes}$

- $(\delta\,(c), z) \models \phi \wedge \psi$ *ssi* $((\delta\,(c), z) \models \phi$ *et* $(\delta\,(c), z) \models \psi) \underset{(rec)}{\textit{ssi}} ((c, z) \models \phi^{2\text{-}etapes}$ *et* $(c, z) \models \psi^{2\text{-}etapes})$ *ssi* $(c, z) \models (\phi \wedge \psi)^{2\text{-}etapes}$.
  De même $(\delta\,(c), z) \models \phi \vee \psi$ *ssi* $(c, z) \models (\phi \vee \psi)^{2\text{-}etapes}$

Démonstration 3 *Par récurrence sur* k *:*

- $R_0 = R$ *donc* $\delta^1 = \delta = \delta_0$
- *Soit* c *une configuration,* $z \in \mathbb{Z}$ *et* $s \in S$,
  $s \in \delta_{k+1}(c)(z)$ *ssi* $\exists(\psi, s) \in R_{k+1} : (c, z) \models \psi \underset{(def\ de\ R_{k+1})}{\textit{ssi}} \exists(\phi, s) \in R_k :$
  $(c, z) \models \phi^{2\text{-}etapes} \underset{(Lemme\ 2)}{\textit{ssi}} \exists(\phi, s) \in R_k : (\delta(c), z) \models \phi$ *ssi* $s \in \delta_k(\delta(c))(z) \underset{(rec)}{=}$
  $\delta^{k+2}(c)(z)$
  ainsi $\delta_{k+1} = \delta^{k+2}$

B - Enoncés en Rocq des lemmes principaux

```
Lemma description_config_5 :
  In wall (delta_n regles config_0 5 0) /\
  (forall z, z <> 0 ->
    ~(In wall (delta_n regles config_0 5 z))
  ) /\
  In slow_1 (delta_n regles config_0 5 2) /\
  In gen (delta_n regles config_0 5 2) /\
  (forall z, z <> 2 ->
    ~(In slow_1 (delta_n regles config_0 5 z)) /\
    ~(In gen (delta_n regles config_0 5 z))
  ) /\
  In composite (delta_n regles config_0 5 (-4)) /\
  (forall z, z <> - 4 ->
    ~(In composite (delta_n regles config_0 5 z))
  ) /\
  (forall z,
    ~(In init_1 (delta_n regles config_0 5 z)) /\
    ~(In init_2 (delta_n regles config_0 5 z)) /\
    ~(In init_3 (delta_n regles config_0 5 z)) /\
    ~(In slow_2 (delta_n regles config_0 5 z)) /\
    ~(In slow_3 (delta_n regles config_0 5 z)) /\
    ~(In build_r (delta_n regles config_0 5 z)) /\
    ~(In build_l (delta_n regles config_0 5 z)) /\
    ~(In bounce_r (delta_n regles config_0 5 z)) /\
    ~(In bounce_l (delta_n regles config_0 5 z))
   ).
```

1 – Description de la configuration 5

```
Lemma propagation_slow_m_1 (z1 : Z) (c : config) (n : nat) (m :
   nat) :
  (forall z, ~(In init_1 (delta_n regles c n z)) /\ ~(In init_2 (
     delta_n regles c n z)) /\ ~(In init_3 (delta_n regles c n z))
     ) ->
  (forall n0, (n0 < m)%nat -> ~(In build_r (delta_n regles c (n+3*
     n0) (z1 + Z.of_nat n0)))) /\
  In slow_1 (delta_n regles c n z1) /\
  ~(In slow_2 (delta_n regles c n z1)) /\
  ~(In slow_3 (delta_n regles c n z1)) /\
  (forall z2, z1 <> z2 ->
    ~(In slow_1 (delta_n regles c n z2)) /\
    ~(In slow_2 (delta_n regles c n z2)) /\
    ~(In slow_3 (delta_n regles c n z2)))
  ->
  (forall n0, (n0 < m)%nat ->
  In slow_1 (delta_n regles c (n+3*n0+3) (z1 + Z.of_nat n0 + 1))
     /\
  ~(In slow_2 (delta_n regles c (n+3*n0+3) (z1 + Z.of_nat n0 + 1))
     ) /\
  ~(In slow_3 (delta_n regles c (n+3*n0+3) (z1 + Z.of_nat n0 + 1))
     ) /\
  (forall z2, z1 + Z.of_nat n0 + 1 <> z2 ->
    ~(In slow_1 (delta_n regles c (n+3*n0+3) z2)) /\
    ~(In slow_2 (delta_n regles c (n+3*n0+3) z2)) /\
    ~(In slow_3 (delta_n regles c (n+3*n0+3) z2)))).
```

2 – Propagation du signal slow_1 (et conservation du fait qu'il n'y en ait qu'un)

```
Lemma propagation_build_l_m (z : Z) (n m : nat) :
  forall c,
  (forall z, ~(In init_1 (delta_n regles c n z)) /\ ~(In init_2 (
     delta_n regles c n z)) /\ ~(In init_3 (delta_n regles c n z))
     ) ->
  In build_l (delta_n regles c n (z + Z.of_nat m)) /\
  (forall z_prime, (z < z_prime) /\ (z_prime < z + Z.of_nat m + 1)
      ->
    ~(In wall (delta_n regles c n z_prime)) /\
    ~(In gen (delta_n regles c n z_prime))) /\
  (forall z_prime, (z_prime < z + Z.of_nat m + 1) ->
    ~(In slow_1 (delta_n regles c n z_prime)) /\
    ~(In slow_2 (delta_n regles c n z_prime)) /\
    ~(In slow_3 (delta_n regles c n z_prime)))
  ->
  In build_l (delta_n regles c (n+m) z).
```

3 – Propagation du signal build_l

```
Lemma propagation_build_r_m (c : config) (z : Z) (n m : nat) :
  In build_r (delta_n regles c n z) /\
  (forall p, (p < m)%nat -> ~(In slow_1 (delta_n regles c (n+p) (z
      + Z.of_nat p))))
  ->
  In build_r (delta_n regles c (n+m) (z + Z.of_nat m)).
```

4 – Propagation du signal build_r

```
Lemma gen_or_build_r_or_build_l_m (c : config) (n m : nat) :
  (forall z, ~(In init_1 (delta_n regles c n z)) /\ ~(In init_2 (
     delta_n regles c n z)) /\ ~(In init_3 (delta_n regles c n z))
     ) ->
  (exists z,
    ((In gen (delta_n regles c n z) /\ ~(In build_l (delta_n
       regles c n z)) /\ ~(In build_r (delta_n regles c n z))) \/
    (~(In gen (delta_n regles c n z)) /\ In build_l (delta_n
       regles c n z) /\ ~(In build_r (delta_n regles c n z))) \/
    (~(In gen (delta_n regles c n z)) /\ ~(In build_l (delta_n
       regles c n z)) /\ In build_r (delta_n regles c n z))) /\
    forall z_prime, z <> z_prime -> ~(In gen (delta_n regles c n
       z_prime)) /\ ~(In build_l (delta_n regles c n z_prime)) /\
       ~(In build_r (delta_n regles c n z_prime)))
  ->
  (exists z,
    ((In gen (delta_n regles c (n+m) z) /\ ~(In build_l (delta_n
       regles c (n+m) z)) /\ ~(In build_r (delta_n regles c (n+m)
       z))) \/
    (~(In gen (delta_n regles c (n+m) z)) /\ In build_l (delta_n
       regles c (n+m) z) /\ ~(In build_r (delta_n regles c (n+m) z
       ))) \/
    (~(In gen (delta_n regles c (n+m) z)) /\ ~(In build_l (delta_n
        regles c (n+m) z)) /\ In build_r (delta_n regles c (n+m) z
       ))) /\
    forall z_prime, z <> z_prime -> ~(In gen (delta_n regles c (n+
       m) z_prime)) /\ ~(In build_l (delta_n regles c (n+m)
       z_prime)) /\ ~(In build_r (delta_n regles c (n+m) z_prime))
       ).
Proof.
```

5 – Conservation de l'unicité du signal gen, build_l ou bouild_r

```
Lemma slow_isnt_that_slow (z1 : Z) (c : config) (n : nat) (m : nat
   ) :
  (forall z, ~(In init_1 (delta_n regles c n z)) /\ ~(In init_2 (
     delta_n regles c n z)) /\ ~(In init_3 (delta_n regles c n z))
     ) ->
  In slow_1 (delta_n regles c n z1) /\
```

```
  ~(In slow_2 (delta_n regles c n z1)) /\
  ~(In slow_3 (delta_n regles c n z1)) /\
  (forall z2, z1 <> z2 ->
    ~(In slow_1 (delta_n regles c n z2)) /\
    ~(In slow_2 (delta_n regles c n z2)) /\
    ~(In slow_3 (delta_n regles c n z2)))
  ->
  (forall z2, z2 < z1 + Z.of_nat m ->
    ~(In slow_1 (delta_n regles c (n+3*m) z2)) /\
    ~(In slow_2 (delta_n regles c (n+3*m) z2)) /\
    ~(In slow_3 (delta_n regles c (n+3*m) z2)) /\
    ~(In slow_1 (delta_n regles c (n+3*m+1) z2)) /\
    ~(In slow_2 (delta_n regles c (n+3*m+1) z2)) /\
    ~(In slow_3 (delta_n regles c (n+3*m+1) z2)) /\
    ~(In slow_1 (delta_n regles c (n+3*m+2) z2)) /\
    ~(In slow_2 (delta_n regles c (n+3*m+2) z2)) /\
    ~(In slow_3 (delta_n regles c (n+3*m+2) z2))).
```

6 – Borne sur la vitesse de propagation de slow

```
Lemma build_r_cannot_tp (c : config) (z : Z) (n : nat) :
  (forall z, ~(In init_1 (delta_n regles c n z)) /\ ~(In init_2 (
     delta_n regles c n z)) /\ ~(In init_3 (delta_n regles c n z))
     ) ->
  (forall z, ~(In build_l (delta_n regles c n z)) /\  ~(In gen (
     delta_n regles c n z))) /\
  (forall z_prime, z < z_prime -> ~(In build_r (delta_n regles c n
      z_prime)))
  ->
  (forall m z_prime, z + Z.of_nat m < z_prime ->
  ~(In build_r (delta_n regles c (n+m) z_prime)) /\
  ~(In build_l (delta_n regles c (n+m) z_prime)) /\
  ~(In gen (delta_n regles c (n+m) z_prime))).
```

7 – Borne sur la vitesse de propagation de build_r

```
Lemma from_a_gen_to_another (c : config) (z : Z) (n m : nat) :
  (m > 1)%nat ->
  (forall z, ~(In init_1 (delta_n regles c n z)) /\ ~(In init_2 (
     delta_n regles c n z)) /\ ~(In init_3 (delta_n regles c n z))
     ) /\
  (forall z, ~(In build_l (delta_n regles c n z)) /\ ~(In build_r
     (delta_n regles c n z))) /\
  In slow_1 (delta_n regles c n (z + Z.of_nat m)) /\
  ~(In slow_2 (delta_n regles c n (z + Z.of_nat m))) /\
  ~(In slow_3 (delta_n regles c n (z + Z.of_nat m))) /\
  In gen (delta_n regles c n (z + Z.of_nat m)) /\
  (forall z_prime, (z + Z.of_nat m <> z_prime) ->
    ~(In gen (delta_n regles c n z_prime)) /\
```

```
    ~(In slow_1 (delta_n regles c n z_prime)) /\
    ~(In slow_2 (delta_n regles c n z_prime)) /\
    ~(In slow_3 (delta_n regles c n z_prime))) /\
  In wall (delta_n regles c n z) /\
  (forall z_prime, (z < z_prime) ->
    ~(In wall (delta_n regles c n z_prime)))
  ->
  (forall z, ~(In build_l (delta_n regles c (n+3*m + 1) z)) /\ ~(
     In build_r (delta_n regles c (n+3*m + 1) z))) /\
  In slow_1 (delta_n regles c (n+3*m + 1) (z + 2*Z.of_nat m + 1))
     /\
  ~(In slow_2 (delta_n regles c (n+3*m + 1) (z + 2*Z.of_nat m + 1)
     )) /\
  ~(In slow_3 (delta_n regles c (n+3*m + 1) (z + 2*Z.of_nat m + 1)
     )) /\
  In gen (delta_n regles c (n+3*m + 1) (z + 2*Z.of_nat m + 1)) /\
  (forall z_prime, (z + 2*Z.of_nat m + 1 <> z_prime) ->
    ~(In gen (delta_n regles c (n+3*m + 1) z_prime)) /\
    ~(In slow_1 (delta_n regles c (n+3*m + 1) z_prime)) /\
    ~(In slow_2 (delta_n regles c (n+3*m + 1) z_prime)) /\
    ~(In slow_3 (delta_n regles c (n+3*m + 1) z_prime))) /\
  In wall (delta_n regles c (n+3*m + 1) (z + Z.of_nat m)) /\
  (forall z_prime, (z + Z.of_nat m < z_prime) ->
    ~(In wall (delta_n regles c (n+3*m + 1) z_prime))).
```

8 – Construction du générateur suivant et invariants

```
Lemma init_goes_well_for_gens :
  (forall z, ~(In init_1 (delta_n regles config_0 5 z)) /\ ~(In
     init_2 (delta_n regles config_0 5 z)) /\ ~(In init_3 (delta_n
      regles config_0 5 z))) /\
  (forall z, ~(In build_l (delta_n regles config_0 5 z)) /\ ~(In
     build_r (delta_n regles config_0 5 z))) /\
  In slow_1 (delta_n regles config_0 5 (0 + Z.of_nat 2)) /\
  ~(In slow_2 (delta_n regles config_0 5 (0 + Z.of_nat 2))) /\
  ~(In slow_3 (delta_n regles config_0 5 (0 + Z.of_nat 2))) /\
  In gen (delta_n regles config_0 5 (0 + Z.of_nat 2)) /\
  (forall z_prime, (0 + Z.of_nat 2 <> z_prime) ->
    ~(In gen (delta_n regles config_0 5 z_prime)) /\
    ~(In slow_1 (delta_n regles config_0 5 z_prime)) /\
    ~(In slow_2 (delta_n regles config_0 5 z_prime)) /\
    ~(In slow_3 (delta_n regles config_0 5 z_prime))) /\
  In wall (delta_n regles config_0 5 0) /\
  (forall z_prime, (0 < z_prime) ->
    ~(In wall (delta_n regles config_0 5 z_prime))).
```

9 – La configuration 5 respecte les conditions pour la création des générateurs

```
Lemma propagation_bounce_1 (c : config) (n m : nat) :
```

```
  forall z,
  In bounce_l (delta_n regles c n (z + Z.of_nat m)) /\
  (forall p, (0 <= p < m)%nat -> ~(In wall (delta_n regles c (n+p)
      (z + Z.of_nat (m - p)))))
  ->
  In bounce_l (delta_n regles c (n+m) z).

Lemma propagation_bounce_r (c : config) (z : Z) (n m : nat) :
  In bounce_r (delta_n regles c n z) /\
  (forall p, (0 <= p < m)%nat -> ~(In wall (delta_n regles c (n+p)
      (z + Z.of_nat p))))
  ->
  In bounce_r (delta_n regles c (n+m) (z + Z.of_nat m)).
```

10 – Propagation des bounce

```
Lemma no_bounce_l_bounce_r_before_gen_fischer :
  forall m, (m > 0)%nat ->
    forall n z_prime, (2*m*m-(sum_nat m) < n <= 2*(m+1)*(m+1)-(
       sum_nat (m+1)))%nat ->
      ((Z.of_nat (sum_nat m) - 1 <= z_prime) ->
        ~(In bounce_l (delta_n regles config_0 n z_prime))) /\
      ((Z.of_nat (sum_nat m) - 1 < z_prime) ->
        ~(In bounce_r (delta_n regles config_0 n z_prime))).
```

11 – Pas de bounnce sans les générateurs

```
Lemma bounce_l_between_walls (c : config) (z : Z) (n m : nat) :
  (m > 1)%nat ->
  In wall (delta_n regles c n z) /\
  In wall (delta_n regles c n (z+Z.of_nat m)) /\
  In bounce_l (delta_n regles c n (z+Z.of_nat m-1)) /\
  (forall z_prime,
    ((z <= z_prime < z+Z.of_nat m - 1) -> ~(In bounce_l (delta_n
       regles c n z_prime))) /\
    ((z < z_prime <= z+Z.of_nat m) ->
      ~(In bounce_r (delta_n regles c n z_prime)) /\ (forall p, ~(
         In gen (delta_n regles c (n+p) z_prime)))
    ) /\
    ((z < z_prime < z+Z.of_nat m) -> forall p, ~(In wall (delta_n
       regles c (n+p) z_prime)))
  )
  ->
  (forall q p,
    ((0 <= p < m)%nat -> (
      In bounce_l (delta_n regles c (n+q*2*m+p) (z + Z.of_nat m -
         1 - Z.of_nat p)) /\
      (~(In bounce_r (delta_n regles c (n+q*2*m+p) (z + Z.of_nat m
          - 1 - Z.of_nat p)))
```

```
      \/ (z + Z.of_nat m - 1 - Z.of_nat p = z)) /\
     (forall z_prime, (z_prime <> (z + Z.of_nat m - 1 - Z.of_nat
        p)) ->
       ((z <= z_prime < z+Z.of_nat m) -> ~(In bounce_l (delta_n
          regles c (n+q*2*m+p) z_prime))) /\
       ((z < z_prime <= z+Z.of_nat m) -> ~(In bounce_r (delta_n
          regles c (n+q*2*m+p) z_prime)))
     )
   )) /\
   ((m <= p < 2*m)%nat -> (
     (~(In bounce_l (delta_n regles c (n+q*2*m+p) (z + Z.of_nat p
         - Z.of_nat m + 1)))
      \/ (z + Z.of_nat p - Z.of_nat m + 1 = z + Z.of_nat m)) /\
     In bounce_r (delta_n regles c (n+q*2*m+p) (z + Z.of_nat p -
        Z.of_nat m + 1)) /\
     (forall z_prime, (z_prime <> (z + Z.of_nat p - Z.of_nat m +
        1)) ->
       ((z <= z_prime < z+Z.of_nat m) -> ~(In bounce_l (delta_n
          regles c (n+q*2*m+p) z_prime))) /\
       ((z < z_prime <= z+Z.of_nat m) -> ~(In bounce_r (delta_n
          regles c (n+q*2*m+p) z_prime)))
     ))
   )
 ).
```

12 – Caractérisation des bounce entre les murs à partir d'un générateur

```
Lemma propagation_composite (c : config) (z : Z) (n m : nat) :
  In composite (delta_n regles c n z) -> In composite (delta_n
     regles c (n+m) (z - Z.of_nat m)).
```

13 – Propagation du signal composite