UNIVERSITÉ D'ORLÉANS

ÉCOLE DOCTORALE MATHÉMATIQUES, INFORMATIQUE, PHYSIQUE THÉORIQUE ET INGÉNIERIE DES SYSTÈMES

Laboratoire d'Informatique Fondamentale d'Orléans

Thèse présentée par :

Diego MALDONADO

soutenue le : **version soutenance 26 novembre 2018**

pour obtenir le grade de : **Docteur de l'Université d'Orléans**

Discipline/ Spécialité : **Informatique**

Universalité et complexité des automates cellulaires coagulants

Thèse dirigée par :

Eric GOLES	Professeur, Universidad Adolfo Ibañez
Nicolas OLLINGER	Professeur, Université d'Orléans

RAPPORTEURS :

Julien CERVELLE	Professeur, Université Paris-Est Créteil
Sylvain SENE	Professeur, Aix-Marseille Université

JURY :

Julien CERVELLE	Professeur, Université Paris-Est Créteil
Eric GOLES	Professeur, Universidad Adolfo Ibañez
Ines KLIMANN	Maître de conférences, Université Paris Diderot
Nicolas OLLINGER	Professeur, Université d'Orléans
Nicolas SCHABANEL	Directeur de recherche, CNRS
Sylvain SENE	Professeur, Aix-Marseille Université
Véronique TERRIER	Maître de conférences, Université de Caen Basse Normandie

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Automates cellulaires

Dans les années 40, John von Neumann, à la recherche d'un système théorique capable de modéliser des systèmes naturels auto-reproductifs, suit les conseils de Stanislaw Ulam et décide d'étudier un système dynamique local, discret et synchrone que nous appelons désormais *automate cellulaire*. Il s'agit d'un réseau régulier, généralement une grille de dimension d, où chaque cellule a un nombre fini d'états possibles qui lui sont associés. Ces cellules changent d'état simultanément selon un *règle* qui dépend des états des cellules voisines et qui est le même pour toutes les cellules. Les voisinages les plus utilisés en dimension un sont :

- le voisinage des *premiers voisins*, il se compose de la cellule centrale, de la cellule gauche et de la cellule droite ; et
- le voisinage à *sens unique*, il se compose de la cellule centrale et de la cellule gauche ou de la cellule droite.

et en dimension deux :

- le voisinage *de von Neumann*, il consiste en la cellule centrale, la cellule nord, la cellule sud, la cellule est et la cellule ouest ; et
- le voisinage *de Moore*, il s'agit du voisinage de von Neumann ajoutant la cellule nord-est, nord-ouest, sud-est et sud-ouest.

Il est possible d'étendre les voisinages précédents à des dimensions supérieures selon le même principe.

De manière synthétique, les automates cellulaires sont :

- discrets dans le temps et dans l'espace,
- homogènes dans le temps et dans l'espace (la manière de changer est la même pour chaque cellule à chaque fois),
- locaux et synchrones dans leurs interactions.

Nous obtenons ainsi des systèmes dynamiques qui montrent un comportement très complexe à partir de règles très simples. L'exemple le plus simple est la famille des *automates cellulaires élémentaires*. C'est la famille des automates cellulaires dans la dimension un, deux états et le voisinage des premiers voisins et chaque automate cellulaire dans cette famille est représenté par le nombre $\sum_{i=0}^{7} f([\![i]\!])2^i$, où f est la règle à trois voisins et $[\![i]\!]$ est la codification binaire á trois chiffres de i, de $[\![0]\!] = 000$ à $[\![7]\!] = 111$. Ces règles sont au nombre de 256, dont 88 sont deux à deux distinctes.

C'est dans cette famille que l'on trouve la règle 110 qui est capable de simuler n'importe quelle machine-ordinateur, la règle 184 simulant un flux de trafic et la règle 30, une règle au comportement «chaotique».

Les automates cellulaires ont été étudiés sous de nombreux angles : comme environnement pour les phénomènes physiques, chimiques, sociaux entre autres, comme systèmes dynamiques discrets, comme modèle de calcul ou simplement pour le plaisir.

Ainsi, l'automate le plus exploré en dimension deux est sans conteste le *Jeu de la Vie* [5] introduit par John H. Conway en 1970. C'est un automate cellulaire défini sur la grille bidimensionnelle avec deux états pour les cellules ($\{0, 1\}$ ou $\{\text{mort}, \text{vivant}\}$), où chaque cellule considère son voisinage Moore. La règle pour chaque cellule est la suivante

- Toute cellule vivante ayant moins de deux voisins vivants meurt, comme si elle était sous-peuplée.
- Toute cellule vivante avec deux ou trois voisins vivants survit jusqu'à la prochaine génération.
- Toute cellule vivante ayant plus de trois voisins vivants meurt, comme si elle était surpeuplée.
- Toute cellule morte ayant exactement trois voisins vivants devient une cellule vivante, comme par reproduction.

Cette règle simple montre un comportement très complexe, par exemple nous pouvons trouver un ensemble fini de cellules vivantes se déplaçant sur le plan appelé *Gliders*, d'autres ayant un comportement périodique celluled *Pulsars* et d'autres encore sont capables de reproduire des calculs universels.

Modélisation

Les automates cellulaires comme modèle du monde réel partagent, par définition, des propriétés importantes avec les lois physiques classiques :

- les règles, de la même manière que les lois physiques, sont uniformes dans le temps et dans l'espace, c'est-à-dire que peu importe les coordonnées de la cellule sur la grille, la règle est la même,
- est défini par un réseau régulier, de la même manière que les lois physiques sont définies uniformément sur l'espace.

Il y a aussi des automates cellulaires avec d'autres propriétés physiques intéressantes, comme les automates cellulaires dont le comportement est *réversible*, c'est-à-dire un automate cellulaire inversible dont l'inverse est aussi un automate cellulaire. Ce sont des automates cellulaires qui préservent pleinement l'information. Le modèle HPP [6, 7], nommé d'après les initiales des noms de ses auteurs : Hardy, Pomeau et de Pazzis, est un bon exemple d'automate cellulaire réversible. Il s'agit d'un modèle de gaz au niveau microscopique, où chaque particule dans le gaz est représentée par une flèche représentant la direction des particules et se trouve dans une cellule sur une grille bidimensionnelle. Chaque cellule ne peut contenir qu'un maximum d'une particule pour chaque direction, c'est-à-dire qu'elle contient au total entre zéro et quatre particules et, facultativement, nous ajoutons des cellules "paroi" pour contenir les particules.

Les règles suivantes régissent le modèle :

- Une seule particule se déplace dans une direction fixe jusqu'à ce qu'elle entre en collision avec une autre.
- Deux particules subissant une collision frontale sont déviées perpendiculairement.
- Deux particules subissent une collision qui n'est pas frontale, se traversent et continuent dans la même direction.
- Lorsqu'une particule entre en collision avec les parois, elle va rebondir.

Ici nous pouvons obtenir un automate cellulaire inverse en déplacent les particules dans le sens inverse.

Une autre propriété importante des automates cellulaires réversibles est leur implémentation de hardware, qui peut - théoriquement - être plus efficace énergétiquement que les implémentations des systèmes irréversibles utilisés de nos jours. Selon le principe de Landauer [8], l'énergie de dissipation minimale pour l'effacement d'un bit d'information à température absolue de T est au moins de $kT \ln 2$ Joule d'énergie, où k est la constante de Bolzmann. Cela finira par devenir prohibitif pour la miniaturisation et l'emballage de plus en plus dense de portes irréversibles telles que AND et OR. Le calcul réversible a été proposé comme alternative où aucun bit n'a besoin d'être effacé, évitant ainsi la limite inférieure ($kT \ln 2$) de la dissipation d'énergie par bit.

Une autre classe d'automates cellulaires est la classe des *automates cellulaires conservateurs*, où il existe une fonction des états aux nombres réels, appelée *quantité additive*, et pour toute configuration où l'addition de la quantité additive de son état est un nombre et ce nombre ne change pas après avoir appliqué l'automate cellulaire conservatif. Cette classe d'automates cellulaires s'intéresse à la physique comme source de modèles de systèmes de particules régis par les lois de conservation de la masse et/ou de l'énergie, en particulier les automates cellulaires de flux de trafic (règle 184), le flux fluide, les alliages eutectiques, les échanges entre individus, etc. et englobe le modèle HPP où la quantité additive de cellule représente le nombre de flèches dans cette cellule.

Dynamique discrète et topologie

Dans l'espace cellulaire, nous pouvons définir une *métrique* entre les configurations c et c' comme suit : $d(c, c') = 0$ si $c = c'$ et $d(c, c') = 2^{-\min\{\|z\|: c_z \neq c'_z\}}$, satisfaisant aux trois propriétés élémentaires d'une métrique : d est positive, $d(c, c') = 0$ si et seulement si $c = c'$, d est symétrique, et d satisfait à l'inégalité triangulaire. La valeur $d(c, c')$ s'appelle *distance* entre c et c'.

Toute paire d'un ensemble et d'une métrique avec ces propriétés précédentes est appelée *espace métrique*, en particulier l'espace cellulaire est un espace métrique. Avec notre définition de la distance, nous pouvons caractériser les automates cellulaires comme les fonctions continues qui commute avec la fonction *shift* dans toutes les directions, où la fonction shift est la fonction qui déplace chaque cellule dans une position dans une direction. L'espace cellulaire est compact et est généré par le *cylindres* Cyl (c, D) un ensemble de toutes les configurations qui sont égales à la configuration c cellule par cellule sur les cellules sur $D \subseteq \mathbb{Z}^d$. Chaque cylindre est topologiquement et ouvert et fermé simultanément.

Par compacité, tout automate cellulaire inversible est réversible. De même, toute fonction continue sur un espace compact définit un *système dynamique*, puis un automate cellulaire définit un système dynamique sur l'espace de configuration.

Dans le système dynamique est étudié le *ensembles de limites*, l'ensemble de toutes les configurations un t-ème pré-images pour tout $t \in \mathbb{N}$. Dans les automates cellulaires, l'ensemble des limites est un *sous-shift*, c'est-à-dire qu'il est fermé non vide et shift-invariant.

Complexité

Le point de vue informatique est basé sur la thèse de Church-Turing, dans laquelle tous les modèles de calcul "raisonnables" sont équivalents, c'est-à-dire que chacun peut calculer les mêmes choses. Parmi ces modèles de calcul figurent le lambda-calcul [9], tag systèmes [10], machines à compteurs [11] et machines de Turing [12], parmi d'autres, et nous disons qu'il s'agit de modèles *universels pour le calcul* (aussi appelés Turing universels).

Universalité Turing

Dès le début, l'universalité pour le calcul apparaît dans les automates cellulaires, comme les automates cellulaires auto-reproductifs de von Neumann [13] simulant l'autoreproduction de systèmes biologiques. Von Neumann construit un automate cellualire à 29 état, deux dimensions et voisinage von Neumann simulant une machine de Turing qui contrôle le bras du constructeur qui construit la copie de la configuration originale. Les chercheurs ont cherché des automates de plus en plus simples, avec moins d'états, de dimensions et de voisins, qui montrent une universalité pour le calcul. Smith (1971) [14] a montré que 18 couleurs et premières voisines des règles unidimensionnelles pouvaient être universelles, et Lindgren et Nordahl (1990) [15] ont construit un automate cellulaire universel 7 couleurs pour les premiers voisins, mais le cas le plus important est celui de Cook (2004) [16] où, en simulant un système de balises sur l'automate cellulaire élémentaire 110, il montre que deux états et les premier voisin suffisent pour obtenir l'universalité pour le calcul.

Universalité intrinsèque

L'idée de complexité dans les automates cellulaires donnée par sa capacité à simuler les machines de Turing ne semble pas être tout à fait pratique.

Alors que les machines de Turing sont des modèles informatiques séquentiels et finis, les automates cellulaires sont un modèle extrêmement parallèle et spatialement infini. C'est ainsi qu'est née la notion d'universalité intrinsèque d'un automate cellulaire, qui est la capacité d'un automate cellulaire à simuler tout autre automate cellulaire. Quelques exemples d'automates intrinsèquement universels sont les automates unidimensionnels d'Albert et Culik avec 14 états et premiers voisinages [17], le jeu de la vie de Conway [18] et le automates cellulaire unidimensionnel d'Ollinger et Richard avec 4 états et premiers voisins [19] . Le dernier est, à notre connaissance, l'automate cellulaire avec le moins d'états intrinsèquement universels en dimension 1 et le voisinage des premiers voisins. Restez ouvert pour savoir si la règle 110 est intrinsèquement universelle.

L'étude de l'universalité intrinsèque ne peut pas seulement être explorée à travers la réduction des états des automates intrinsèquement universels, il est également intéressant de voir quelles propriétés supplémentaires nous pouvons exiger et nous continuons à trouver des automates intrinsèquement universels. Si nous ajoutons la conservation, il est possible de trouver un automate intrinsèquement universel qui est aussi conservateur [20] . Ce n'est pas vrai avec la réversibilité, car les automates réversibles ne peuvent simuler que des automates réversibles. Ainsi commence l'étude de l'universalité intrinsèque seulement pour les automates cellulaires réversibles, où il y a un automate cellulaire intrinsèquement universel pour les automates réversibles [21] . De plus, il en existe aussi un qui est réversible et conservateur [22] et même conservateur à symétrie temporelle [1] (la symétrie temporelle est une sous-classe des automates cellulaires réversibles, pour la définition formelle et les propriétés voir [23]).

Bien que chacun des auteurs précédents considère sa propre notion de la simulation, y compris la simulation cellule par cellule, cellule par macro-cellule, échelle temporelle ou même translation spatiale, ceci est formalisé dans [24, 25] qui regroupe tous ces cas dans une seule définition.

Complexité des calculs

Un autre aspect calculatoire très étudié est la complexité de calcul, qui compte le nombre maximum d'opérations effectuées par une machine de Turing pour effectuer un calcul en fonction de la taille de l'entrée, c'est-à-dire le temps (ou le nombre d'opérations) utilisé pour résoudre le pire des cas. Cette idée de la mesure du temps, ainsi que celle de la mesure spatiale, a été introduite par Hartmanis et Stearns [26]. Edmonds propose comme problèmes qui peuvent être efficacement calculés à ceux dont la complexité peut être limitée par un polynôme sur la taille de l'entrée, appelés problèmes polynomiaux ou problèmes dans la classe **P**. Edmonds donne également une description informelle pour le temps polynomial sur les machines de Turing non déterministes [27], ouvrant l'une des questions les plus importantes dans la complexité des calculs, est $\mathbf{P} = \mathbf{NP}$? et plus tard formalisé par Cook [28] et Levin [29] .

Il existe également une classe de complexité pour les ordinateurs parallèles avec un grand nombre de processeurs, appelée classe **NC**, qui correspond au type de problèmes qui peuvent être résolus en temps poly-logarithmique en utilisant un nombre polynomial de processeurs. Il porte le nom de Nick Pippenger pour ses études sur les circuits à profondeur poly-logarithmique et nombre polynomial de portes logiques. Il s'agit d'une sous-classe de **P**, c'est-à-dire $\mathbf{NC} \subseteq \mathbf{P}$ et reste ouverte pour savoir si $\mathbf{NC} = \mathbf{P}$ est disponible.

Pour **P** et **NP**, il existe un sous-ensemble de problèmes appelés **P**-complet et **NP**-complet respectivement. Dans ces problèmes, n'importe quel problème dans **P** (**NP**) peut lui être réduit en temps poly-logarithmique en utilisant un nombre polynomial de processeurs (temps polynomial utilisant un processeur), où réduire le problème A à B c'est-à-dire transformer les entrées de A en entrées pour B par une fonction calculable en temps **NC** (**P**) tel qu'une entrée de A soit *accepté* si et seulement si la transformation de cette entrée est *accepté* par B .

Donc si un problème **P**-complet (**NP**-complet) est aussi dans **NC** (**P**), alors les deux classes s'effondrent, c'est-à-dire $\mathbf{P} = \mathbf{NC}$ ($\mathbf{P} = \mathbf{NP}$).

En supposant que $\mathbf{P} \neq \mathbf{NP}$, alors les problèmes **NP**-complets peuvent être résolus "effectivement" dans une machine séquentielle . De même, il est largement admis que $\mathbf{P} \neq \mathbf{NC}$ c'est-à-dire qu'il existe des problèmes polynomiaux qui ne peuvent pas améliorer exponentiellement leurs performances en ajoutant un nombre raisonnable de processeurs (polynômes), ce qui signifie qu'ils ne peuvent être "efficacement" mis en parallèle, puis les problèmes **P**-complètes sont les candidats pour être les problèmes non parallélisés. Les problèmes complets sont les plus complexes de sa classe, parce que n'importe quel autre problème dans la classe peut être résolu par ceux-ci par une réduction en préservant la complexité du premier problème.

Une façon d'étudier les automates cellulaires de ce point de vue est par le problème de décision suivant : Décider si, étant donné une configuration, un état et un temps, la cellule à l'origine de la grille est dans l'état donné au temps donné. Ce problème est connu sous le nom de problème Prediction, et il est polynomial, en simulant simplement la règle des automates. Si nous trouvons un algorithme "vite" pour Prediction (plus vite que **P**), alors nous pouvons obtenir des itérations d'automates beaucoup plus rapides que l'itération cellule par cellule, simplement en "prédisant" la valeur de chaque cellule. L'un des premiers à étudier ce problème a été Moore [30, 31] montrant que les automates majoritaires et la vie sans mort sont à la fois **P**-complets. Ils peuvent simuler des circuits logiques dans une configuration, de sorte qu'il est possible de résoudre le problème de valeur de circuit en prédisant la valeur de la cellule de sortie du circuit dans la configuration. Le problème de la valeur du circuit est **P**-Complet. Donc, même dans le cas $\mathbf{NC} = \mathbf{P}$, il n'y aurait pas de méthode plus efficace pour calculer les itérations que l'itération des automates. Il est également important de souligner l'automate cellulaire élémentaire 110, qui est le seul automate élémentaire pour lequel nous savons jusqu'à présent que Prediction est **P**-Complet[32].

Automates cellulaires coagulants

C'est la principale famille d'automates cellulaires dans ce travail, les *Automates cellulaires coagulants*, introduits dans [33]. Ce sont des automates cellulaires où l'ensemble des états est partiellement ordonné et les cellules ne peuvent passer d'un état à un autre qu'en respectant cet ordre partiel. Cette définition induit un nombre limité de changements par cellule, si chaque cellule peut changer jusqu'à k fois, on parle d'automate cellulaire à k-changements.

Le nom "coagulant" vient de la propriété qu'en automate cellulaire coagulant, toute configuration se termine asymptotiquement à un point fixe, c'est-à-dire qu'à long terme, la configuration est «coagulée». A partir de cette propriété, nous intuitions rapidement l'irréversibilité de ces automates cellulaires. En fait, le seul automate cellulaire coagulant réversible (ou même subjectif) est la fonction d'identité [33].

Ce comportement coagulant est observé dans des phénomènes naturels tels que la percolation Bootstrap [34], modélisant les phénomènes physiques suivants [34] :

> *"Consider a pure magnetic system in which the exchange forces on a given spin from its neighbours are barely strong enough to overcome the crystal field. The introduction of non-magnetic impurities would then clearly have a strong effect: if the number of magnetic neighbours is sufficiently reduced, a spin with a magnetic moment can become non-magnetic by being forced into the singlet state."*

> *"Considérez un système magnétique pur dans lequel l'échange de forces sur une rotation donnée à partir de son les voisins sont à peine assez forts pour surmonter le champ de cristal. L'introduction des impuretés amagnétiques auraient alors clairement un fort effet : si le nombre de voisins magnétiques est suffisamment réduit, un spin avec un moment magnétique peut devenir amagnétique en étant forcé à l'état de singulet."*

Ce problème peut être modélisé par un automate cellulaire coagulant avec états $\{0, 1\}$. (0 : magnétique, 1 : non magnétique) suivant la règle locale : $f(c) = 1$ si $c_0 = 1$, $f(c) = 1$ si $c_0 = 0$ et $\sum_{i=0}^{n} c_i > \theta$ et $f(c) = 1$ sinon, où n est le nombre de voisins de la cellule à changer et θ est le seuil des cellules non magnétiques nécessaire pour rendre la cellule non magnétique.

D'autres modèles sont les modèles de feux de forêt [35] ou le modèle SIR [36] de propagation de l'infection (Susceptible, Infected and Recover and acquire immunity), mais on observe aussi d'autres modèles théoriques qu'ils sont "coagulé" comme le modèle d'assemblage abstrait de tuiles (aTAM) [37] ou la vie sans mort [30], où la complexité du problème Prediction a été démontrée.

Ces automates cellulaires en dimension un sont moins complexes que ceux qui ne sont pas congelés, car Prediction est au maximum NLOGSPACE (NLOGSPACE $\subseteq$ **NC**) alors que Prediction est **P**-complet en général.

Dans cette thèse, nous étudions les automates cellulaires coagulants et montrons, à l'exception du cas des automates cellulaires de dimension un, que malgré les limites apparentes dues au fait que les états ne peuvent avancer et donc ne peuvent changer qu'un nombre fini de fois, ceux-ci montrent une grande complexité, tant du point de vue de l'universalité intrinsèque que de la complexité calculatoire.

Dans le cas de l'universalité intrinsèque, nous montrons d'abord qu'il n'existe pas d'automates intrinsèquement universels pour les automates cellulaires coagulants selon la définition donnée dans [24, 25]. Une définition de simulation un peu plus flexible est donc nécessaire. Étant donnée la bonne définition de la simulation, nous montrons que même dans ce contexte, il n'est pas possible de trouver un automate cellulaire intrinsèquement universel dans la dimension un, nous passons donc à l'étude dans la dimension 2 ou plus. Pour la dimension deux, nous montrons qu'il n'est pas possible de trouver un automate cellulaire intrinsèquement universel avec voisinage de von Neumann pour les automates cellulaires coagulants où les cellules peuvent changer au plus une fois, mais cela est possible avec deux changements ou plus par cellule. Nous montrons explicitement un qui est intrinsèquement universel pour les automates cellulaires coagulants avec deux changements et le voisinage de von Neumann. Nous explorons peu de changements, la forme du voisinage et les dimensions pour trouver les automates cellulaires coagulants intrinsèquement universels les plus simples.

D'autre part, pour étudier la complexité des calculs, nous avons exploré la famille des automates cellulaires coagulants dans le voisinage de von Neumann, totalistiques et à deux états, sur des configurations périodiques. Puisque les automates cellulaires coagulants atteignent toujours un point fixe dans un temps fini, alors nous étudierons une variante de Prediction, appelée Stability, dans laquelle la question est de savoir si une cellule de l'automate cellulaire coagulant aura la même valeur au point fixe atteint ou non. Ce problème est **P**, parce que le point fixe est atteint dans un temps polynomial. Il reste donc à voir si l'on peut faire mieux. En résolvant Stability cellule par cellule, il est possible de calculer le point fixe atteint par l'automate coagulant en un temps polynomial ou plus rapidement si on trouve des algorithmes plus efficaces pour Stability.

Nous avons montré que pour cette famille d'automates cellulaires, il y a des problèmes **P**-complets, pour lesquels nous ne pouvons pas améliorer significativement le temps d'exécution, sauf si **P** = **NC** . Nous avons également constaté que la plupart des automates cellulaires de cette famille ont une complexité moindre, la stabilité est en **NC**, sauf dans les cas triviaux, où sa complexité est encore plus faible.

Nous terminons l'étude de la complexité des calculs en répétant l'étude précédente mais en utilisant des schémas de mise à jour des séquences, c'est-à-dire que les cellules changent une à une suivant un ordre préétabli. Pour trouver la complexité maximale du problème Stability dans ce cas (**NP**) nous devons recourir à la troisième dimension et nous en montrons une avec cette complexité. Le problème de savoir si les automates où Stability est **P**-complet dans le cas de schémas de mise à jour parallèles est **NP**-complet dans le cas séquentiel est toujours ouvert.

Nous montrons donc que, sauf dans le cas de la dimension un, où il n'y a pas d'automates cellulaires coagulants intrinsèquement universels ou où la complexité de calcul est au mieux NLOGSPACE, le reste des automates coagulants cellulaire a une complexité élevée. Qu'il s'agisse d'automates intrinsèquement universels ou d'automates cellulaires coagulants où le problème Stability atteint son maximum complété avec très peu d'états et de très petits voisinages.

Les résultats précédents sont répartis dans les chapitres comme suit :

Chapter 3

Le chapitre suivant aborde les définitions élémentaires des automates cellulaires et les éléments généraux de topologie, la théorie des graphes des systèmes dynamiques et quelques extensions de la notion habituelle d'automate cellulaire. Il comprend également une section sur la complexité des calculs parallèles et séquentiels, ainsi qu'une brève description des algorithmes **NC** qui seront utilisés dans ce travail. Ce chapitre se termine par les définitions de la simulation standard dans les cellules et des exemples d'automates cellulaires intrinsèquement universels.

Chapter 4

Le chapitre suivant présente le sujet des automates cellulaires coagulants. Après avoir expliqué la nature irréversible de ces automates cellulaires du point de vue des systèmes dynamiques, il est montré que le problème Stability dans automates cellulaires coagulants est moins complexe dans le dimension 1 que dans les dimensions supérieures, de plus, en passant à la dimension 2, le problème atteint la complexité informatique maximale. En outre, dans la dimension 2, il y a déjà un automate cellulaire coagulant Turing universel. Ceci est d'abord démontré par l'encodage de machines à compteurs Turing universelles dans un automate cellulaire coagulant. Le chapitre se termine par les propriétés des automates cellulaires monotones coagulants et leur relation avec le schéma de mise à jour asynchrone.

Chapter 5

Notre première contribution traite d'un automate coagulant cellulaire intrinsèquement universel, en d'autres termes, un automate cellulaire coagulant capable de simuler tout autre automate cellulaire coagulant, et commence par un résultat négatif : il n'y a pas de tels automates avec la définition habituelle de la simulation, il est donc nécessaire de relâcher un peu la définition de la simulation. Maintenant, au lieu de simuler une cellule par une macrocellule (bloc de cellules) qui ne dépend que de la cellule simulée, nous allons simuler une cellule par une macrocellule qui dépend de la cellule simulée et son voisinage. Etablissant la bonne notion de simulation, nous avons construit un automate cellulaire coagulant intrinsèquement universel avec le voisinage de von Neumann. Les cellules peuvent changer deux fois et fonctionner en stockant les états dans des câbles et en envoyant cette information aux voisins. La simulation s'effectue en calculant la fonction locale à l'aide de circuits logiques. Ce résultat est le meilleur en ce qui concerne le nombre de changements par cellule dans la dimension deux. Pour ce qui précède, nous montrons d'abord que les automates cellulaires coagulants avec un changement et voisinage von Neumann ne peuvent pas croiser les signaux, ce qui est possible avec deux changements. Essayer de construire un automate cellulaire coagulant avec un seul changement qui peut simuler un autre automate cellulaire coagulant qui traverse des signaux mène à une contradiction. Cette limitation est perdue dans la dimension trois ou plus, où l'on peut utiliser la troisième dimension pour croiser l'information. Dans ce cas nous trouvons donc un automate cellulaire coagulant intrinsèquement universel avec un seul changement.

De plus, en utilisant l'automate cellulaire coagulant qui active une cellule avec exactement deux voisins actifs (règle 2 du chapitre 6), nous trouvons que notre automate celluaire coagulant universel ne possède que deux états. Le dernier résultat de ce chapitre est une caractérisation des automates cellulaires qui peut être simulée par un automate cellulaire coagulant. Ce sont celles que l'on peut définir comme étant celles qui diminuent une énergie locale.

Chapter 6

Notre deuxième contribution est l'étude du point de vue de la complexité calculatoire du problème Stability dans la famille des automates cellulaires totalistiques figés à deux états (actif et inactif) et considérant deux grilles différentes : grille à cellules triangulaires (trois voisins par cellule) et grille à cellules carrées (quatre voisins par cellule). Au total, il y a 16 règles pour la grille triangulaire et 32 règles pour la grille carrée, mais au départ nous ne considérons que les règles où les cellules inactives sont quiescentes, laissant dans chaque cas la moitié des règles à étudier. Nous désignons cette règle par un nombre, où chaque chiffre signifie que la règle active les cellules avec exactement ce nombre de voisins actifs. Dans les règles de la grille triangulaire, la chose la plus complexe que nous avons trouvée était des règles avec une complexité en **NC**, alors que dans la grille carrée, nous avons également trouvé des règles **P**-complètes à savoir 2 et 24.

Chapter 7

Enfin, nous étudions la même famille de règles que dans le chapitre précédent, mais en considérant les mise à jour asynchrones, où chaque cellule change à un moment différent des autres. Avec cette simple modification du schéma de mise à jour, nous étudions le problème AsyncStability, où une cellule est stable si pour tout schéma de mise à jour choisi, la cellule reste inactive. Nous ne trouvons ici que les règles sont **NC**, à l'exception des règles 2 et 24 de la grille carrée, qui sont les règles **P**-complètes du chapitre précédent, où le problème reste ouvert. En en dimension 3, c'est lorsque ces règles deviennent **NP**-complètes.

Chapter 8

Le dernier chapitre conclut que malgré les limites apparentes des automates cellulaires coagulants, ils présentent un comportement très intéressant. Si nous considérons le voisinage de von Neumann dans la dimension deux, avec deux états, nous ne pouvons trouver qu'un automate cellulaire coagulant **P**-complet, mais si nous permettons plus de ceux-ci, mais seulement deux changements, nous obtenons un automate cellulaire coagulant, intrinsèquement universel. La question reste ouverte de savoir s'il est possible de le faire avec seulement trois états (le nombre minimum de cellules pour obtenir deux changements), ce qui est déjà possible dans des dimensions supérieures avec seulement deux cellules.

Chapter 2

Introduction

Cellular automata

During the 40's John von Neumann in his search for a theoretical system capable of modeling natural auto-repoductive systems follows the advice of Stanislaw Ulam and decides to study a local dynamic system, discreet and synchronous that we now know as *cellular automata*. It consists of a regular network, generally a d dimensional grid, where each cell has a finite number of possible states associated with it. These cells change states simultaneously according to a *rule* that depends on the states of neighbors cells and that is the same for all the cells. The more used neighbors in dimension one are:

- the *first neighbors* neighborhood, it consists in the center cell, the left cell and the right cell; and
- the *one-way* neighborhood, it consists in the center cell and the left cell or the right cell.

and in dimension two they are:

- the *von Neumann* neighborhood, it consists in the center cell, the north cell, south cell, east cell and west cell; and
- the *Moore* neighborhood, it consists in von Neumann neighborhood adding the north-east cell, north-west , south-east cell and south-west cell.

It is possible to extend this previous neighborhood to higher dimensions following the base idea.

In short, the cellular automata are

- discrete in both time and space,
- homogeneous in both time and space (the way to change is the same for each cell for each time),
- local and synchronous in its interactions.

Thus we obtain dynamical systems that show a very complex behavior from very simple rules. The most simple example is the family of *elementary cellular automata*. This is the family of cellular automata in dimension one, two states and first neighbors neighborhood and each cellular automata in this family is represented by the number $\sum_{i=0}^{7} f([\![i]\!])2^i$, where f is the three neighbors rule and $[\![i]\!]$ is the three digits binary codification of i, from $[\![0]\!] = 000$ to $[\![7]\!] = 111$. This rules are 256, which 88 are inequivalent.

From there are studied the rule 110 as a rule able to simulate any computer machine, the rule 184 simulating a traffic flow and the rule 30 as a rule with a chaotic behavior.

Thus, cellular automata have been studied from many points of view: as an environment for the physical, chemical, social phenomena among others, as discrete dynamic systems, as model of computation or simply for fun.

About fun with cellular automata, the most explored one in dimension two is the *Game of Life* [5], introduced by John H. Conway in 1970. This is a cellular automata defined on the two-dimensional grid with two state for the cells ($\{0, 1\}$ or $\{\text{die}, \text{live}\}$), where each cell considers its Moore neighborhood and the following rule for each cell

- Any live cell with fewer than two live neighbors dies, as if by underpopulation.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by overpopulation.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

This simple rule shows a very complex behavior, as example we can find a finite set of live cells moving on the plane called *Gliders*, others with a periodical behavior celled *Pulsars* and others are able to reproducing universal computation.

Modeling

The cellular automata as model of the real world shares, by definition, important properties with the classical physical laws:

- the rules, in the same way that the physical laws, are uniform in time and spaces, i.e. no matter the coordinates of a the cell on the grid the rule is the same,
- is defined by a regular network, analogously to the physical laws are defined uniformly on the space.

Also there is cellular automata with others interesting physical properties, there is cellular automata where its behavior is *reversible*, i.e. is an invertible cellular automata and the inverse is a cellular automata too. The HPP model [6, 7] for the initials of the authors last name: Hardy, Pomeau and de Pazzis is a very good example of a reversible cellular automaton. This is a model for gas in microscopic level, where each particle in the gas is represented by a arrow representing the particle direction and it is in a cell on a two-dimensional grid. Each cell can only contain a maximum of one particle for each direction, i.e., contain a total of between zero and four particles and optionally we add wall-cells to contain the particles. The following rules also govern the model:

- A single particle moves in a fixed direction until it experiences a collision.
- Two particles experiencing a head-on collision are deflected perpendicularly.
- Two particles experience a collision which isn't head-on simply pass through each other and continue in the same direction.

- When a particles collides with the edges of a lattice it can rebound.

Here we can obtain a inverse cellular automata moving the particles in reverse direction.

Other important property of reversible cellular automata is their hardware implementation, this may – theoretically – be more energy efficient than the implementations of irreversible systems used today. Following the Landauer's principle [8] the minimum dissipation energy for erasure of one bit of information at absolute temperature T is at least $kT \ln 2$ Joule of energy, where k is the Bolzmann's constant. Reversible computation has been proposed as an alternative where no bits need to be erased, hence avoiding the $kT \ln 2$ lower bound on the energy dissipation per bit.

Another class of cellular automata is the *conserving cellular automata*, this satisfies that there is a function from the states to real numbers, called *additive quantity*, and for any configuration where the addition of the additive quantity of its state is a number and this number not changes after to apply the conserving cellular automaton.

This class of cellular automata is interest in physics as a source for models of particles systems ruled by conservations laws of mass and/or energy, particularly highway traffic cellular automata (rule 184), fluid flow, eutectic alloys, the exchange of goods between individuals, etc. and includes the HPP model, where the additive quantity of a cell is the number of arrows in this cell.

Discrete dynamics and topology

In the cellular space we can define a *metric* between to configurations c and d as follow: $d(c,d) = 0$ if $c = d$ and $d(c,d) = 2^{-\min\{\|z\|:c_z \neq d_z\}}$, satisfying the three elemental properties of a metric: d is positive, $d(c,d) = 0$ if and only if $c = d$, d is symmetric and d satisfies the triangular inequality.

The value $d(c,d)$ is called *distance* between c and d.

Any pair of a set and a metric with this previous properties is called *metric space*, in particular the cellular space is a metric space. With our definition of distance, we can characterize the cellular automata as the continuous function commuting with the *shift* function in any direction, where the shift function is the function that moves each cell one position in a direction.

The cellular space is compact and is generated by the *cylinders* $\mathrm{Cyl}(c, D)$ a sets of all configurations that are equals to configuration c cell by cell over the cells on $D \subseteq \mathbb{Z}^d$. Each cylinder is topologically closed and open simultaneously.

By compactness any invertible cellular automata is reversible. Also any continuous function over a compact space define a *dynamical system*, then a cellular automata define a dynamical system over the configuration space.

In dynamical system is studied the *limit set*, the set of all configurations a t-th pre-images for any $t \in \mathbb{N}$. In cellular automata the limit set is a *sub-shift* i.e. is closed non-empty and shift invariant.

Complexity

The computational viewpoint is based on Church-Turing's thesis, in which all "reasonable" computational models are equivalent, i.e. everyone can compute the same things. Among these computational models are lambda calculus [9], tag systems [10], counter machines [11] and Turing machines [12], among others, and we say that these are *computationally universal* (also called Turing universal).

Turing universality

From the beginning, the computational universality appears in the cellular automata, as the von Neumann's auto-reproductive cellular automata [13] simulating self-reproduction in nature. Von Neumann build a 29 state, two dimension and von Neumann neighborhood simulating a Turing machine controlling constructor arm that build the copy of the original configuration. Researchers have looked for more and more simple automata, with fewer states, dimensions and neighbors, that show computational universality. Smith (1971) [14] showed that 18 colors and firsts-neighbor 1-dimensional rules could be universal, and Lindgren and Nordahl (1990) [15] constructed a 7-color firsts-neighbor universal cellular automaton, but the most important case is Cook (2004) [16], where, simulating a tag system on the elementary cellular automata 110, shows that two states and firsts-neighbor are enough to show computational universality.

Intrinsic universality

The idea of complexity in cellular automata given by its ability to simulate Turing machines does not seem to be entirely practical. While Turing machines are sequential and finite computer models, cellular automata are an extremely parallel and spatially infinite model. This is how the notion of intrinsic universality of a cellular automata is born, which is the ability of a cellular automaton to simulate any other cellular automata. Some examples of intrinsically universal automata are Albert and Culik's one-dimensional with 14 states and first neighbor neighborhoods [17], Conway's life game [18] and Ollinger and Richard's one-dimensional with 4 states and first neighbor neighborhoods [19]. The last one is, as far as we know, the cellular automata with the least states intrinsically universal in dimension 1 and first neighbors neighborhood. Remain open to know if the rule 110 is intrinsically universal.

The study of intrinsic universality can not only be explored through the reduction of the states of intrinsically universal automata, it is also interesting to see what additional properties we can demand and we continue to find intrinsically universal automata. If we add conserving it is possible to find an intrinsically universal automata that is also conservative [20].. It is not true with reversibility, because reversible automata can only simulate reversible automata. Thus begin the study of intrinsic universality only for reversible cellular automata, where there is an intrinsically universal cellular automata for the reversible ones [21]. Moreover, there is also one that is reversible and conservative [22] and even conservative time-symmetric [1] (time-symmetric is a subclass of reversible cellular automata, for formal definition and properties see [23]).

Although each of the previous authors considers his own notion of simulation, including simulating cell by cell, cell by macro-cell, temporal scales or even space translations, this is formalized in [24, 25], including all these cases in a single definition.

Computational complexity

Another computational aspect studied is the computational complexity, which counts the maximum number of operations performed by a Turing machine to perform a calculation as a function of the size of the input, i.e. the time (or number of operations) used to solve the worst case. This idea of time measurement, along with one for space measurement, was introduced by Hartmanis and Stearns [26]. Edmonds proposes as problems that can be efficiently calculated to those whose complexity can be bounded by a polynomial on the size of the input, called polynomial problems or problems in **P** class. Edmonds also gives an informal description for polynomial time on non-deterministic Turing machines [27], opening one of the most important questions in computational complexity, is $\mathbf{P} = \mathbf{NP}$?, later formalized by Cook [28] and Levin [29].

There is also a class of complexity for parallel computers with a large number of processors, called class **NC**, which corresponds to the kind of problems that can be solved in poly-logarithmic time using a polynomial number of processors. It is named after Nick Pippenger for his studies on circuits with poly-logarithmic depth and polynomial number of logic gates. This is a subclass of **P**, i.e **NC** $\subseteq$ **P** and remains open to know if **NC** = **P**.

For both **P** and **NP** there is a sub-set of problems called **P**-completeand **NP**-complete respectively. In these problems any problem in **P** (**NP**) can be reduced to it in poly-logarithmic time by using a polynomial amount of processors (polynomial time using a processor), where reduction the problem A to B meaning transform the inputs for A in inputs for B through a function computable in **NC** (**P**) time such that a input of A is *accepted* if and only if the transformation of this input is *accepted* by B. So if a **P**-complete (**NP**-complete) problem is also **NC** (**P**), then both classes collapse i.e., **P** = **NC** (**P** = **NP**).

Assuming that **P** $\neq$ **NP**, then the problems **NP**-complete can be solved "efficiently" in a sequential machine . Analogously, it is widely believed that **P** $\neq$ **NC** i.e. there is polynomial problems that cannot exponentially improve their performance by adding a reasonable number of processors (polynomial), it meaning, they cannot be "efficiently" parallelized, then the problems **P**-complete are the candidates to be the problems not parallelized. The complete problems are the most complex in its class, because any other problem in the class can be solved by these through a reduction preserving the complexity of the first problem.

One way to study at cellular automata for this point of view is through the following decision problem: to decide if, given a configuration, a state and a time, the cell in the origin of the grid is in given state at the given time. This problem is known as the Prediction problem, and is polynomial, by simply simulating the automata rule. If we find an "fast" algorithm for Prediction (faster that **P**), then we can get automata iterations much faster than iterating cell by cell, simply 'predicting' the value of each cell. One of the first to study this problem was Moore [30, 31] Showing that the problem of knowing if a cell is going to change is **P**-complete for majority automata and life without death. They can simulate logic circuits in a configuration, so it is possible to solve the circuit value problem by predicting the value of the circuit output cell in the configuration. The circuit value problem is **P**-complete. Thus, even if that **NC** = **P**, there would be no more efficient way to calculate iterations than iterate the automata. It is also important to mention the elementary cellular automaton 110, which is the only elementary one in which it is known until now that Prediction is **P**-Complete [32].

Freezing cellular automata

This is the main family of cellular automata in this work, the *freezing cellular automata*, introduced in [33]. These are cellular automata where the set of states is partially ordered and the cells can only pass from one state to another minor respecting this partial order. This definition induces a limited number of changes per cell, if each cell can change to as much as k times it is known as a cellular automaton with k-changes.

The name "freezing" comes from the property that in freezing cellular automata any configuration ends asymptomatically at a fixed point, i.e. in the long term the configuration is "freezed". From this property we quickly intuit the irreversibility of these cellular automata. In fact, the only reversible (or even surjective) freezing cellular automata is the identity map [33].

This freezing behavior is seen in natural phenomena such as Bootstrap percolation [34], modeling the following physical phenomena:

> *"Consider a pure magnetic system in which the exchange forces on a given spin from its neighbours are barely strong enough to overcome the crystal field. The introduction of non-magnetic impurities would then clearly have a strong effect: if the number of magnetic neighbours is sufficiently reduced, a spin with a magnetic moment can become non-magnetic by being forced into the singlet state."*

This problem can be modeled by a freezing cellular automata with states $\{0, 1\}$ (0: magnetic, 1: non-magnetic) following local rule: $f(c) = 1$ if $c_0 = 1$, $f(c) = 1$ if $c_0 = 0$ and $\sum_{i=0}^{n} c_i > \theta$ and $f(c) = 1$ otherwise, where n is the number of neighbors of the cell to changes and θ is the threshold of non-magnetic cells necessary to make the cell non-magnetic.

Other models are forest fire models [35] or the SIR model [36] of infection propagation (Susceptible, Infected and Recover and acquire immunity), but it is also observed other theoretical models that they are "freezing" such as Abstract Tile Assembly Model (aTAM) [37] or the life without death [30], where the completeness of the PREDICTION problem was shown.

This cellular automata in dimension one are less complex that the non-freezing ones, because PREDICTION is at most NLOGSPACE (NLOGSPACE$\subseteq$ **NC**) while PREDICTION is **P**-complete in general case.

In this thesis we study the freezing cellular automatons and show that, except for the case of the cellular automatons of dimension one, in spite of the apparent limitations due to the fact that states can only advance and therefore can only change a finite number of times, these show a high complexity, both from the point of view of intrinsic universality and computational complexity.

In the case of intrinsic universality, we first show that there is not a intrinsically universal automata for the freezing cellular automata using the definition given in [24, 25], so a slightly more flexible simulation definition is necessary. Given the good definition of simulation, we show that even in this context it is not possible to find an intrinsically universal cellular automata in dimension one, so we move on to study in dimension two or higher. For dimension two we show that it is not possible to find an intrinsically universal cellular automata with von Neumann neighborhood for the freezing cellular automata where cells can change at most once, but this is possible with two or more changes per cell. We show explicitly one that is intrinsically universal for freezer cellular automata with two changes and von Neumann neighborhood. We explore few changes, neighborhood shape and dimensions to find the most simple intrinsically universal freezer cellular automata.

On the other hand, to study computational complexity, we explored the family of cellular freezing automata in the vicinity of von Neumann, totalistic and two-state on periodic configurations. Since freezing cellular automatons always reach a fixed point in a finite time, then we will study a variant of PREDICTION, called STABILITY, in which the question is whether a cell of the freezing cellular automaton will have the same value at the reached fixed point or not. This problem is **P**, because the fixed point is reached in a polynomial time. So it remains to see if there is can be done better. By solving STABILITY cell by cell it is possible to calculate the fixed point reached by the freezing automaton in a polynomial time or faster if we find more efficient algorithms for STABILITY.

We showed that for this family of cellular automata there are **P**-complete problems, where we cannot significantly improve the execution time, except that **P** = **NC**. We also found that most of the cellular automata of this family have a lower complexity, stability is in **NC**, except in trivial cases, where its complexity is even lower.

We finish the study of computational complexity repeating previous study but using sequences update schemes, i.e. the cells change one by one following a pre-established order. To find the maximum complexity of the problem STABILITY in this case (**NP**) we must resort to the third dimension and we show one with this complexity. It is still open the problem of knowing if the automata where STABILITY is **P**-complete in the case of parallel updates is **NP**-complete in the sequential case.

So we show that, except for the case of dimension one, where there is no intrinsically universal freezing cellular automata or computational complexity is at best NLOGSPACE, the rest of the freezing cellular automata has a high complexity. Whether there is an intrinsically universal automata or a freezing cellular automata where the STABILITY problem reaches its maximum complemented with very few states and very small neighborhoods.

The previous results are distributed in the chapters as follows:

Chapter 3

Next chapter deals with the elementary definitions of cellular automata and general elements on topology, dynamic systems graph theory and some extensions of the usual notion of cellular automata. It also includes a section on computational complexity for both parallel and sequential calculations, as well as a brief description of the **NC** algorithms that will be used throughout this work. This chapter closes with the definitions of standard simulation in cells and examples of intrinsically universal cellular automata.

Chapter 4

The following chapter presents the topic of the freezing cellular automata. After explaining the irreversible nature of these cellular automata from the point of view of dynamic systems, it is showed that Stability problem in freezing cellular automata is less complex in dimension one than in higher dimensions, moreover, passing to dimension two the Stability problem reaches the maximum computational complexity. Moreover, in dimension 2 there is already freezing cellular automaton Turing universal. This is shown first by the universal Turing of the counting machines and encoding this in a freezing cellular automata. The chapter ends with the properties of monotones freezing cellular automata and their relationship to asynchronous updating scheme.

Chapter 5

Our first contribution deals with an intrinsically universal freezing cellular automata, in other words, a freezing cellular automata capable of simulating any other freezing cellular automata, and begins with a negative result: there is no such automata with the usual definition of simulation, so it is necessary to relax a little the definition of simulation. Now instead of simulating a cell by a macro-cell (block of cells) that only depends on the simulated cell, we will simulate a cell by a macro-cell that depends on the simulated cell and its neighborhood. Established the good notion of simulation, we built an intrinsically universal freezing cellular automaton with von Neumann neighborhood. Cells can change twice and work by storing the states in cables and sending that information to the neighbors. The simulation is performed by calculating the local function through logical circuits. This result is the best with respect to the number of changes per cell in dimension two. For the above, we first show that freezing cellular automatons with a von Neumann neighborhood and one change cannot cross signals, which is possible with two changes. Trying to build a freezing cellular automata with a one change that can simulate a freezing cellular automata crossing signals leads to a contradiction. This limitation is lost in dimension three or higher, where we can use the third dimension to make crosses. So we find an intrinsically universal freezing cellular automaton with only one change. Moreover, using the freezing cellular automaton that activates a cell with exactly two active neighbors (rule 2 of the chapter 6) we find our intrinsically universal freezing cellular automaton has only two states. The last result of this chapter is a characterization of the cellular automata that can be simulated by a freezing cellular automata. These are those that can be defined as decreasing local energy.

Chapter 6

Our second contribution is the study from the point of view of the computational complexity of the Stability problem in the family of freezing totalistic cellular automata with two states (active and inactive) and considering two different grids: grid with triangular cells (three neighbors per cell) and grid with square cells (four neighbors per cell). In total there are 16 rules for the triangular grid and 32 rules for the square grid, but initially we only consider the rules where the inactive cells are quiescent, leaving in each case half of the rules for study. We denote this rule by a number, where each digit means that the rule activates the cells with exactly that number of active neighbors. In the rules on the triangular grid the most complex thing we found were rules with complexity in **NC**, while in the square grid we also found **P**-complete rules namely 2 and 24.

Chapter 7

Finally we study the same family of rules as in the previous chapter, but considering asynchronous rules, where each cell changes at a different time than the others. With this simple modification in the updating scheme we study the AsyncStability problem, where a cell is stable if for any updating scheme chosen the cell remains inactive. Here we only find **NC** rules, except rules 2 and 24 in the square grid, which are the **P**-complete rules of the previous chapter, where the problem remain open. In dimension 3 is when these rules become **NP**-complete.

Chapter 8

The last chapter concludes that despite the apparent limitations of freezing cellular automata they show a very interesting behavior. If we consider the von Neumann neighborhood in dimension two, with two states we can only find a freezing cellular automata **P**-complete, but if we allow more of these, but only two changes, we get an automata cellular freezing intrinsically universal. It remains open the question if it is possible to do this with only three states (the minimum number of cell to get two change), which is already possible to do in higher dimensions with only two cells.

Chapter 3

Preliminaries

In the present chapter we give the elementary definitions in the framework of the study of cellular automata and the notation that we have chosen for the development of this thesis. There are also definitions and results of related study topics, for example elements of graph theory or computational complexity, which are used throughout this work. All this accompanied by examples that facilitate the understanding of these concepts.

3.1 Basic Notions

Let us first recall the classical definition of cellular automata (CA).

Definition 3.1.1. A *cellular automaton* F of dimension d and state set Q is a tuple $F = (d, Q, N, f)$, where d, its *dimension* is an integer, Q, its set of *states* is a finite set, $N \subsetneq \mathbb{Z}^d$ is its finite *neighborhood*, and $f : Q^N \to Q$ is its *local function*.

It induces a global function, which we also note F, acting on the set of configurations $Q^{\mathbb{Z}^d}$ as follows:

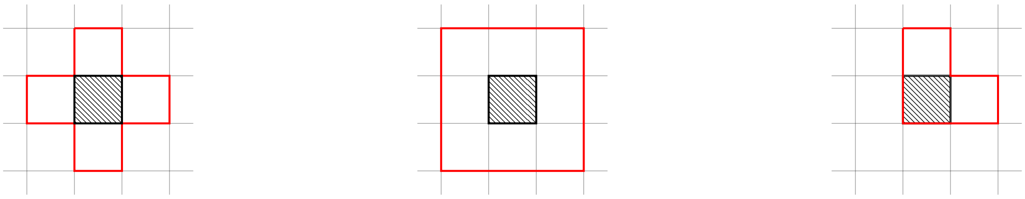
$$\forall c \in Q^{\mathbb{Z}^d}, \forall z \in \mathbb{Z}^d,\ F(c)_z = f(c|_{N(z)})$$

where $N(z)$ denotes the set of all neighbors of z, i.e., $N(z) = \{z + v : v \in N\}$.

Definition 3.1.2. Let $N \subseteq \mathbb{Z}^d$ be a finite set. We call a *pattern* or *finite configuration* to any function $c : N \to Q$. The patterns are simililar to configuration, but they are defined on a finite part of the space.

We define $\mathrm{VN}_d = \{(x_1, ..., x_d) \in \mathbb{Z}^d : |x_1| + ... + |x_d| \leq 1\}$ as the *von Neumann neighborhood* in dimension d. We also use the following neighborhoods in dimension 2: $\mathrm{MN} = \{(x, y) \in \mathbb{Z}^2 : x, y \in \{0, 1, -1\}\}$ is the *Moore Neighborhood*; $\mathrm{LN} = \{(0, 0), (1, 0), (0, 1)\}$ is the *L-neighborhood*.

In Figure 3.1 we show this neighborhoods in dimension 2.



(a) Von Neumann neighborhood.

(b) Moore neighborhood.

(c) L neighborhood.

Figure 3.1: Example of different neighborhoods on $\mathbb{Z}^2$. The cells inside of the red line are the neighborhood and the cells with dashed lines are the center cell of the neighborhood.

A configuration c is *periodic* with period n if

$$\forall z \in \mathbb{Z}^d, \forall i = 1, ..., d : c_z = c_{z+ne_i},$$

meaning that the configuration c is a periodical repetition of the hypercube $[0, ..., n-1]^d$. In dimension 2 this is equivalent to work on the torus. We will define $[k] = \{1, 2, 3, ..., k\}$. Given a finite configuration $x \in Q^{[0,...,n-1]^d}$, $c(x)$ is the periodic configuration with period n such that $x = c(x)|_{[0,...,n-1]^d}$.

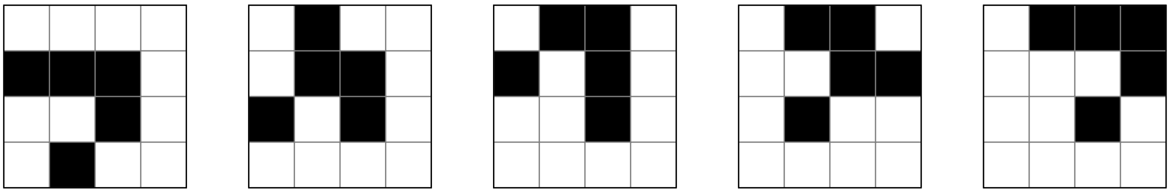
Example 3.1.1 (Game of life)**.** The *Game of Life* (GoL) is probably the most famous cellular automaton. It was created by the British mathematician John Conway in 1970 [5] and it is a CA on the bi-dimensional grid with states $\{0, 1\}$ or $\{\text{dead}, \text{alive}\}$ and Moore neighborhood, where a dead cell becomes alive if it has exactly 3 living neighbors and cell remains alive if it has 2 or 3 living neighbors alive. Formally $GoL = (2, \{0, 1\}, MN, f)$, where

$$f(c) = \begin{cases} 1 & \text{if } (c_0 = 0 \wedge \sum_{w \in M} c_w = 3) \vee (c_0 = 1 \wedge \sum_{w \in M} c_w \in \{2, 3\}) \\ 0 & \text{otherwise.} \end{cases}$$

Some configuration of finite alive cells that can be observed in the GoL are:

the glider , 5 alive cells configuration that reappears shifted one cell at direction north-est after 4 iterations, see Figure 3.2, and

the pulsar , 48 alive cells configuration that reappears in the same position after 3 iterations, see Figure 3.3.



(a) Initial configuration.

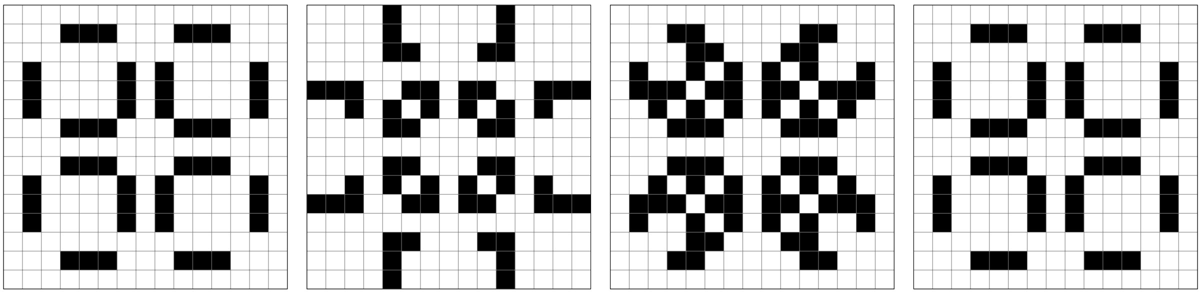
(b) Step 1.

(c) Step 2.

(d) Step 3.

(e) Step 4.

Figure 3.2: A glider. After 4 iterations, the initial pattern appears again but shifted one cell in the northeast direction.



(a) Initial configuration. (b) Step 1. (c) Step 2. (d) Step 3.

Figure 3.3: A pulsar. After 3 iterations, the initial pattern appears again in its initial position.

A simple and very studied family of CA is the family of *elementary cellular automata*, the one dimensional CA with two states and von Neumann neighborhood.

Definition 3.1.3. The *elementary cellular automata*, $E_i = (1, \{0,1\}, (-1,0,1), f_i)$ are CA with 2 states and the von Neumann neighborhood (three cells). We have $2^3 = 8$ possible configurations with 3 cells, then there are 8 possible inputs for a local function with this neighborhood. For each possible neighborhood $(n_2, n_1, n_0) \in \{0,1\}^3$, we can associate a number $k = n_2 2^2 + n_1 2^1 + n_0 2^0 \in \{0, ..., 7\}$. Then we can represent each local function $f_i : \{0,1\}^3 \to \{0,1\}$ a table, where in each row appear N_K and $f_i(N_k)$.

Also we can denote this rules by a number i given by $\sum_{k=0}^{7} f(N_k)2^k$.

Example 3.1.2. The elementary cellular automata 110 or simply rule 110 ($110 = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$), is given by the table 3.1.

Neighborhood(N)	$f_{30}(N)$
000	0
001	1
010	1
011	1
100	0
101	1
110	1
111	0

Table 3.1: Table of local function of elementary CA 110.

A very simple way to see the dynamic on a 1-dimensional CA is in its *space-time diagram*. Given a configuration c, its space-time diagram is a configuration in $Q^{\mathbb{Z}\times\mathbb{N}}$where in the cell at coordinate (z,t) is the state $F^t(c)_z$. We define $\mathrm{Diagr}(F)$ as the set of all space-time diagram of the CA F.

A space-time diagram for a random initial configuration for the rule 110 is given in Figure 3.4.

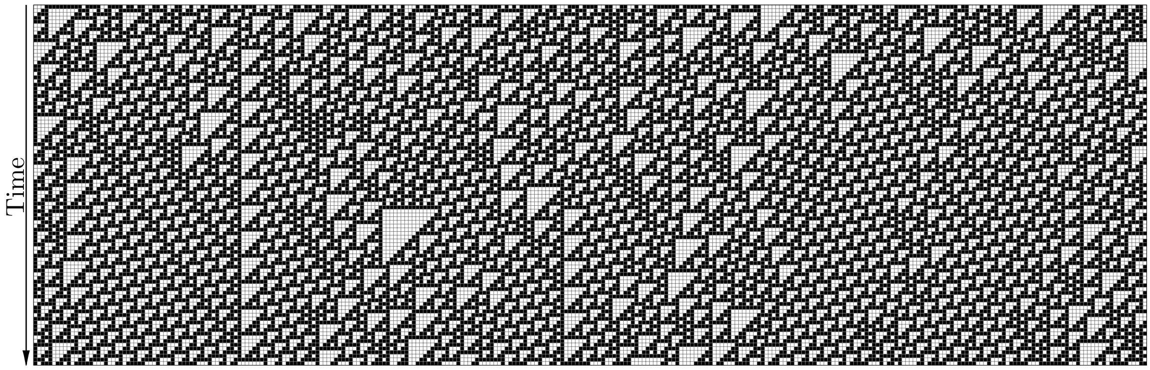


Figure 3.4: Example of space-time diagram for the rule 110. In this image the time is top to bottom.

We generalize the space-time diagram for a d-dimensional configuration c as a $(d+1)$-dimensional configuration where in its coordinate (z,t) is in the state $F^t(c)_z$, $z \in \mathbb{Z}^d$.

In several cellular automata from the literature [38, 36, 34, 39, 30], there is a global bound on the number of times a cell can change: they are *bounded-change* cellular automata. This property is found in the bootstrap percolation [34], as well as in other well-known examples such as "Life without death" [30] and various models of propagation phenomena like in [36].

Moreover, in all those examples, the bound is defined through an explicit order on states. Such an automaton is a *freezing cellular automaton.* This *freezing-order* on state can also be used to define interesting subclasses (see Section 5.3).

The main subject in this thesis is to study the family of freezing cellular automata.

Definition 3.1.4 (Freezing Cellular Automaton [33])**.** A CA F is a $\prec$*-freezing* CA, for some (partial) order $\prec$ on states, if $F(c)_z \prec c_z$ for any configuration c and any cell z. A CA is *freezing* if it is $\prec$-freezing for some order.

Definition 3.1.5. We say that a cellular automaton is k*-change* if any cell in any orbit changes at most k times its state.

Example 3.1.3 ([33])**.** The one dimensional cellular automaton given by the following local rule f : $\{0,1,2\}^2 \to \{0,1,2\}$is a 2-change cellular automata non-freezing:

$$f(a,b) = \begin{cases} 1 & \text{if } b = 0 \\ 2 & \text{otherwise.} \end{cases}$$

because both $2 \to 1$ and $1 \to 2$ are possible state changes in one cell, but after 2 steps all cells are in state 2.

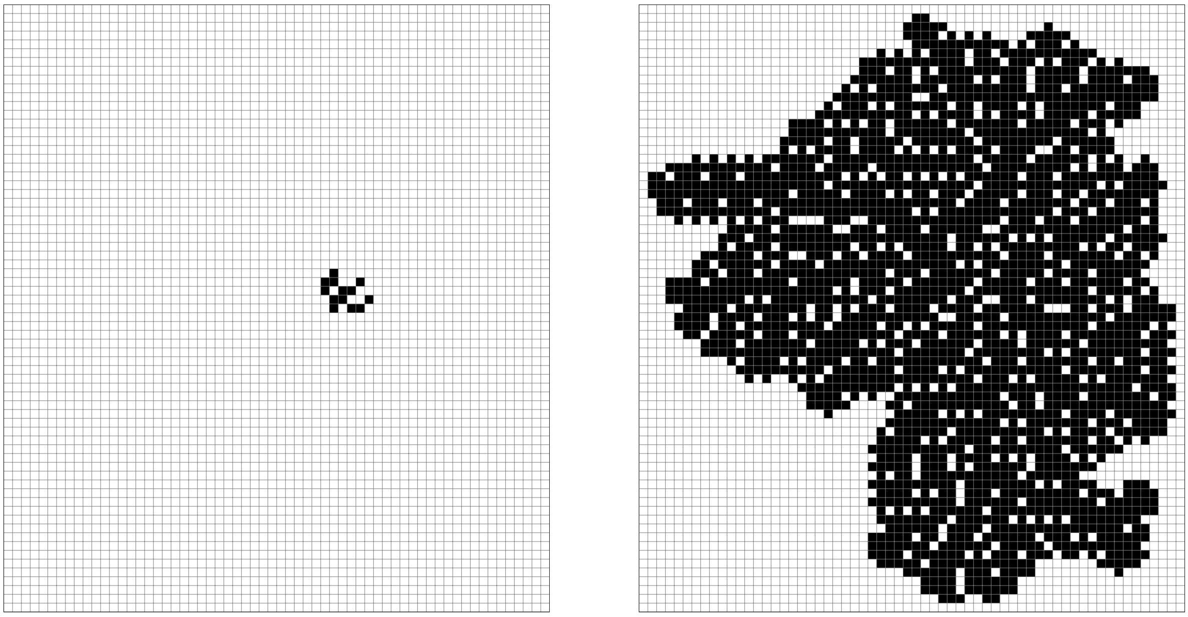
Every freezing CA is k-change for some k (at most the depth of its freezing order, but possibly less, as in [35], where a cell can be unburnable $[U]$, not currently ignited$[N]$, ignited$[I]$ or consumed $[C]$, but the only possible transitions are $N \to I \to C$ and U). For $V \subsetneq \mathbb{Z}^d$, we note FCA_V for the class of d-dimensional freezing cellular automata with neighborhood V. Finally, we set $\text{FCA}^d = \bigcup_{V \subsetneq \mathbb{Z}^d} \text{FCA}_V$ and omit d when context makes it clear.

Example 3.1.4 (Life without death)**.** The *life without death* (LWoD) is the freezing version of the game of life, where alive cells remains in this state forever. Formally life without death is defined as $LWoD = (2, \{0,1\}, MN, f)$, where

$$f(c) = \begin{cases} 1 & \text{if } c_1 = 1 \vee \sum_{w \in M} c_w = 3) \\ 0 & \text{otherwise.} \end{cases}$$

This meaning that a cell is born when it has exactly 3 alive neighbors cells. Some patterns that can be observed in the Game of Life are:

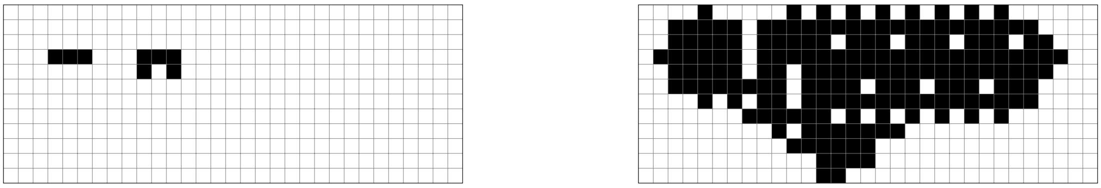
- the Chaotic lava, cells growing indefinitely connected in at less 2 directions like a stain, see Figure 3.5, and
- the ladders, cells growing indefinitely in one direction, see Figure 3.6.



(a) Initial configuration.

(b) Step 100.

Figure 3.5: A lava. Alive cells grow in all directions.



(a) Initial configuration.

(b) Step 51.

Figure 3.6: A ladder. After 51 iterations, an 8-cell-wide signal begins to grow to the right.

Another family of CA that we will study is the *monotone cellular automata*. Monotony in cellular automata is analogous to the monotony in functions of real numbers, when we apply a monotone function over two points, the first one is less or equal that the second one, then the image of the first one is less or equal that the image of the second one. These CA include the majority vote CA [31] and the bootstrap percolation model [34].

Definition 3.1.6 (Monotone Cellular Automaton)**.** A CA F is a $\prec$*-monotone* CA, for some (partial) order $\prec$ on configurations, if $c \prec c' \Rightarrow F(c) \prec F(c')$ for any configurations c and c'. A CA is *monotone* if it is $\prec$-monotone for some order.

Example 3.1.5 (Majority vote CA)**.** The *majority vote CA* is a CA with states 0 and 1 and a cell changes to 1 if the majority of its neighbors is in state 1, i.e. if the number of neighbors is greater (or equal) that the half of the size of the neighborhood. If in the previous case we do not admit the equality to change to 1 it is the *strict majority CA*, but if we admit the equality it is the *non-strict majority CA*. The difference between them is that in the strict majority there are no changes in the ties.

Formally the (non-)strict majority CA is defined as $(d, \{0,1\}, N, f)$, where

$$f(c) = \begin{cases} 1 & \text{if } \sum_{w \in N} c_w > \frac{|N|}{2} \left(\geq \frac{|N|}{2} \right) \\ 0 & \text{otherwise.} \end{cases}$$

The order ($\prec$) defined in the set of states for freezing cellular automata induces a partial order for the configurations, where a configuration $c \prec c'$ if and only if for all z , $c_z \prec c'_z$. Thus, we say that a CA is a *monotone freezing* CA if it is freezing and monotone for the order in the configurations induced by the order in the states. For example, the bootstrap percolation model is a freezing monotone CA, this is the freezing version of the majority CA.

Definition 3.1.7 (Totalistic Cellular Automaton [40]). A CA F is a *totalistic* CA, if its states are numbers and its local function depend only of the value of the center cell and the sum of its neighbors values, i.e.

$$f(c_N) = f\left(c_\theta, \sum_{z \in N} c_n \right).$$

Note that in a totalistic CA the position of the neighbors does not matter, only the total sum. Thus, in the case of symmetric neighborhoods, we obtain invariant rules for rotations and reflections. We can see this symmetry in physical processes, such as Bootstrap percolation [34], where a cell changes its state to another if the number of neighbors in the other state passes a certain threshold.

Example 3.1.6. The game of life, life without death, and Majority vote CA are totalistic CA.

3.1.1 Some terminology of topology

We will give the basic notions of topology on the configuration space. Along with these we include classic results on the topological aspects of cellular automata.

It is possible to define a metric on the set of configurations and with this look at the CA as functions in a metric space. In particular, we will show that CAs are continuous functions [41] and with this we can use the analysis tools to obtain interesting properties.

Definition 3.1.8. Let Q be a finite set. We define the *distance* function d between two configurations $x, y \in Q^{\mathbb{Z}^d}$ as

$$\begin{aligned} d : Q^{\mathbb{Z}^d} \times Q^{\mathbb{Z}^d} &\longrightarrow \mathbb{R} \\ (x,y) &\longmapsto d(x,y) = \left\{ \begin{array}{ll} 0 & \text{if } x = y \\ 2^{-\min\{\|i\|:\ x_i \neq y_i\}} & \text{if } x \neq y \end{array} \right. \end{aligned}$$

Proposition 3.1.1. $\left(Q^{\mathbb{Z}^d}, d\right)$ *is a metric space.*

Proof. We must prove the following properties:

a) $\forall\ x, y \in Q^{\mathbb{Z}^d}, (\ d(x,y) \geq 0) \wedge (d(x,y) = 0\ \Leftrightarrow x = y)$.

b) $\forall\ x, y \in Q^{\mathbb{Z}^d}, d(x,y) = d(y,x)$.

c) $\forall\ x, y, z \in Q^{\mathbb{Z}^d}, d(x,z) \leq d(x,y) + d(y,z)$.

Indeed,

a) It is true by definition of d.

b) Let $x, y \in Q^{\mathbb{Z}^d}$.

  - If $d(x,y) = 0$, then $x = y$, given $x = y \iff y = x$, then $d(y,x) = 0$.
  - If $d(x,y) \neq 0$, then $\exists n \in \mathbb{N} : n = \min\{|i| : \; x_i \neq y_i\} = \min\{|i| : \; y_i \neq x_i\}$, thus $d(x,y) = d(y,x)$.

c) Let $x, y, z \in Q^{\mathbb{Z}^d}$ and $d(x,y) = 2^{-i}$, $d(y,z) = 2^{-j}$, then:

$$\begin{aligned} |k| < i &\Longrightarrow x_k = y_k \\ |k| < j &\Longrightarrow y_k = z_k \end{aligned}$$

Without loss of generality let us suppose that $i \leq j$, then $|k| \leq i \Longrightarrow |k| \leq j$ then $|k| \leq i \Longrightarrow x_k = y_k = z_k$. Finally,

$$\begin{aligned} \iff & \quad \min\{|k| : x_k \neq z_k\} \geq i \\ \iff & \quad -\min\{|k| : x_k \neq z_k\} \leq -i \\ \iff & \quad 2^{-\min\{|k| : x_k = z_k\}} \leq 2^{-i} \\ \iff & \quad d(x,z) \leq d(x,y) \\ \Longrightarrow & \quad d(x,z) \leq d(x,y) + d(y,z) \end{aligned}$$

□

Note that, from the proof of property c) we see that this is a little more general; in fact the property we have is

$$\forall \; x, y, z \in Q^{\mathbb{Z}^d}, d(x,z) \leq \min\{d(x,y), d(y,z)\}$$

The metrics with this property are known as *ultrametrics.*

Definition 3.1.9. The *ball* centered in x and with radius r is the set $B(x,r) = \{y \in Q^{\mathbb{Z}^d} : d(x,y) < r\}$.

The ball centered in x and with radius r is the set of all configurationS equal to x in a hyper-cube centered in θ and with width $2r + 1$.

Definition 3.1.10. Given a set of configurations $U \subseteq Q^{\mathbb{Z}^d}$, we say that $x \in U$ is an *interior point* if $\exists r > 0 : B(x,r) \subsetneq U$.

If every point in U is an interior point we say that U is a *open set.* If the complement of U is an open set we say that U is *closed.*

Proposition 3.1.2. *The open sets of* $Q^{\mathbb{Z}^d}$ *satisfy the following properties:*

- $\emptyset$ *and* $Q^{\mathbb{Z}^d}$ *are open.*
- *the arbitrary union of open sets is an open set.*
- *the intersection of finitely many opens is an open set.*

In general, given a set X and $\mathcal{T} \subseteq \mathcal{P}(X)$, where $\mathcal{P}(X)$ is the power set of X, we say that $(X, \mathcal{T})$ is a *topological space* if it satisfies the previous properties. The elements of $\mathcal{T}$ are called *open set.*

Definition 3.1.11. A *sequence* is any $(x^1, x^2, ..., x^n, ...)$, where $\forall i \in \mathbb{N} : x^i \in Q^{\mathbb{Z}^d}$. We denote a sequence by $(x^n)_{n\in\mathbb{N}}$.

A *subsequence* of $(x^n)_{n\in\mathbb{N}}$ is a sequence $(x^{k_1}, x^{k_2}, ..., x^{k_n}, ...)$, where $k_1, ..., k_n, ...$ is the set of index satisfying $k_i < k_{i+1}$. We denote a subsequence of $(x^n)_{n\in\mathbb{N}}$ indexed by k as $(x^{k_n})_{n\in\mathbb{N}}$.

Definition 3.1.12. Given a sequence by $(x^n)_{n\in\mathbb{N}}$, we say that $(x^n)_{n\in\mathbb{N}}$ *converges* to x, denoted $x^n \to x$ or $\lim_{n\to\infty} x^n = x$ if

$$\forall \varepsilon > 0, \exists n_0 \in \mathbb{N}, \forall n > n_0 : \, d(x^n, x) < \varepsilon.$$

Definition 3.1.13. A *continuous function* is any function $F : Q^{\mathbb{Z}^d} \to Q^{\mathbb{Z}^d}$ such that for any sequence $x^n \to x$ satisfies

$$\lim_{n\to\infty} F(x^n) = F(x).$$

The following theorem allows us to relate the CA to the continuity in $Q^{\mathbb{Z}^d}$.

Theorem 3.1.3 (Curtis–Hedlund–Lyndon [41]). *(d, f, Q, N) is a cellular automaton if and only if F is continuous in $\left(Q^{\mathbb{Z}}, d\right)$ and $\forall i = 1, ..., d : F \circ \sigma_i = \sigma_i \circ F$, where $\sigma_i(x)_z = x_{z-e_i}$ is a shift function and e_i is the vector with 1 at possition i and 0 otherwise.*

Definition 3.1.14 (Compactness). A $K \subseteq Q^{\mathbb{Z}^d}$ is *compact* if and only if every sequence $(x^n)_{n\in\mathbb{N}}$ on K has a subsequence that converges to an element of K.

Theorem 3.1.4. *$(Q^{\mathbb{Z}^l}, d)$ is a compact metric space, i.e. is a compact set under the metric d.*

Proof. Let $(x^n)_{n\in\mathbb{N}}$ be a sequence in $Q^{\mathbb{Z}^d}$ and $r = 1$. Now, we have infinites patterns $x^n_{[-1,1]^d}$ of finite values, so by pigeonhole principle at least there is one pattern that is repeated infinitely. Let k_1 the first index such that $x^{k_1}_{[-1,1]^d}$ is this pattern and let $x_{[-1,1]^d}$ be the pattern. Note that $d(x^{k_1}_{[-1,1]^d}, x) \leq 2^{-(r+1)}$ for any x configuration containig the pattern $x_{[-1,1]^d}$.

Inductively, we build a subsequence $(x^{k_r})_{r\in\mathbb{N}}$ and a sequence of patterns $x_{[-r,r]^d} \in Q^{[-r,r]^d}$ where for all r:

$$x^{k_n}_{[-r,r]^d} = x_{[-r,r]^d},$$

then $d(x^{k_r}_{[-r,r]^d}, x) \leq 2^{-(r+1)}$ for any x configuration containig the pattern $x_{[-r,r]^d}$. Let x be a configuration containg all the patterns $x_{[-r,r]^d}$, then $d(x^{k_r}_{[-r,r]^d}, x) \leq 2^{-(r+1)}$. Finally, $d(x^{k_r}_{[-r,r]^d}, x) \to 0$ then $x^{k_r} \to x$.

□

In topology the compact sets have very nice properties, in particular we will use the following, called *Cantor's intersection theorem.*

Theorem 3.1.5 (Cantor's intersection theorem). *Any decreasing nested sequence of non-empty compact subsets of $Q^{\mathbb{Z}^d}$ has a non-empty intersection, i.e., supposing $(C_n)_{n\in\mathbb{N}}$ is a sequence of non-empty, compact subsets of $Q^{\mathbb{Z}^d}$ satisfying:*

$$C_0 \supseteq C_1 \supseteq \cdots C_k \supseteq C_{k+1} \cdots$$

it follows that

$$\bigcap_k C_k \neq \emptyset.$$

3.1.2 Some terminology of dynamical systems

The dynamical system is a mathematical formalization for any fixed "rule" which describes the time dependence of a point's position in its ambient space. In our case we will consider the configurations as point being changed over time by the cellular automaton. We are interested to study the behavior that remains in the time, as the points that never change, called *fixed points* and the points that are reached when indeterminately iterate the cellular automaton, called *limit configuration.*

Definition 3.1.15. Given a cellular automaton F, an *orbit* of a point c is the sequence $(F^t(c))_{t\in\mathbb{N}}$

When we see a space-time diagram of a cellular automaton we see the orbit of the configuration from below, where the base of the diagram is the initial configuration, upward, where we see the time advance.

Definition 3.1.16. A *fixed point* of F is a configuration c such that $F(c) = c$, i.e a configuration such that its orbit is a singleton.

We see the fixed point in the space-time diagram of a cellular automata as a row repeated infinitely.

Example 3.1.7. The homogeneous configuration with only 0 is a fixed point for the game of life, life without death, and Majority vote CA. A configuration with cells in state 1 surrounded by cells in state 0 is a fixed point for an automata that simulates Bootstrap percolation.

Definition 3.1.17. A *limit configuration* of c for F, denoted $F^\infty(c)$ is the limit of the orbits of c, i.e. $\lim_{t\to\infty} F^t(c)$ when this limit exists.

Example 3.1.8. For the Game of life, the limit configuration of a configuration with a single glider is the homogeneous configuration with only 0, but a configuration with a pulsar does not have a limit configuration, because if c is the configuration with a single pulsar, then $(F^{3t})_{t\in\mathbb{N}}$ converges to Figure 3.3a and $(F^{3t+1})_{t\in\mathbb{N}}$ converges to Figure 3.3b.

Proposition 3.1.6. *Given a cellular automata F and a configuration c, if there is the limit configuration $F^\infty(c)$, then it is a fixed point.*

With the orbit and limit configuration we study the long-term behavior of cellular automata. In this context the *limit set* is the set that contains the configuration with an "infinite" behavior.

Definition 3.1.18. Given a cellular automaton F, the *limit set* of F, denoted Ω_F is the set of configuration c such that for any t, c has a t-th pre-image, i.e. $F^{-t}(c)$ is not empty. Formally

$$\Omega_F = \bigcap_{t=0}^{\infty} F^t(Q^{\mathbb{Z}^d}).$$

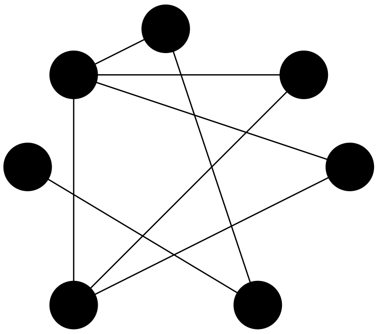
Example 3.1.9. The fixed point always are in the limit set.

Example 3.1.10. In reversible cellular automata the limit set is $Q^{\mathbb{Z}^d}$.

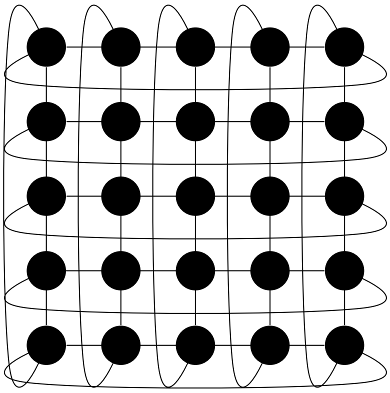
3.1.3 Some graph terminology

In the study of cellular automata properties, it is very useful to study the relationship between neighboring cells and the structure they form. These structures can be seen from the point of view of graph theory. A *graph* is simply a finite set of points (called *vertices*) some of them connected in pairs by lines (called *edges*). For example, a graph can be a group of people and two people are connected if they are friends. Another more related example would be cells in a grid and two cells are connected when being neighbors.

We represent the vertices of a graph as disks and an edge between two vertices as a line connecting them as Figure 3.7.



(a) Example of a graph.



(b) Example of a graph from a torus grid with von Neumann neighborhood.

Figure 3.7: Example of graphs.

A formal definition is the following:

Definition 3.1.19. A graph is a pair $G = (V, E)$ comprising a finite set V of vertices together with a set E of edges, which are 2-element subsets of V.

Definition 3.1.20. Given a graph $G = (V, E)$ the *neighborhood* of $v \in V$ is the set $N(v) = \{u \in V : \{u, v\} \in E\} \cup \{v\}$, i.e. it is the set of all vertices connected with v by a single edge also by itself. The *degree* of v is the number of adjacent cells of v or the neighborhood of v, denoted by $\delta(v) = |N(v)| - 1$ and we *maximum degree* of G is $\Delta(G) = \max_{v \in V} \delta(v)$.

When the notation is confusing, we will add a sub-index indicating the graph on which we are calculating, e.g $N_G(v)$ is the set of neighbors of v in G.

Note that we do not consider that v is connected with v, but v is a neighbor to himself.

For a set of cells $S \subseteq V$, the *subgraph induced by* S denoted by $G[S] = (S, E')$ is the graph defined with vertex set S, where two vertices of S are adjacent if the corresponding sites are neighbors for the neighborhood, i.e. $E = \{u, v \in S : \{u, v\} \in E\}$. If $c \in Q^V$ and $q \in Q$, we define $G[q] = G[\{v \in V : c_v = q\}]$, meaning, $G[q]$ is the subgraph induced by the vertices that are in state q in c.

For a graph $G = (V, E)$, a sequence of vertices $P = (v_1, \ldots, v_k)$ is called a $(v_1 - v_k)$- *path* if $\{v_i, v_{i+1}\}$ is an edge of G, for each $i \in [k-1]$. Two $(u - v)$-paths P_1, P_2 are called *disjoint* if these only share the vertices u and v. A $(u - u)$-path does not repeat vertices except u as first and last vertex is called a *cycle.*

Definition 3.1.21. A graph G is called *k-connected* if for every pair of distinct vertices $u, v \in V(G)$, G contains at least k two-by-two disjoint $(u - v)$-paths.

A 1-connected graph is simply called *connected*, a 2-connected graph is called *bi-connected* and a 3-connected graph is called *tri-connected.* A maximal set of vertices of a graph G that induces a k-connected subgraph is called a *k-connected component* of G.

3.1.4 Some generalizations of cellular automata

Asynchronous cellular automata

When we study cellular automata, this is defined over a regular grid divided in cells, each cell having a state which evolves according to the states of their neighbors in the grid in synchronous time-steps. For some applications on real models as biological or social models, the hypothesis that each cell has a clock capable of synchronizing the interactions between the cells is somehow unrealistic. A dynamical system that considers this difficulty is the *asynchronous cellular automata*, where cells evolve one-by-one, following an order called *updating scheme*.

Definition 3.1.22. An *Asynchronous Cellular Automaton* (ACA) $F = (d, Q, N, f)$, with states Q, neighborhood $N \subseteq \mathbb{Z}^d$ and *local function* $f : Q^N \to Q$, defines a global function $F : Q^{\mathbb{Z}^d} \to Q^{\mathbb{Z}^d}$, where the new states of the configuration c are defined by the *asynchronous* application cell by cell of the local function on $c(x)$ following the order given by $\sigma : \mathbb{N} \to \mathbb{Z}^d$. Formally the t-th asynchronous iteration of c is given by,

$$F^{\sigma(0)}(c) = c$$

$$F^{\sigma(t)}(c)_z = \begin{cases} f(F^{\sigma(t-1)}(c)_{N(z)}) & \text{if } z = \sigma(t) \\ F^{\sigma(t-1)}(c)_z & \text{otherwise,} \end{cases}$$

where $\sigma : \mathbb{N} \to \mathbb{Z}^d$, called *updating scheme*, is a function.

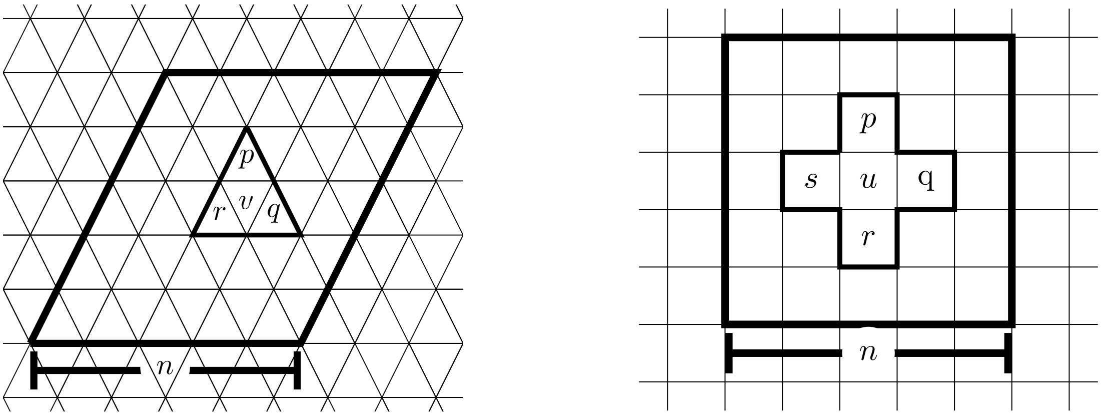
This definition means that in each time t we update only the cell $\sigma(t)$ and the other cells remains in their previous states.

If c is a periodic configuration with period T, we define $\sigma : \mathbb{N} \to [T]^d$, where for all $a, b \in [T]$, $a + b$ is computed modulo T. Thus, in dimension two this is equivalent to work on the torus.

An asynchronous cellular automaton is called *freezing*, *monotone* and/or *totalistic* if the CA with the same local function is monotone, totalistic and/or freezing.

Triangular grid

Another important generalization of CA is the shape of the cells on the grid. In the plane we are using a grid obtained by the regular tessellation of the plane by square cells, as in Figure 3.8b, but it is possible to define CA over other regular tessellations. In this work we will also consider plane tessellations by equilateral triangles, as in Figure 3.8a, which we will call a *triangular grid*.



(a) Triangular grid, with its neighbors p, q and r.

(b) Square grid. Neighbors p, q, r and s.

Figure 3.8: Triangular (a) and square (b) grids with the von Neumann neighborhood of a cell u.

Definition 3.1.23. In the triangular grid the *von Neumann neighborhood* of a cell u is the set of its three adjacent cells and itself.

Definition 3.1.24. We define $T(n)$ as the set of cells in the diamond of n cells per edge, as in Figure 3.8a. Let $x \in Q^{T(n)}$ by a finite pattern as in Figure 3.8a, $c(x)$ is the periodic configuration obtained by the repetition of x in the plane. So, this is equivalent to defining a configuration on a torus.

Automata Network

Here we will introduce a more general dynamical system called *Automata Network*, where we replace the grid by a graph, the cells by the vertices of the graph and the neighborhood by the set of adjacent cells. Also, we allow different local functions for each cell.

Definition 3.1.25. An *Automaton Network A* is a triple (Q, G, F), whose components are as follows:

- Finite set of states Q.
- $G = (V, E)$ is a finite undirected graph without self loops. The vertices of G are numbered using the integers 1, 2, ..., $n = |V|$.
- For each vertex i of G, F specifies a local transition function, denoted by f_i . This function maps $Q^{N(i)}$ into Q.

Also we consider updating schemes σ,

- σ is an ordered partition of V with n parts specifying the order in which nodes update their states using their transition functions, i.e. at time t all the vertices $\sigma(t \text{mod} n)$ are updated simultaneously.

When $\forall t : \sigma(t) = V$ we say that σ is a *parallel* update scheme, if $\forall t : |\sigma(t)| = 1$ it is called an *asynchronous* update scheme and σ is called a *block sequential* update scheme or *serial-parallel* update scheme otherwise [42].

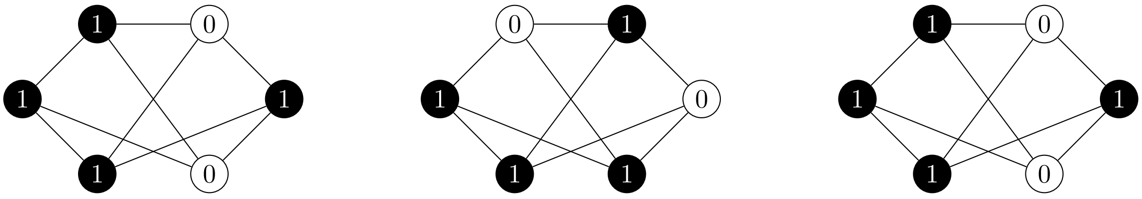
A *configuration* c of Q can be interchangeably regarded as a $|V|$-vector $(c_1, c_2, ..., c_n)$, where each $c_i \in Q$, or as a function $c : V \to Q$. From the first perspective, c_i is the state value of node i in configuration c, and from the second perspective, $c(i)$ is the state value of node i in configuration c.

We use F^σ to denote the global transition function associated with Q following the updating scheme σ. This function can be viewed as a function that maps D^V into D^V. F^σ represents the transitions between configurations and can therefore be considered as defining the dynamic behavior of the Automaton Network. Formally,

$$F^{\sigma(0)}(c) = c$$

$$F^{\sigma(t)}(c)_v = \begin{cases} f_v(F^{\sigma(t-1)}(c)_{N(v)}) & \text{if } v \in \sigma(t) \\ F^{\sigma(t-1)}(c)_z & \text{otherwise.} \end{cases}$$

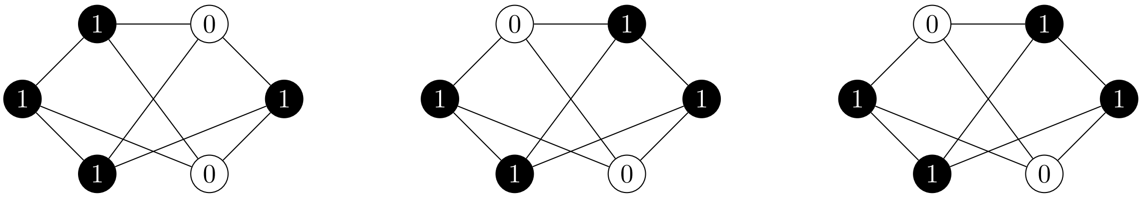
Example 3.1.11. Given the initial configuration of Figure 3.9a, and a parallel update scheme we obtain the following dynamics for the automata network where $f_v(c) = 1$ if $\sum_{w \in N(v)} c_w \geq 3$ and $f_v(c) = 0$ otherwise.



(a) Initial state. (b) Step 1. (c) Step 2.

Figure 3.9: Dynamics on an automata network considering the parallel update scheme.

Example 3.1.12. Given the following configuration of Figure 3.10a, and a block sequential update scheme we obtain the following dynamics for the automata network where $f_v(c) = 1$ if $\sum_{w \in N(v)} c_w \geq 3$ and $f_v(c) = 0$ otherwise.



(a) Initial state. (b) Step 1. (c) Step 2.

Figure 3.10: Dynamics on an automata network considering a block sequential update scheme, in this cases $\sigma(1) = \{1, 2, 3\}$ and $\sigma(2) = \{4, 5, 6\}$.

With this definition Automata Network are more general that the (asynchronous) cellular automata, choosing (Q, G, F) properly.

Example 3.1.13. Let $F = (2, Q, VN, f)$ be a cellular automaton over periodical configuration of size n^2, we can define an Automaton Network (Q, G, F') and choose an updating scheme σ, as follow,

- $G(V, E)$ is a graph as Figure 3.7b and $V = [n]^2$,
- F' is such that $\forall v \in v : f_v = f$ and
- σ is such that $\forall t \in \mathbb{N} : \sigma(t) = V$.

Thus, $\forall c \in Q^V : F(c) = F'^{\sigma(t)}(c)$.

We can extend this example to dimension d and an arbitrary neighborhood choosing (Q, G, F) properly. Moreover, we can extend this example to non-periodical configurations if we admit that $G = (V, E)$ has infinite vertex.

If in the previous example we change the graph by an analogous of the Figure 3.7b for a triangular grid, the we can obtain also a cellular automaton over a triangular grid.

Example 3.1.14. Let $F = (d, Q, VN, f)$ be an asynchronous cellular automaton over periodical configurations of size n^2 and updating scheme σ, we can define Automaton Network (Q, G, F') and choose an updating scheme σ', as follows,

- $G(V, E)$ is a graph as Figure 3.7b and $V = [n]^2$,
- F' is such that $\forall v \in v : f_v = f$ and
- σ' is such that $\forall t \in \mathbb{N} : \sigma(t) = \sigma'(t)$.

Thus, $\forall c \in Q^V : F^{\sigma(t)}(c) = F'^{\sigma'(t)}(c)$.

3.2 Computational Complexity

In computational complexity we study the time (number of operations) it takes for the algorithms (either on Turing machines or another model, see [43, 44]) to respond, as a function of the size of the input. In particular, we are interested in algorithms that solve *decision problems*, where the algorithm is capable to *decide* if the input satisfies a property.

Example 3.2.1. The decision problem PREDICTION [31] has input a site u, a time T, a state q and a finite configuration x such that it is possible to compute $F^T(c(x))_u$. It responds "*Accept*" if $F^T(c(x))_u = q$ and "*Reject*" if $F^T(c(x))_u \neq q$.

PREDICTION
Input: A time $T > 0$, a state q and a site u and a finite configuration x containing the T-th neighborhood of u.
Question: Does $F^T(c) = q$ when $c = c(x)$?

where the 1-st neighborhood of u is $N(u)$ and the T-th neighborhood of u is the union of the neighbors of the $(T-1)$-th neighborhood of u.

3.2.1 The big-O notation

In order to compare the efficiency of two algorithms we count the number of operations performed by a Turing machine to compute the calculation as a function of the input size. But the difference in computational cost between an algorithm that uses n^2 operations and another that uses $2n^2$ operations is little compared to the difference between an algorithm that uses n^2 operations and another that uses n^4 operations when $n \to \infty$.

With this idea we introduce the *asymptotic analysis*, in which we only consider the higher order coefficients in the polynomials. In *asymptotic notation* an algorithm that uses $2n^2 + 3$ operations is denoted $2n^2 + 3 = \mathcal{O}(n^2)$, hence the name *The big-O notation.*

We formalize this notion in the following definition.

Definition 3.2.1 ([43])**.** Let f and g be functions $f, g : \mathbb{N} \to \mathbb{R}^+$. Say that $f(n) = \mathcal{O}(g(n))$ if there are positive integers c and n_0 such that for every integer $n \geq n_0$,

$$f(n) \leq cg(n) \ .$$

When $f(n) = \mathcal{O}(g(n))$, we say that $g(n)$ is an *upper bound* for $f(n)$, or more precisely, that $g(n)$ is an *asymptotic upper bound* for $f(n)$, to emphasize that we are suppressing constant factors.

This means that a function f is asymptotically greater than a function g if f grows much faster than g, regardless of whether g initially has higher values that f. When an algorithm is asymptotically greater than another we say that it is more complex that the other.

A function of interest for its applications mainly in parallel algorithms is the logarithm function. First, using the definition of the big-O notation, note that all logarithms have the same complexity, using the identity $\log_b(x) = \frac{\log_k(x)}{\log_k(b)}$. Another properties of logarithm is that $\log(n^k) = k\log(n)$, then we will not explicitly say the basis of the logarithm used in the expression $\mathcal{O}(\log(n))$ to say $\mathcal{O}(\log_b(n^k))$.

A function $f(n) = \mathcal{O}(n^k)$ is called a *polynomial bound* and a function $f(n) = \mathcal{O}(\log^k(n))$ is a called *logarithmic bound* and we will use this several times to measure the complexity of algorithms.

3.2.2 Parallel Computation

The class **P** is the class of problems that can be solved by a deterministic Turing machine in time $\mathcal{O}(n^{\mathcal{O}(1)})$, where n is the size of the input. i.e. its execution time has a polynomial bound.

Proposition 3.2.1 ([31]). Prediction *is in **P**.*

Proof. The size of the configuration in dimension d is $\mathcal{O}(t^d)$, then each iteration use $\mathcal{O}(t^d)$ time in a sequential machine and we need to repeat these once for each iteration. So, the total iterations steps are $\mathcal{O}(t^{d+1})$.

□

The class **NC** (Nick's class) is a subclass of **P**, consisting of all problems solvable by a *fast-parallel algorithm*. A fast-parallel algorithm is one that runs in a parallel random access machine (PRAM) (model of share-memory multi-processor computer, where each processors accesses asynchronously to the memory, for more details see [44]), in poly-logarithmic time (i.e. in time $\mathcal{O}((\log n)^{\mathcal{O}(1)})$) using $\mathcal{O}(n^{\mathcal{O}(1)})$ processors or equivalently it can solve by a *Boolean circuit* with poly-logarithmic depth and a polynomial number of gates, assigning one processor by gate and solving the circuit by levels.

Proposition 3.2.2. ***NC*** $\subseteq$ ***P***.

Proof. It is enough iterating gate by gate starting by the first level in the circuit. Given that there is a polynomial number of gates the algorithm runs in a polynomial time in a sequential machine. □

It is a wide-believed conjecture that the inclusion is proper [43]. Indeed, **NC** = **P** would imply that for any problem solvable in polynomial time, there is a parallel algorithm solving that problem *exponentially faster*.

The problems in **P** that are the most likely not to belong to **NC** are the **P**-complete problems. A problem p is **P**-complete if it is contained in **P** and every other problem in **P** can be reduced to p via a function computable in logarithmic-space. A very used **P**-complete problem is to calculate the value of a Boolean circuit, called circuit value problem (CVP) in which the inputs are a Boolean circuit and an assignment of values for their inputs, and you want to know the value obtained to the end of the circuit.

Circuit Value Problem
Input: A Boolean circuit and an assignment of values for their inputs.
Question: Is 1 the output value?

Note that if we can reduce CVP to another decision problem, then this is also **P**-complete. For further details we refer to the books of [43, 45].

Example 3.2.2. Examples where Prediction is **P**-complete are: Majority Vote CA for dimension 3 or higher [31], Life without Death [30] and Elementary CA Rule 110 [32]. Examples where Prediction is in **NC** are some case of Bootstrap Percolation Model [39].

Thus, under the assumption that **NC** $\neq$ **P**, there is not a faster parallel algorithm that computes the T-th iteration of Majority Vote CA, Life without Death or Elementary CA Rule 110, but for some case of Bootstrap Percolation Model there is a way to compute if a cell is in state 1 at time T faster than iterating the CA.

3.2.3 Sequential Computation

There is a kind of problem larger than **P**, called **NP** (for *nondeterministic polynomial time*), which consists of all decision problems that can be solved by a non-deterministic Turing Machine in polynomial time. From the definition of **NP** it is clear that $\mathbf{P} \subseteq \mathbf{NP}$, but the question of the equality of these two sets is open. Probably this should be one of the most important questions in computer science[1], because of its implications in cryptography, optimization, among others.

There are also the **NP**-complete problems, those **NP** problems to which any **NP** problem can be reduced in polynomial time. This way if there is an **NP** problem that is not in **P**, then all the complete **NP** problems are not in **P**. Thus, just as **NC** is considered the kind of problems that can be solved efficiently on a parallel machine and **P**-complete problems are considered inherently sequential problems, unless $\mathbf{NC} = \mathbf{P}$, **P** is considered the kind of problems that can be solved efficiently on a sequential machine and **NP**-complete would be the kind of problems that are difficult to solve on a sequential machine.

The classical **NP**-complete problem is the satisfiability of a Boolean formula (SAT).

Satisfiability Problem (SAT)
Input: A Boolean formula.
Question: Is there an input value such that the output value is 1?

A Boolean formula is built from variables, AND operators (conjunction, denoted by $\wedge$), OR (disjunction, denoted by $\vee$), NOT (negation, denoted by $\neg$), and parentheses. A formula is said to be satisfiable if it can become TRUE (1) by assigning appropriate logical values (i.e., TRUE (1), FALSE (0)) to its variables. From now on we will consider that all Boolean formulas are in their normal conjunctive form (CNF), i.e. a conjunction of disjunctions or they are ANDs of ORs formulas.

Proposition 3.2.3 (Cook-Levin theorem [43])**.** *SAT is **NP**-complete.*

3.2.4 Parallel subroutines

In this subsection, we will give some **NC** algorithms that we will use as subroutines of our fast-parallel algorithm.

Prefix-sum

First, we will study a general way to compute in **NC** called *prefix sum algorithm* [44]. Given an associative binary operation $*$ defined on a group G, and an array $A = (a_1, \dots, a_n)$ of n elements of G, the prefix sum of A is the vector B of dimension n such that $B_i = a_1 * \cdots * a_i$. Computing the prefix sum of a vector is very useful. For example, it can be used to compute the parity of 1s in a Boolean array by choosing $* = \veebar$, the presence of a nonzero coordinate in an array by choosing $* = \vee$, etc.

Proposition 3.2.4 ([44])**.** *There is an algorithm that computes the prefix-sum of an array of n elements in time $\mathcal{O}(\log n)$ with $\mathcal{O}(n)$ processors.*

Connected components

The following propositions state that the connected, bi-connected and tri-connected components of an input graph G can be computed by fast-parallel algorithms.

[1] The question $\mathbf{P} = \mathbf{NP}$? is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute, each of which carries a US$1,000,000 prize for the first correct solution.

Proposition 3.2.5 ([44])**.** *There is an algorithm that computes the connected components of a graph with n vertices in time $\mathcal{O}(\log^2 n)$ with $\mathcal{O}(n^2)$ processors.*

Proposition 3.2.6 ([46])**.** *There is an algorithm that computes the bi-connected components of a graph with n vertices in time $\mathcal{O}(\log^2 n)$ with $\mathcal{O}(n^3/\log n)$ processors.*

Proposition 3.2.7 ([46])**.** *There is an algorithm that computes the tri-connected components of a graph with n vertices in time $\mathcal{O}(\log^2 n)$ with $\mathcal{O}(n^4)$ processors.*

Vertex level algorithm

Given a rooted tree we are interested in computing the level *level*(v) of each vertex v, which is the distance (number of edges) between v and the root r. The following proposition shows that there is a fast-parallel algorithm that computes the level of every vertex of the graph.

Proposition 3.2.8 ([44])**.** *There is an algorithm that computes, on an input rooted tree (T, r) the level(v) of every vertex $v \in V(T)$ in time $\mathcal{O}(\log n)$ and using $\mathcal{O}(n)$ processors, where n is the size of T.*

All pairs shortest paths

Given a graph G of size n. Name $v_1, \dots, v_n$ the vertices of G. A matrix B is called an *All Pairs Shortest Paths matrix* if $B_{i,j}$ corresponds to the length of a shortest path from vertex v_i to vertex v_j. The following proposition states that there is a fast-parallel algorithm computing an All Pairs Shortest Path matrix of an input graph G.

Proposition 3.2.9 ([44])**.** *There is an algorithm that computes all Pairs Shortest Paths matrix of a graph with n vertices in time $\mathcal{O}(\log^2 n)$ with $\mathcal{O}(n^3 \log n)$ processors.*

3.2.5 Decision problems in cellular automata

To our knowledge, the first study related with the computational complexity of Cellular Automata was done by E. Banks. In his PhD thesis he studied the possibility for simple Cellular Automata in two dimensional grids, to simulate logical gates. If such simulation a is possible, the automaton is capable of universal Turing computation [47]. Directly in the context of PREDICTION problems C. Moore et al [31] studied the Majority Automata. They proved that for a given $T \geq 0$, PREDICTION is **P**-complete in three and more dimensions, but the complexity remains open in two dimensions.

In the case of freezing CA with $|Q|$ state we have the following property: given a periodical configuration with N cells, it arrives to a fixed point at time $|Q|N$, because each cell can change at most $|Q|$ times. We will study this property in Chapter 4. Thus after $|Q|N$ each cell was changed or remains in its initial state forever. The cells that never changes are called *stable* and it induces the following decision problem, called STABILITY, and this is to know if, given a configuration and a cell, the cell is stable.

STABILITY
Input: A finite configuration x of dimensions $n \times n$ and a site $u \in [n] \times [n]$ such that $x_u = 0$.
Question: Is u stable for configuration $c = c(x)$?

Analogous to PREDICTION, if we find an algorithm faster than polynomial that decides STABILITY for a freezing cellular automaton, then we have an algorithm capable of calculating the fixed point of the freezing cellular automaton faster than iterating it $|Q|N$ times.

Also, we define STABILITY for asynchronous freezing cellular automata, called Asynchronous Stability (ASYNCSTABILITY). In this case we say that a cell is stable if for any updating scheme the cell remains in its initial state forever, then the decision problem is as follow:

AsyncStability
Input: A finite configuration x of dimensions $n \times n$ and a site $u \in [n] \times [n]$ such that $x_u = 0$.
Question: Does there exists an updating scheme σ and $T > 0$ such that $F^{\sigma(T)}(c(x))_u = 1$?

In general, this problem is **NP**. Note that if the answer to AsyncStability is *Accept*, then there is an update scheme σ that changes every time the local function is applied to a cell. This update scheme reaches a fixed point at most $n^2Q = \mathcal{O}(n^2)$ iterations. If we build a Turing machine that applies the local function in a non-deterministic way each cell, if the answer to AsyncStability is *Accept*, then if the Turing machine applies the local function following σ, then in $\mathcal{O}(n^2)$ it already changes the cell u and decides.

Finally, for FCA, we consider a weaker version of Stability, in which we ask if, from an initial configuration, the fixed point that it reaches is equal to a second given configuration. This problem is called Reachability and is weaker than Stability, because if we can solve Stability in all the cells, it would be enough to compare the stability result cell by cell with the configuration that we want to know if it is reached.

Reachability
Input: Two finite configurations x and y^∞ of dimensions $n \times n$ when y^∞ a fixed point.
Question: Does there exists $t > 0$ such that $F^t(x) = y^\infty$?

We can think of Stability as the computing of the fixed point of an initial configuration and Reachability as the verification of the fixed point of an initial configuration.

3.3 Simulation Between Cellular Automata and Universality

Various classification ideas have been defined to categorize CAs. Some have been classified qualitatively, such as [48], according to properties or behaviors observed in them; comparing the time it takes a computer to calculate its t-th iteration or the quantity of information that two processors must exchange to calculate the state of a cell at some time.

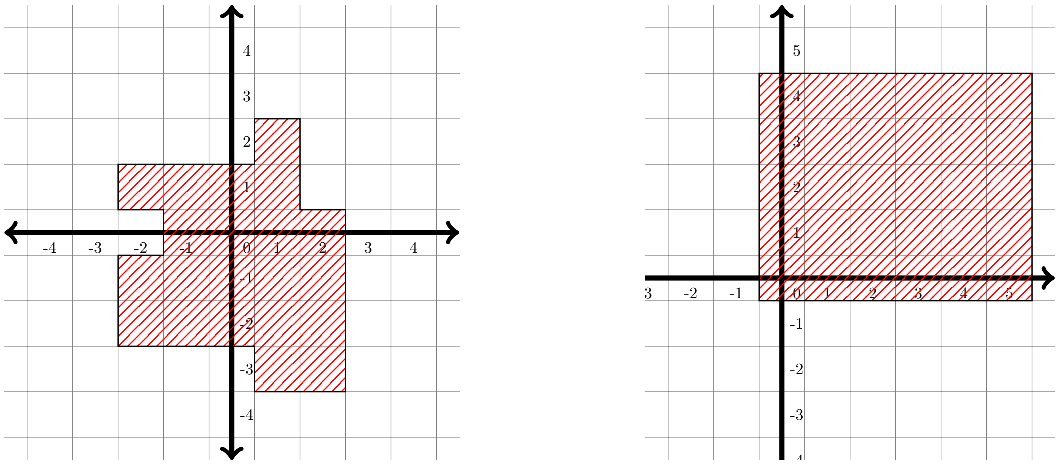
Another way to classify CAs is according to their ability to "simulate other CAs". Different criteria have been used for the idea of simulation, such as transformations between states, cell grouping, time scales, etc. To our knowledge, the most general the notion of simulation is given by M. Delorme, J. Mazoyer, N. Ollinger, G. Theyssier [25, 24] who, through three basic transformations, hierarchize the CA. A cellular automata with the highest hierarchy in a class, i.e. it is capable of simulate all the other CA that share the same class as him is called *intrinsically universal cellular automata* (for this class).

3.3.1 Geometric transformations

The simulation relation that we will study is based mainly on the idea of grouping different cells of the space-time diagram into new macro-cells, which will be the cells of other more complex CA that will simulate the cellular automata.

Definition 3.3.1. An *pattern* $\mathcal{P}$ is any finite subset of $\mathbb{Z}^d$. Given $m \in \mathbb{N}^d$ we define the m-rectangular pattern as the pattern

$$\boxplus_m = \{0, ..., m_1\} \times ... \times \{0, ..., m_d\}.$$



(a) Example of a pattern in $\mathbb{Z}^2$.

(b) Pattern $\boxplus_{(5,4)}$ in $\mathbb{Z}^2$.

Figure 3.11: Different patterns in $\mathbb{Z}^2$.

Definition 3.3.2. We say $V = (v_1, ..., v_d) \in (\mathbb{Z}^d)^d$ is a *base* of $\mathbb{Z}^d$ if $\{v_i : i = 1, ..., d\}$ is linearly independent. In particular, given $m \in \mathbb{N}^d$ we define the *base m-rectangular* as

$$\square_m = (m_1 e_1, ..., m_d e_d).$$

Example 3.3.1. $\square_{(1,...,1)}$ is the canonical base of $\mathbb{Z}^d$.

Example 3.3.2. $\square_{(1,2,3)} = \left(\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix} \right).$

Geometrically, $\square_m$ are the edges touching the origin in the hyper-rectangle $\boxplus_m$.

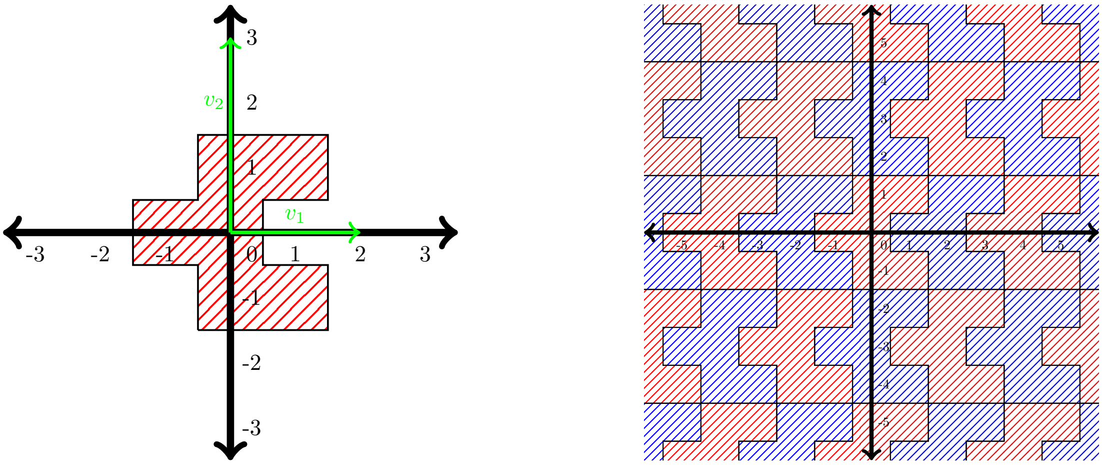
Note that any base $V = (v_1, ..., v_d) \in (\mathbb{Z}^d)^d$ defines a “linear” transformation $T_V : \mathbb{Z}^d \to \mathbb{Z}^d$, where

$$\forall z \in \mathbb{Z}^d, T_V(z) = Vz = \sum_{i=1}^{d} z_i v_i$$

considering the base V as a square matrix of size d.

Also, in contrast to linear transformations in $\mathbb{R}$, there is no property that $T_V : \mathbb{Z}^d \to \mathbb{Z}^d$ is injective if and only if T_V is surjective, but, if V is base, then T_V is injective, so it is bijective in $T(\mathbb{Z}^d)$.

Definition 3.3.3. A *tiling* of $\mathbb{Z}^d$ is the pair $(\mathcal{P}, V)$ where $\mathcal{P}$ is a pattern and V is a base of $\mathbb{Z}^d$ such that the family $\{\mathcal{P} + T_V(z)\}_{z \in \mathbb{Z}^d}$ is a partition of $\mathbb{Z}^d$.



(a) $\mathcal{P} = \{(0,0),(0,1),(1,1),(1,-1),(0,-1),(-1,0)\}$ and base $V = ((2,0),(0,3))$.

(b) Pattern tessellating the plane.

Figure 3.12: Example of a tiling with pattern $\mathcal{P}$ and base $V = (v_1, v_2)$

Definition 3.3.4. Given a tiling $(\mathcal{P}, V)$ and a set of states Q, an *packaging function* $\langle \mathcal{P}, V \rangle$ is the function

$$\langle \mathcal{P}, V \rangle : Q^{\mathbb{Z}^d} \to (Q^{\mathcal{P}})^{\mathbb{Z}^d}$$

where $\forall \alpha \in Q^{\mathbb{Z}^d}, \forall z \in \mathbb{Z}^d, \langle \mathcal{P}, V \rangle (\alpha)_z(p) = \alpha_{T_V(z)+p}, \forall p \in \mathcal{P}$.

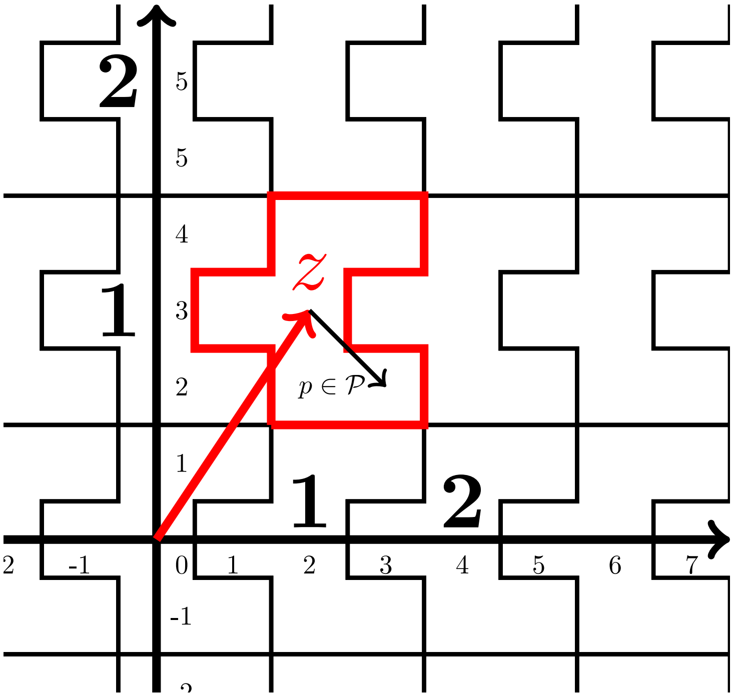


Figure 3.13: Example of packaging function. $\mathcal{P} = \{(0,0),(0,1),(1,1),(1,-1),(0,-1),(-1,0)\}$ and base $V = ((2,0),(0,3))$. If $z = (1,1)$, then $\langle \mathcal{P}, V \rangle (\alpha)_z$ is the function $p \in \mathcal{P} \to \alpha_{T_V(z)+p}$. In this case $p \in \mathcal{P} \to \alpha_{(2,3)+p}$. The red cells represent q single cell in $\langle \mathcal{P}, V \rangle (\alpha)$.

The previous definition meaning that, first a packaging function takes a configuration and divides this in pattern configurations and then the packaging function create another configuration where each cell in this new configuration is a pattern configurations on the initial configuration.

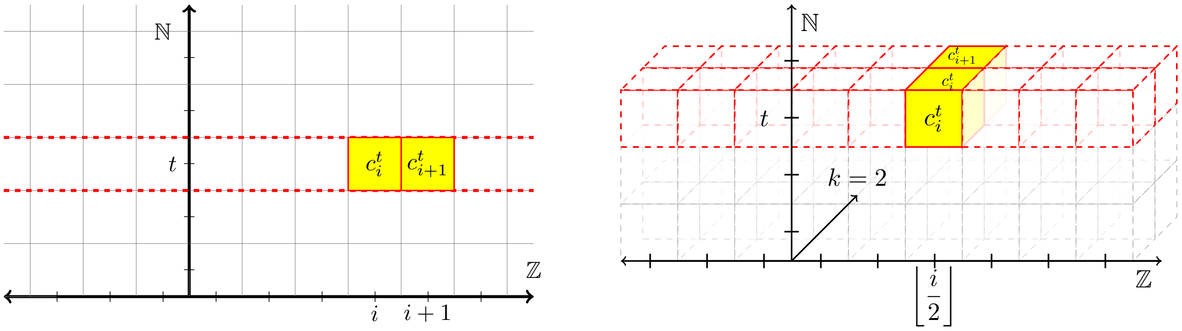
Proposition 3.3.1 ([24, 25])**.** *Let $(\mathcal{P}, V)$ be a tiling and Q a set of states, then $\langle \mathcal{P}, V \rangle$ is bijective.*

Definition 3.3.5. A *geometrical transformation* is a pair (k, Λ), with $k \in \mathbb{N}$ and $\Lambda : \mathbb{N} \times \mathbb{Z}^d \to (\mathbb{N} \times \mathbb{Z}^d)^k$. The *space-time diagram transformation* over set of states Q by a geometrical transformation (k, Λ) is the function $\overline{\Lambda} : Q^{\mathbb{N} \times \mathbb{Z}^d} \to Q^{(\mathbb{N} \times \mathbb{Z}^d)^k}$ defined as $\overline{\Lambda}(c)_z = (c_{\Lambda(z)_1}, c_{\Lambda(z)_2}, ..., c_{\Lambda(z)_k})$

For example, the following are geometrical transformation and space-time diagram transformation respectively.

Definition 3.3.6. Given a tiling $(\mathcal{P}, V)$ with $\mathcal{P} = \{p^1, ..., p^k\} \subseteq \mathbb{Z}^d$ and $V \in (\mathbb{Z}^d)^d$, we define the function *packaging*

$$\begin{array}{rcl} \mathbf{P}_{\mathcal{P},V} : \mathbb{N} \times \mathbb{Z}^d & \to & (\mathbb{N} \times \mathbb{Z}^d)^k \\ (t, z) & \mapsto & \mathbf{P}_{\mathcal{P},V}(t, z) = ((t, p^1 + T_V(z)), (t, p^2 + T_V(z)), ..., (t, p^k + T_V(z))) \end{array}$$



(a) Original space-time diagram.

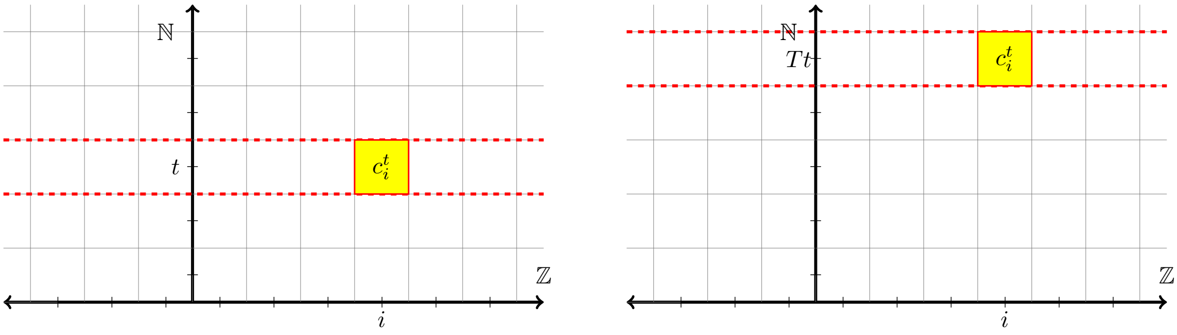
(b) Packaged space-time diagram.

Figure 3.14: Example of a packaging on a configuration c, in this case the packaging is $\langle \boxplus_2, \square_2 \rangle$.

Proposition 3.3.2 ([24, 25])**.** *Given a CA $A = (\mathbb{Z}^d, f, Q, N)$ with global function F and the packaging $(\mathcal{P}, V)$, the space-time diagram transformation, $\overline{\mathbf{P}_{\mathcal{P},V}}$ of A is given by the space-time diagram of $\overline{\langle \mathcal{P}, V \rangle} \circ F \circ \overline{\langle \mathcal{P}, V \rangle}^{-1}$.*

Definition 3.3.7. Given $T \in \mathbb{N}$, we define the function *cutting*

$$\begin{array}{rcl} \mathbf{C}_T : \mathbb{N} \times \mathbb{Z}^d & \to & \mathbb{N} \times \mathbb{Z}^d \\ (t, z) & \mapsto & \mathbf{C}_T(t, z) = (tT, z) \end{array}$$



(a) Original space-time diagram.

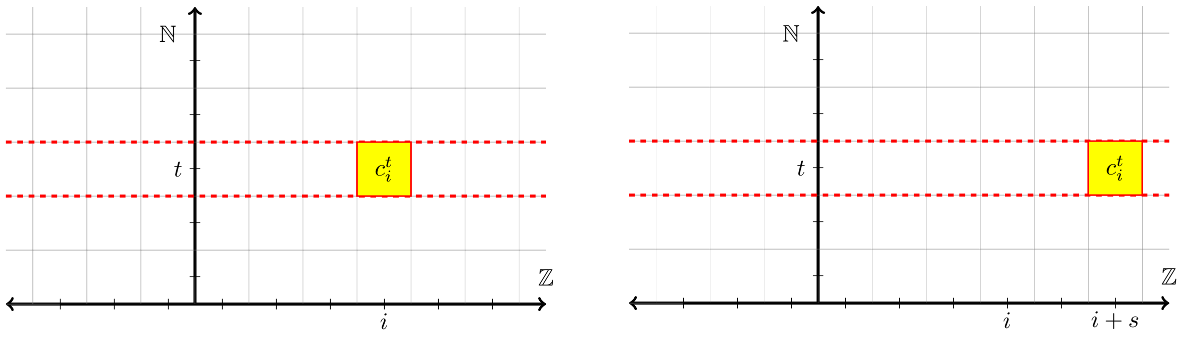
(b) Cut space-time diagram.

Figure 3.15: Example of a cutting on a configuration c.

Proposition 3.3.3 ([24, 25])**.** *Given a CA $A = (\mathbb{Z}^d, f, Q, N)$ with global function F and the cut $\mathbf{C}_T$, the space-time diagram transformation, $\overline{\mathbf{C}_T}$ is given by the space-time diagram of F^T.*

Definition 3.3.8. Given $s \in \mathbb{Z}^d$, we define the function *shifting*

$$\begin{array}{rcl} \mathbf{S}_s : \mathbb{N} \times \mathbb{Z}^d & \to & \mathbb{N} \times \mathbb{Z}^d \\ (t,z) & \mapsto & \mathbf{S}_T(t,z) = (t, z+st) \end{array}$$



(a) Original space-time diagram.

(b) Shifted space-time diagram.

Figure 3.16: Example of a shifting on a configuration c.

Proposition 3.3.4 ([24, 25]). *Given a CA $A = (\mathbb{Z}^d, f, Q, N)$ with global function F and the shifting $\mathbf{S}_s$, the space-time diagram transformation, $\overline{\mathbf{S}_s}$ is given by the space-time diagram of $\sigma_s \circ F$.*

Definition 3.3.9. Given two geometrical transforms (k, Λ) and (k', Λ'), we define the *composition* of (k, Λ) and (k', Λ') as the geometrical transformation given by

$$(k, \Lambda) \circ (k', \Lambda')(t,z) = (kk', \Lambda \circ \Lambda')(t,z) = (\Lambda_1(\Lambda'_1(t,z)), \Lambda_1(\Lambda'_2(t,z)), ..., \Lambda_k(\Lambda'_{k'}(t,z))).$$

In particular we are interested in the composition of packaging, cutting and shifting.

Definition 3.3.10. A **PCS** *transformation* is any function $\mathbf{P}_{\mathcal{P},V} \circ \mathbf{C}_T \circ \mathbf{S}_s$. Given the CA A we denote the **PCS** transformation of A by $\mathbf{P}_{\mathcal{P},V} \circ \mathbf{C}_T \circ \mathbf{S}_s$ as $A^{<\mathcal{P},V,t,s>}$.

When we find geometrical transforms to simulate CA not every transformation defines properly a simulation between CA. For example, in dimension 1 if

$$\Lambda(t,z) = \begin{cases} (t,-z) & \text{if } |z| = 2^k \; k \in \mathbb{N} \\ (t,z) & \text{otherwise} \end{cases},$$

then we have that the transformation of a space-time diagram of σ CA in general is not a space-time diagram of another CA, because the exchanging of cells in coordinates $|z| = 2^k$ is not local, then we are not interested in this kind of transforms.

Definition 3.3.11. We say that a space-time transformation (k, Λ) is *nice* if it satisfies the following conditions:

1. $\forall A \in \mathcal{A}(Q, N), \exists B \in \mathcal{A}(Q^k, M) : \overline{\Lambda}_Q(\text{Diagr}(A)) = \text{Diagr}(B)$.
2. $\forall t \in \mathbb{N}, \bigcup_{z \in \mathbb{Z}^d} \{\Lambda_1(t+1,z), ..., \Lambda_k(t+1,z)\} \not\subseteq \bigcup_{z \in \mathbb{Z}^d} \{\Lambda_1(t,z), ..., \Lambda_k(t,z)\}$.

Theorem 3.3.5 ([24]). *A space-time transformation is nice if and only if is a* **PCS** *transformation.*

3.3.2 Quasi-order and simulation

Now that we have defined how we will group the cells, we will define the notion of simulation through the definition of nice geometrical transformation, for cell grouping, and the concepts of sub-automaton and quotient, for the relationship between cellular automata. The notion of simulation is an extension of the definition of sub-automata and quotient, so that a greater variety of cellular automata can be related to this definition, but without trivializing the relationship.

Definition 3.3.12 (sub-automata and quotient)**.** Let $A = (\mathbb{Z}^d, f_A, Q_A, N_A)$ and $B = (\mathbb{Z}^d, f_B, Q_B, N_B)$ be two CA. We say that

- A is a *sub-automata* of B, denoted $A \sqsubseteq B$ if there is an injective function $\phi : Q_A \to Q_B$ such that
$$\overline{\phi} \circ F_A = F_B \circ \overline{\phi}$$
where, given a configuration $\alpha \in {Q_A}^{\mathbb{Z}^d}$, we define that $\forall i \in \mathbb{Z}^d,\ \overline{\phi}(\alpha)_i = \phi(\alpha_i)$.
- A is a *quotient* of B, denoted $A \trianglelefteq B$ if there is a surjective $\psi : Q_B \to Q_A$ such that
$$\overline{\psi} \circ F_B = F_A \circ \overline{\psi}$$
where, given a configuration $\alpha \in {Q_B}^{\mathbb{Z}^d}$, we define that $\forall i \in \mathbb{Z}^d,\ \overline{\psi}(\alpha)_i = \psi(\alpha_i)$.

It is possible to prove that $\sqsubseteq$ and $\trianglelefteq$ are quasi-orders (reflexive and transitive) and that they induce an equivalence relationships, given by a bijection between the sets of states, denoted by $\equiv$.

Definition 3.3.13. Given two relations R_1 and R_2 over the set A, we define the *composition of* R_1 *with* R_2 as
$$R_1 \cdot R_2 = \{(x, y) \in A \times A : \exists z \in A, (x, z) \in R_1 \wedge (z, y) \in R_2\}$$
and we denote $\trianglelefteq \cdot \sqsubseteq = \trianglelefteq\!\!\sqsubseteq$

The following theorem justifies working only with relationships $\sqsubseteq$, $\trianglelefteq$ and $\trianglelefteq\!\!\sqsubseteq$.

Theorem 3.3.6 ([24])**.** *Denoted $\mathcal{R}$ to the set of all the finite compositions of $\sqsubseteq$ and $\trianglelefteq$, then*

- *Every relationship $R \in \mathcal{R}$ is included in $\trianglelefteq\!\!\sqsubseteq$ ($ARB \Rightarrow A \trianglelefteq\!\!\sqsubseteq B$).*
- *The only ones transitive relationship in $\mathcal{R}$ are $\sqsubseteq$, $\trianglelefteq$ and $\trianglelefteq\!\!\sqsubseteq$.*

Definition 3.3.14 (Simulation)**.** Lets A and B CA, we say that

- B *simulates injectively* to A, denoted $A \preccurlyeq_i B$, if there is a two **PCS** α and β such that
$$A^{<\alpha>} \sqsubseteq B^{<\beta>}$$
- B *simulates surjectively* to A, denoted $A \preccurlyeq_s B$, if there is a two **PCS** α and β such that
$$A^{<\alpha>} \trianglelefteq B^{<\beta>}$$
- B *simulates mixed* to A, denoted $A \preccurlyeq_m B$, if there is a two **PCS** α and β such that
$$A^{<\alpha>} \trianglelefteq\!\!\sqsubseteq B^{<\beta>}$$

Example 3.3.3 ([24])**.** Consider the following CAs

Just gliders. Two states interpreted as particles moving left (<) and right (>) evolve in a quiescent background state (□). When two opposite particles meet they annihilate, leaving a background state (□).

ECA 184. The line of cells is interpreted as a highway where states represent cars and represent free portions of highways. Cars move to the right by one cell if they can (no car present on the next cell), otherwise they don't move.

and the context-free simulation given by $\phi : \{<, >, \square\} \to \{\square, \blacksquare\}^{\{0,1\}}$ as following

- $\phi(<) = \square\square$ = ⬚
- $\phi(>) = \blacksquare\blacksquare$ = ⬛
- $\phi(\square) = \blacksquare\square$ = ⬚

then ECA 184 injective simulates Just gliders, as in Figure 3.17, where we denote ⬚ = ⊡

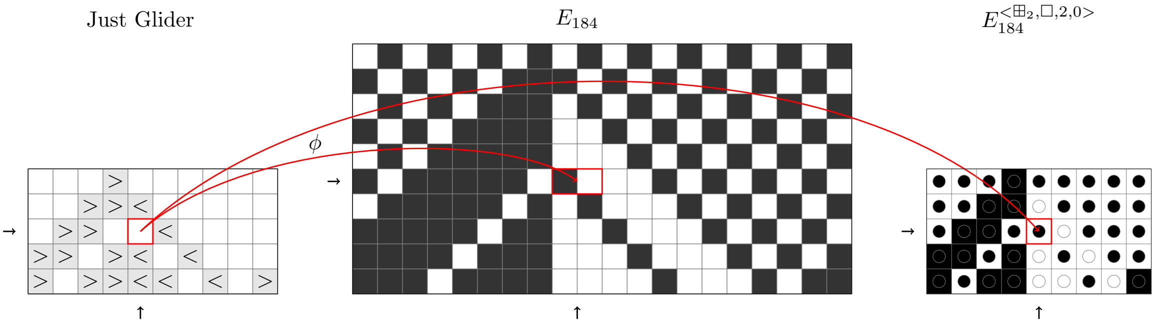


Figure 3.17: Example of injective simulation. In the left figure, we have a space-time diagram for just Glider. In the middle figure, we see the dynamic of E_{184} starting with the initial configuration given by the substitution ϕ over the initial configuration of Just Glider. In the right, we see the dynamic of E_{184} with a **PCS** transformation, starting with the initial configuration given by the substitution ϕ over the initial configuration of Just Glider.

Theorem 3.3.7 ([24]). *The relations $\preccurlyeq_i$, $\preccurlyeq_s$ and $\preccurlyeq_m$ are quasi-orders.*

The top on quasi-orders $\preccurlyeq_i$, $\preccurlyeq_s$ and $\preccurlyeq_m$ is a CA able to simulate any other CA. If there is a CA with this property we denote them $\preccurlyeq_i$*-intrinsically universal CA*, $\preccurlyeq_s$*-intrinsically universal CA* and $\preccurlyeq_m$*-intrinsically universal CA* respectively. When is clear the simulation used we say simply *intrinsically universal CA*.

Given a set of CA $\mathcal{A}$, we say that $\mathcal{U}$ is *intrinsically* $\preccurlyeq$*universal* for $\mathcal{A}$ if $\mathcal{U} \in \mathcal{A}$ and for any $A \in \mathcal{A}$, $A \preccurlyeq \mathcal{U}$, where $\preccurlyeq$ can be $\preccurlyeq_i$, $\preccurlyeq_s$ or $\preccurlyeq_m$.

The following examples consider $\preccurlyeq_i$-simulations.

Example 3.3.4 (Dimension 1)**.**

1. There exists an intrinsically universal 1D cellular automaton with 5 neighbors and 2 states [49, 47].
2. There exists an intrinsically universal 1D cellular automaton with first neighbors neighborhood and 4 states [50].
3. There exists an intrinsically universal 1D for reversible cellular automaton [21].
4. There exists an intrinsically universal 1D for conservative cellular automaton [20].
5. There exists an intrinsically universal 1D for time-symmetric cellular automaton [23].
6. There exists an intrinsically universal 1D for time-symmetric and conservative cellular automaton [1].

Example 3.3.5 (Dimension 2)**.**

1. Game of life is intrinsically universal for 2D cellular automaton [18].
2. There exists an intrinsically universal for 2D cellular automaton with von Neumann neighborhood and 2 states [49, 47].

Chapter 4

Freezing Cellular Automata

The *freezing cellular automata*, introduced in [33], are cellular automata where in the evolution the state each cell can only increase (or decrease depending on the point of view) following a relation of order. This restriction endows this family of CA with specific dynamical and computational properties. We discuss some properties of these CA like their inherent irreversibility or the properties of their fixed points. In addition, from the point of view of computability, we discuss the decrease of complexity of several problems including undecidable problems that turns decidable in freezing cellular automata.

4.1 Basic Dynamical Properties

The simulation on CA is a way to understand its dynamics. Limitation and possibilities on the simulations talk about the properties on the studied system.

Proposition 4.1.1 ([33]). *The family of freezing CA is closed under iterations, Cartesian product, sub-automaton, local factor and grouping, but not under composition by shifts.*

Proof. If a CA F is freezing then F^t is also freezing, considering the same order relation. For grouping it is enough to consider the order on the blocks given by the order cell by cell in each block grouping cells.

Now, let F and G be two FCA of dimensions d with set of states Q_F and Q_G and orders given by the relations $\leq_F$ and $\leq_G$ respectively Then the CA given by the Cartesian product of F and G is $F \times G : Q_F^{\mathbb{Z}^d} \times Q_G^{\mathbb{Z}^d} \to Q_F^{\mathbb{Z}^d} \times Q_G^{\mathbb{Z}^d}$ such that $F \times G(c_F, c_G)_z = (F(c_F)_z, G(c_G)_z)$ and considering the order in $Q_F^{\mathbb{Z}^d} \times Q_G^{\mathbb{Z}^d}$ given by $(q_F, q_G) \leq_\times (q'_F, q'_G) \Leftrightarrow (q_F \leq_F q'_F) \wedge (q_G \leq_G q'_G)$ we obtain that $F \times G$ is a FCA.

Let F be sub-automata of the FCA G by $\phi : Q_F \to Q_G$ injective. Then considering the order relation $\leq_F$ on Q_F such that $q \leq_F q' \Leftrightarrow \phi(q) \leq_G \phi(q')$ we obtain that F is a FCA with order $\leq_F$.

Let F be sub-automata of the FCA G by $\psi : Q_G \to Q_F$ surjective. Then considering the order relation $\leq_F$ on Q_F such that $q \leq_F q' \Leftrightarrow \psi^{-1}(q) \leq_G \psi^{-1}(q')$ we obtain that F is a FCA with order $\leq_F$.

Finally, for the shift cases, it is enough to consider the identity automaton as a freezing cellular automaton, but the shift automaton is not freezing.

□

The main property of FCA is that they progressively "freeze" any initial configuration. The idea is that, given that cells can only change a finite number of times, then asymptotically all cells reaches a state that can no longer go up.

Proposition 4.1.2 ([33])**.** *Given a freezing CA F and any configuration c, the sequence $(F^t(c))_t$ converges in the Cantor metric to some limit configuration $F^\infty(c)$ which is a fixed point for F.*

Proof. Since the state of any cell can only increase, each cell can only change its state a finite number of times. Therefore, starting from configuration x, for any $n \geq 0$ there exists a time t such that no cell at distance n from the center will change its state after time t. This is equivalent to the convergence of the sequence $(F^t(x))_t$. □

A corollary is that any periodic configuration reaches a fixed point in finite time.

Corollary 4.1.2.1 ([33])**.** *Let F be a freezing cellular automaton and x be a finite configuration with N cells, then $c(x)$ reaches a fixed point in $\mathcal{O}(N)$ iterations.*

Proof. Let F be a freezing cellular automaton and x be a finite configuration with N cells. It is enough to note that each cell can change at most $|Q| - 1$ times and at least one cell changes in each iteration before it reaches a fixed point, then, if there are N cells, at time $(|Q| - 1)N$ every cell was changed, and we have a fixed point. □

A CA is *balanced* if every state has the same amount of preimages through the local function, i.e. if the set of state is Q, the neighborhood is N and the set of all preimages of f is Q^N, then in a balanced CA each state has $|Q|^{|N|-1}$ preimages. If a CA is not balanced, we say that it is *unbalanced.*

Proposition 4.1.3 ([33])**.** *Given a FCA F then, exactly one of the following two statements is true:*

- *F is unbalanced.*
- *$F = Id$.*

This proposition means that the only balance FCA is the identity map.

Proof. To prove that if $F = Id$ then F is balanced is direct. Thus, we need to prove that if F is balanced then $F = Id$.

Let F be a balanced FCA and q a maximal state. Without loss of generality, suppose that its neighborhood N contains the center cell. Given that F is a FCA and q is maximal, then for all $p \in Q^N$ such that $p_\theta = q$ and $f(p) = q$. Therefore, every pattern $p \in Q^N$ such that $p_\theta = q$ is a preimage of q. There are $|Q|^{|N|-1}$ and given that F is balanced they are all the possible preimages of q.

Let Q' be the set of states obtained by removing every maximal state of Q and $q' \in Q'$ a maximal state on Q'. Given that F is a FCA q' can only remain in state q' or change to a maximal state, but this is not possible because every preimage of a maximal state has this state as center cell. Thus, a preimage of q' has a q' as a center cell, then, given that F is balanced, there are any with q' as center cell is the preimage of q'.

Inductively we obtain that $\forall p \in Q^N, f(p) = p_\theta$, then F is the identity map.

□

It was proved that the balancedness is a necessary condition for surjectivity and then for reversibility.

Proposition 4.1.4 ([51])**.** *If a CA is surjective then it is balanced.*

Thus the “freezing” behavior is intrinsically irreversible except in the trivial case of the identity.

Corollary 4.1.4.1 ([33]). *If a freezing CA is surjective then it is the identity map.*

Proposition 4.1.2 says that every orbit converges to a fixed point. This does not imply that the limit set is made of fixed points. For example, if we consider the 1D CA with states $\{0,1\}$ and local function $f(a,b,c) = \max\{a,b,c\}$ we obtain that for the configuration $c^i_z = \begin{cases} 1 & \text{if } z \leq i \\ 0 & \text{otherwise} \end{cases}$ c^{i-1}_z is a preimage, then for any $i \in \mathbb{Z}$, c^i is in the limit set, but c^i is not a fixed point ($F(c^i) = c^{i+1}$).

However, we can prove the following proposition which is not true for CA in general.

Proposition 4.1.5 ([33]). *Let F be a freezing CA which is not nilpotent. Then it possesses two distinct fixed points.*

Proof. Let M be a maximal state for the order $\leq$ on states coming from the freezingness of F, and denote by c_M the uniform configuration everywhere equal to M. Since F is freezing, c_M must be a fixed point for F.

Let $\mathcal{C}^t = \{c : F^t(c)_0 \neq M\}$, then $\mathcal{C}^t$ is a compact set for all t. Given that F is not nilpotent, then $\forall t \in \mathbb{N}, \exists c_t : F^t(c_t)_0 \neq F^{t+1}(c_t)_0 \leq M$, thus $\forall t \in \mathbb{N}, \mathcal{C}^t \neq \emptyset$. Since F is freezing then $\mathcal{C}^{t+1} \subsetneq \mathcal{C}^t$.

By Cantor's intersection theorem there is a $c^* \in \bigcap_{t=1}^{\infty} \mathcal{C}^t$, then $F^{\infty}(c^*)$ is a fixed point such that $c^*_0 \neq M$.

□

Note:

- There exists a non nilpotent CA and without fixed point, e.g. $f(x) = \neg x$.
- There exists a FCA with exactly 2 fixed points, e.g. $f(x,y,z) = \begin{cases} 1 & \text{if } x = 1 \vee y = 1 \vee z = 1 \\ 0 & \text{otherwise} \end{cases}$.
- There exists a FCA with infinite fixed points, e.g. the identity map.

4.2 Computational Complexity

In this section we will study two types of problems, those that can be solved by a computer and those that cannot. We will call the first *decidable* and the second *undecidable*. On the one hand, we will show that there are problems that in cellular automata are undecidable, but if we restrict them to freezing cellular automata they become decidable. On the other hand, we will classify the decidable problems according to the time it takes a computer to solve them, in our case in the class $\mathbf{NC} \subseteq \mathbf{P}$. Here we will also show that there are problems that reduce their complexity.

Theorem 4.2.1 ([33]). *The nilpotency problem for freezing CA is:*

- *decidable (in polynomial time) for 1D CA;*
- *undecidable for CA in higher dimension.*

Proof. First, in dimension 2 and more, the classical proof of undecidability of nilpotency (see [52]) works without any modification for freezing CA: given a Wang tile set, we build a freezing CA with a spreading error state, that checks locally if the configuration is a valid tiling and produces the error state in case of local error detection. This CA is nilpotent if and only if the tile set does not tile the plane. In dimension 1, from Proposition 4.1.5, we know that nilpotency is equivalent to the following first-order property

$$\forall x \forall y (x = F(x) \wedge y = F(y)) \Rightarrow x = y) \ .$$

From [53], it follows that any first-order property is decidable for one dimensional CAbb. □

The previous theorem differs from the general case, in which the problem of knowing whether a CA is nilpotent is undecidable [54].

Lemma 4.2.2. *Let F be a freezing cellular automaton, then the problem* Stability(F) *is in* ***P***.

Proof. The application of one step of any FCA can be simulated in polynomial time, simply computing the local function of every cell. Therefore, by corollary 4.1.2.1, computing $F^{(|Q|-1)N}(c(x))$ we obtain a fixed point, where N is the number of cells in x.

Finally we obtain that for every FCA F problem Stability(F) is in **P**. □

Proposition 4.2.3 ([33])**.** *2D freezing CA are Turing-universal and there exists a 2D freezing CA with a P-complete prediction problem.*

Proof. Any 1D CA F with states Q and neighborhood V can be simulated by a 2D freezing CA with states $Q \cup \{*\}$ as follow. Let $V' = \{(v, \ -1) : v \in V\}$. A cell in a state from Q never changes. A cell in states $*$ looks at cells in its V' neighborhood: if they are all in a state from Q then it updates to the state given by applying F on them, otherwise it stays $*$. Starting from a fully $-*$ configuration except on one horizontal line where it is a Q-configuration c_0, this 2D freezing CA will compute step by step the space-time diagram of F on configuration c_0. Then it is enough to consider the elementary CA with rule 110 where Prediction is **P**-complete [32] or any other 1D CA where Prediction is **P**-complete. □

Note that, for the rule 110, 0 is a quiescent state, then it is not necessary to add the state $*$. If we work on a 2D configuration with all cells in state 0, except on a line then we obtain a 2D CA where Prediction is **P**-complete.

Proposition 4.2.4 ([33])**.** *The prediction problem of any 1D freezing CA is NLOGSPACE.*

Proof (sketch).
Consider (without loss of generality) a 1D freezing CA F of radius 1 with k states. To know that F gives state q after n steps on some input u of size $2n+1$ it is sufficient to guess a sequence of $2n+1$ columns of states $(C_i)_{-n \le i \le n}$, where C_i is of height $n - |i| + 1$, and to check that they form a (triangular) valid space-time diagram of F on input u that leads to state q (last letter of C_0). The key observation is that, since F is freezing, at most k changes of state can occur in any column, hence a column of height n can be represented in space $\mathcal{O}(\log(n))$ (constant list of changes, each given by the time step at which it occurs and the new state). We give the non-deterministic LOGSPACE algorithm to solve the prediction problem of F:

Algorithm 1 Solving Prediction 1D

```
Input: $x$ a finite configuration of dimensions in $Q^{[-n,\dots,n]}$.
To guess columns $C_{-n}$ and $C_{-n+1}$ and set current position to $-n+1$.
for $p$ with $-n+1 \le p < n$ do
    To guess columns $C_{p+1}$.
    for $i$ with $0 \le i < |C_p|$ do
        To check that $f(C_{\mathbf{P}-1}[i], \ C_p[i], \ C_{p+1}[i]) = C_p[i+1]$
    end for
end for
return $C_0[n]$
```

Note that the compatibility test between neighboring columns in the main loop can be done using the compact log-space representation of columns. □

4.3 Turing universality

In one-dimensional CA, the elementary 110 CA is Turing-universal, i.e. can simulate any Turing machine, then the 1D or higher dimension CA are Turing-universal, but the elementary 110 CA is not freezing, then we need another way to prove its Turing-universality.

First, we note that we can simulate any one-dimensional CA by a two-dimensional freezing CA, simulating its space-time diagram on the two-dimensional cellular space.

Lemma 4.3.1 ([33]). *Given a one-dimensional CA $F = (1, Q, N, f)$ there is a two-dimensional 1-change freezing CA $F' = (2, Q', N', f')$ such that for any configuration $c \in Q^{\mathbb{Z}}$ there is a configuration $c' \in Q^{\mathbb{Z}^2}$ such that*

$$F^t(c)_z = F^\infty(c')_{(z,t)}$$

Proof. It is enough to follow the ideas in proposition 4.2.3 to simulate the rule 110 on a two-dimensional FCA and use the Turing universality of this rule. □

Theorem 4.3.2 ([33]). *2D or higher dimension freezing CA are Turing-universal.*

To simulate a Turing machine in a one-dimensional FCA, first we study another Turing-complete system, the *counter machine*, then we will give an abstract version of its simulation on a CA and then we will simulate this in a FCA.

Theorem 4.3.3 ([33]). *1D freezing CA are Turing-universal.*

4.3.1 Turing universality on Counter Machine

A k-counter machine (CM) [11] is a kind of only-read multi-tape Turing machine whose heads can only differentiate the initial position (the leftmost squares denoted Z) from the rest of the positions on the tape (denoted P). In each step the CM only can read or move a head, then it can change its state.

Minsky prove that the k-counter machines can simulate any Turing machine [11].

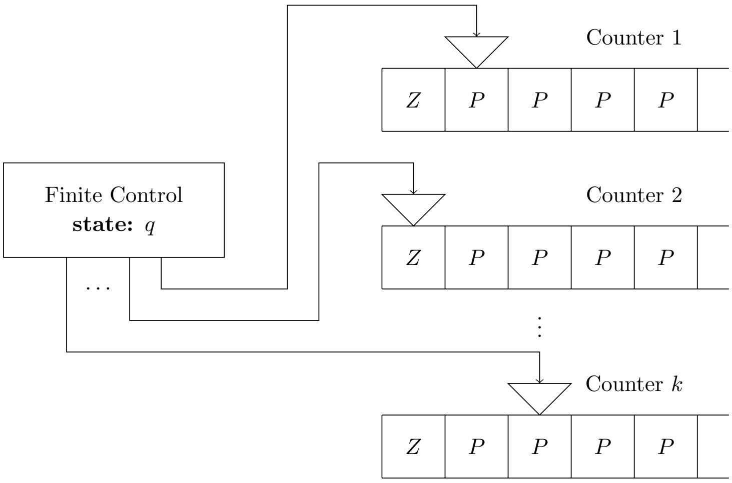


Figure 4.1: Scheme of a k-counter machine.

Definition 4.3.1. A *k-counter machine* is a tuple $M = (k, Q, \delta, q_0, q_f)$ where,

- $k \in \mathbb{N}$.
- Q is a finite set.
- $q_0 \in Q$ is an initial state.
- $q_f \in Q$ is a final state.
- $\delta \subseteq (Q \times \{1, ..., k\} \times \{Z, P\} \times Q) \cup (Q \times \{1, ..., k\} \times \{-, O, +\} \times Q)$.

On one hand, if $(q, i, Z, q') \in \delta$ meaning that if the CM is in state q and in the i-th head is in the starting cell (cell with index Zero) then change to state q'. If we change Z by P then we check if the head is not in the starting cell (cell with Positive index). These are reading actions.

On another hand, if $(q, i, +, q') \in \delta$ means that if the CM is in state q then it moves the i-th head one cell to the right. This represent adds to the i-th counter. Analogously we move the i-th head to the left and we do not move the head with $(q, i, -, q') \in \delta$ and $(q, i, 0, q') \in \delta$ respectively. These represent remove one to counter and do not change the value in the counter. These are moving actions.

We can represent δ in a labeled graph, as in the example 4.3.1, called Finite state representation, analogous to the Turing machines, where the vertices are the states and a label (i, Z) on an edge (q, q') meaning that q changes to q' if the i-th head is in the stating cell. A label $(i, +)$ on an edge (q, q') meaning that q changes to q' and moves the i-th head to the right. Analogously we define the labels (i, P),$(i, -)$ and $(i, 0)$.

Example 4.3.1. To consider the following 2-counter machine $M = (2, \{q_0, q_f, 1, 2, 3\}, \delta, q_0, q_f)$, depicted in Figure 4.2.

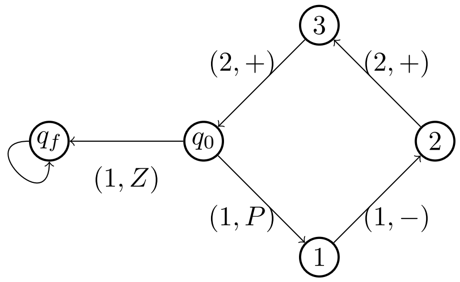


Figure 4.2: Finite state representation of a 2-counter machine multiplying by 2 the value in the first counter (distance from the head to the first cell in the first counter). The output is the distance from the head to the first cell in the second counter.

This CM checks if the first counter (input) is 0. If it is not 0, then it moves the head of the first counter to the left (remove one of the counter), then it moves twice the head of the second counter (add 2 to the second counter). Repeat this process until the first head arrives to the starting cell.

This is equivalent to the following code, where we call C_i to the value in the i-th head (its position on the counter).

Algorithm 2 Multiplying by 2 CM

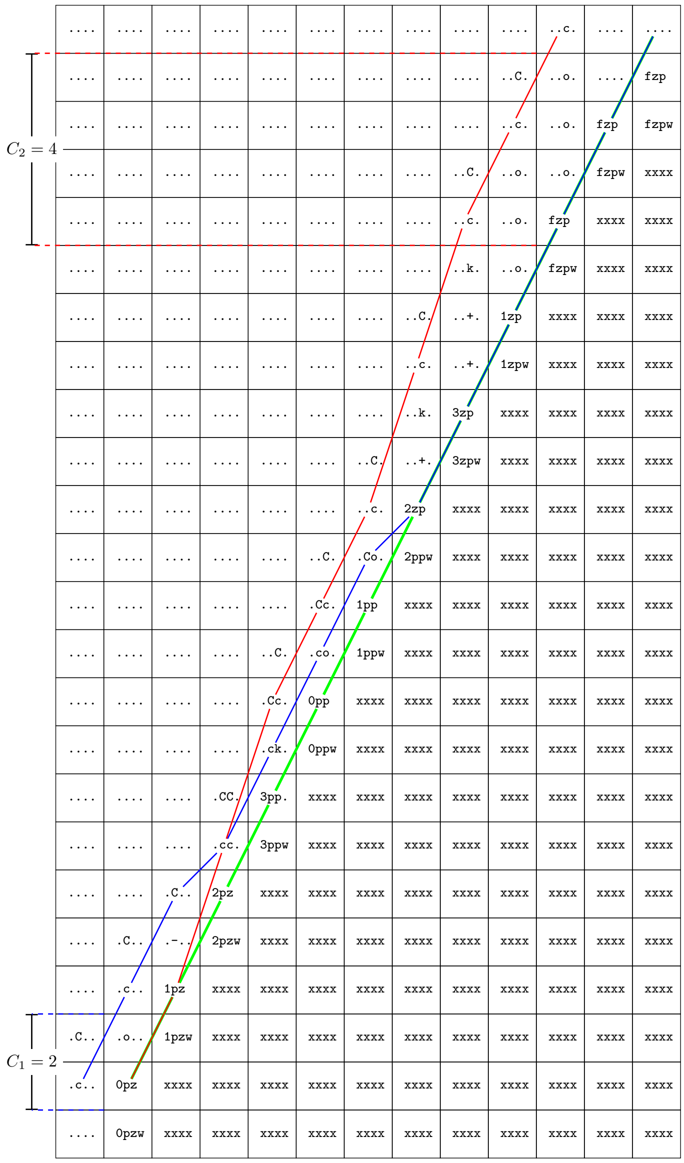
```
Input: $C_1 = n$ and $C_2 = 0$.
while $C_1 > 0$ do                  % $q_0 \to 1$ or $q_0 \to q_f$
    Remove one $C_1$                % $1 \to 2$
    Add one $C_2$                   % $2 \to 3$
    Add one $C_2$                   % $3 \to q_0$
end while
return $C_2$                        % $q_0 \to q_f$
```

Minsky [11] proves that any Turing machine can be simulated by a 2-counter machine, so both models are equivalent.

Theorem 4.3.4 ([11])**.** *For any Turing machine T there is a 2-counter machine M that simulates T.*

Lemma 4.3.5. *For each k-counter machine, we can build a CA simulating it.*

Example 4.3.2. The following example shows as work the simulation of the k-counter machine in the example 4.3.1 by a CA.



....	....	....	....	....	....	....	....	....	....	..c.	....	....
....	....	....	....	....	....	....	....	....	..C.	..o.	....	fzp
....	....	....	....	....	....	....	....	....	..c.	..o.	fzp	fzpw
....	....	....	....	....	....	....	....	..C.	..o.	..o.	fzpw	xxxx
....	....	....	....	....	....	....	....	..c.	..o.	fzp	xxxx	xxxx
....	....	....	....	....	....	....	....	..k.	..o.	fzpw	xxxx	xxxx
....	....	....	....	....	....	....	..C.	..+.	1zp	xxxx	xxxx	xxxx
....	....	....	....	....	....	....	..c.	..+.	1zpw	xxxx	xxxx	xxxx
....	....	....	....	....	....	....	..k.	3zp	xxxx	xxxx	xxxx	xxxx
....	....	....	....	....	....	..C.	..+.	3zpw	xxxx	xxxx	xxxx	xxxx
....	....	....	....	....	....	..c.	2zp	xxxx	xxxx	xxxx	xxxx	xxxx
....	....	....	....	....	..C.	.Co.	2ppw	xxxx	xxxx	xxxx	xxxx	xxxx
....	....	....	....	....	.Cc.	1pp	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
....	....	....	....	..C.	.co.	1ppw	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
....	....	....	....	.Cc.	0pp	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
....	....	....	....	.ck.	0ppw	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
....	....	....	.CC.	3pp.	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
....	....	....	.cc.	3ppw	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
....	....	.C..	2pz	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
....	.C..	.-..	2pzw	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
....	.c..	1pz	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
.C..	.o..	1pzw	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
.c..	0pz	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
....	0pzw	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx

Figure 4.3: A 1D freezing CA simulating a 2-CM

Note that our CA simulating a counter machine is freezing, because for each part of each column on the simulation there is a finite numbers of changes.

Theorem 4.3.6 ([33])**.** *For each k-counter machine, we can build a* freezing *CA simulating it.*

4.3.2 Freezing cellular automata with P-complete Stability problem

To prove that a certain problem is **P**-complete it is enough to prove that it is possible to reduce another **P**-complete problem to it. One option would be to make the reduction from the circuit value problem, but there are several other **P**-complete problems more appropriate on which to make the reduction. The one that we will use in the next cases will be the monotone circuit value problem (MCVP). This problem is similar to CVP, but we restrict our input circuits to monotonous Boolean circuits, i.e. circuits where there are only AND and OR gates.

Monotone Circuit Value Problem
Input: A monotone Boolean circuit and an assignment of values for their inputs.
Question: Is 1 the output value?

To build a circuit on a configuration we need:

- wires,
- wires duplicator,
- turning wires and
- gates.

In dimension 2 it is also necessary that the cables can be crossed. One way to achieve this is to use a gate with two inputs and two outputs $c(p, q) = (q, p)$. This gate can be constructed from XOR gates ($\veebar$) as follows:

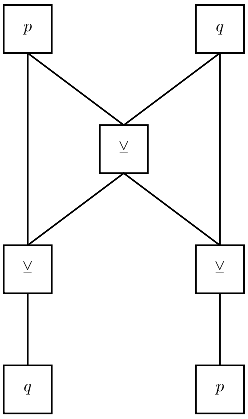


Figure 4.4: Building a crossing gate from XOR gates.

Freezing Majority-Vote Cellular Automata

In [31] is studied the *Majority-Vote Cellular Automata* in dimension 3, and is showed that Prediction is **P**-complete for this rule. Its local function is the following:

$$f(c_{VN_3(z)}) = \begin{cases} 1 & \text{if} \displaystyle\sum_{w \in VN_3(z)} c_w \leq 4 \\ 0 & \text{otherwise} \end{cases} .$$

Theorem 4.3.7 ([31])**.** Prediction *is* ***P****-complete for Majority-Vote Cellular Automata in dimension 3.*

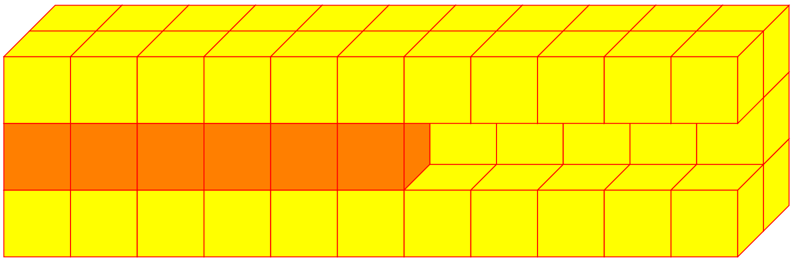
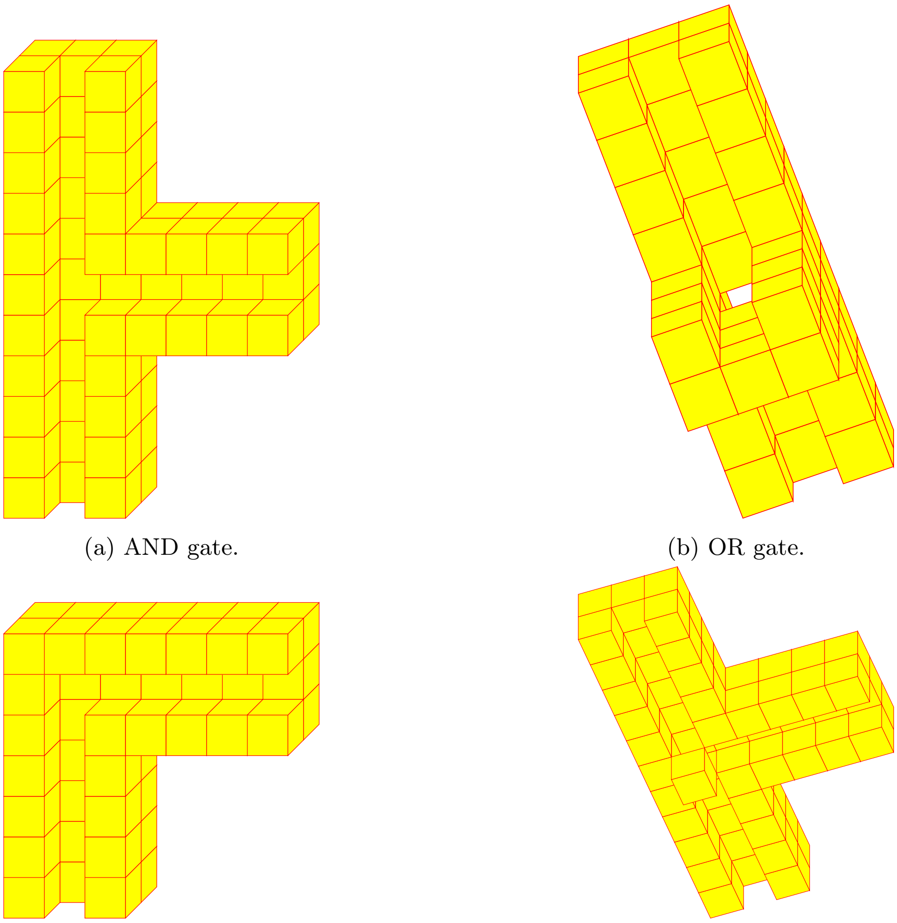


Figure 4.5: Signal (oranges cells) traveling on a wires (yellows cells). Transparent cells are cells in state 0.

Proof. We will build a wire and signal on a configuration of Majority-Vote Cellular Automata. For a wire to the right we will consider a structure with six cells in state 1, the cells $\{(0, \pm 1, 0), (0, \pm 1, -1), (0, 0, -1)\}$, then coping this figure n cell to the right we obtain a wire of length n. Note that any cell in state 1 in a wire remains in this state. Now if we change the cell $(0, 0, 0)$, then the cell $(1, 0, 0)$ has 4 neighbors in state 1, then in the next step this cell will change to 1. This is a signal traveling on a wire, see Figure 4.6



(a) AND gate.

(b) OR gate.

(c) Wire turning.

(d) Duplicator.

Figure 4.6: Elements to build any monotone circuit on a configuration of Majority-Vote Cellular Automata.

□

In the case of the Majority-Vote Cellular Automata build a crossing gate is not needed, because it is enough to turn a wire to build a bridge over another wire using the third dimension.

Now, we need to know if is possible to build a configuration simulating circuit using a log-space algorithm. For this, we consider without loss of generality that the circuit is given by layers where the gates in the layer l are input only for gates on layer higher that l. We denote a gate in the layer l by g^l.

The goal is to put the gates in the plane XY and draw the i-th wire in the coordinates $(x, y, 10i)$, except when entering and leaving a gate, in this case, the wire starts in its gate and travel until you reach its coordinates $(x, y, 10i)$. Thus, the wires do not cross each other. We say that the wire is a *deep* i.

The following algorithm allows us to understand the log-space reduction:

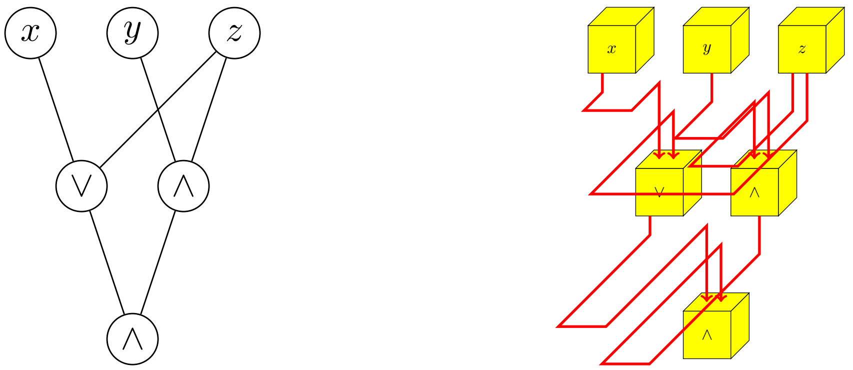
Algorithm 3 LOGSPACE reduction for Majority-Vote Cellular Automata

```
Input: A Boolean circuit with gates $V = \{g_1^1, ..., g_n^L\}$ and connections $E \subseteq V \times V$.
 1: $w = 0$.
 2: for $l = 1, ..., L$ do
 3:     for $i$ such that $g_i^l$ is a gate do
 4:         to put on the plane $XY$ at position $(20i, 20l)$ the gate $g_i^l$, as the Figures 4.6a and 4.6b.
 5:     end for
 6: end for
 7: for $(g_i^l, g_j^m) \in E$ do
 8:     to draw a wire at deep $w$ connecting the gates $g_i^l$ and $g_j^m$.
 9:     $w = w + 1$.
10: end for
```

The input is size $\mathcal{O}(|V|^2)$. In this algorithm the unique variables are w, l and i.

- w is at most the number of connections between the gates, i.e w is $\mathcal{O}(|V|^2)$, then we need $\mathcal{O}(\log(|V|^2))$ space.
- l is at most the number of levels in the circuit, i.e l is $\mathcal{O}(|V|)$, then we need $\mathcal{O}(\log(|V|))$ space.
- i is at most the number of gates in the circuit, i.e i is $\mathcal{O}(|V|)$, then we need $\mathcal{O}(\log(|V|))$ space.

In line 4 we place the gates in coordinates $(20i, 20l)$ not to overlap the gates. In lines 4 and 8 there are more details, as duplicate the signals, but they are possible to calculate in LOGSPACE.



(a) Circuit.

(b) Scheme of the circuit on a configuration.

Figure 4.7: Example of a simulation of a circuit by the Majority-Vote Cellular Automata.

Note that in Figure 4.6 every cell in state 1 remains in this state and when a cell changes to 1 the it remains in this state forever, then we can deduce that Stability is **P**-complete for the Freezing Majority-Vote Cellular Automata.

Corollary 4.3.7.1. Stability *is **P**-complete for the Freezing Majority-Vote Cellular Automata.*

The bootstrap percolation problem

The *bootstrap percolation* phenomena consist in a lattice with cells in states *active* or *inactive*. When there are enough active cells in the neighbors of a cell they can activate an inactive cell. This dynamics is observed in some physical systems as the model of ferromagnetism on statistical mechanics [34], fails on dense storage arrays [55] and neuronal activation on Living Neural Networks [56]. We can see that this is a freezing dynamics (only the inactive cells can belong active).

Here we will study the case when the dynamics is synchronous, on a finite arbitrary lattice.

Definition 4.3.2. We will consider the following automata network $A = (\{0,1\}, G, F)$, where for all $c \in \{0,1\}^G$

$$f(c_{N(z)}) = \begin{cases} 1 & \text{if } c_z = 1 \\ 1 & \text{if } \displaystyle\sum_{w \in N(z)} c_w > \frac{|N(z)|}{2} \wedge c_z = 0 \\ 0 & \text{if } \displaystyle\sum_{w \in N(z)} c_w \leq \frac{|N(z)|}{2} \wedge c_z = 0 \end{cases}$$

We call this automata network the *freezing strict majority rule.*

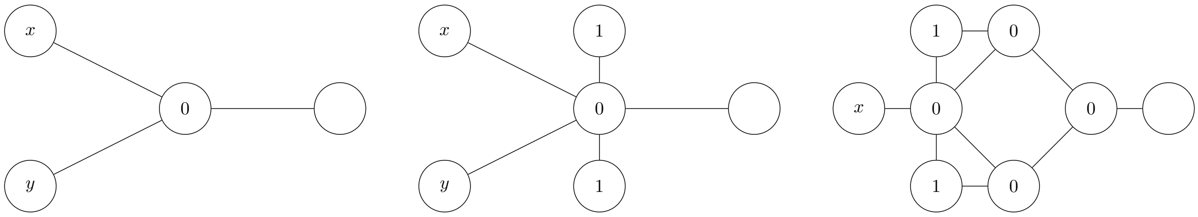
This is a special case of bootstrap percolation which corresponds to a simple model introduced by Chalupa to study properties of some magnetic materials [34] or equivalently is the bootstrap percolation model when the threshold for each cell is the strict majority of its neighborhood.

Goles, Montealegre and Todinca [39] studied Stability for the freezing strict majority rule.

Theorem 4.3.8 ([39])**.** *For the freezing strict majority rule:*

1. *On the family of graphs $G = (V, E)$ such that $\Delta(G) \geq 5$,the problem* Stability *is **P**-Complete.*
2. *On the family of graphs $G = (V, E)$ such that $\Delta(G) \leq 4$,the problem* Stability *is in **NC**.*

Proof (for 1.) Following the same steps of theorem 4.3.7 we need build the gates AND and OR. The duplicator is built using the same figure that the OR gate, but using the output as input. We do not need a crossing gate, because we can build it directly on the graph. This is not valid if we restrict this problem to planar graphs .



(a) AND Gate. (b) OR Gate - duplicator. (c) Diode.

Figure 4.8: Logic gates for a graph with maximum degree at less 5.

□

For the part 2. of the theorem we need the following definition and lemmas.

Definition 4.3.3. Given a graph $G = (V, E)$ and a configuration $c \in \{0,1\}^V$, we say that $S \subsetneq V$ is a *community* if for all $v \in S : c_v = 0$ and $\delta_{G[S]}(v) \geq 2$.

Directly from the definition of community and the fact that in a graph G with $\Delta(G) \leq 4$ the inactive cells with two inactive neighbors remains inactive, we obtain the following lemma.

Lemma 4.3.9 ([39])**.** *Let $G = (V, E)$ be a graph with $\Delta(G) \leq 4$ and a configuration $c \in \{0,1\}^V$. If $S \subseteq V$ is a community, then is stable.*

The following lemma gives a characterization of the stable sets.

Lemma 4.3.10 ([39])**.** *Let G be a graph with $\Delta(G) \leq 4$. A vertex v is stable if and only if there exists a path P of $G[0, v]$ that contains v and, if u is one of its ends, then*

1. *u belongs to a cycle in $G[0, v]$, or*
2. *$\delta(u) \leq 2$,*

where $G[0, v]$ is the community in $G[0]$ containing v and connected.

Proof. Suppose first that v is stable. We know that at most after $T = |V|$ steps, the dynamics given by c gets into a fixed point, by freezing. Let $G'[0, v]$ be the connected component of inactive vertices of G in the step T that contains v. Notice that $G'[0, v]$ is a subgraph of $G[0, v]$. Take P as the longest path in $G'[0, v]$ that contains v. Let u be an end of P. Suppose first that u has only one neighbor in $G'[0, v]$, since u is stable, u has at most one more neighbor in G, hence $\delta(u) \leq 2$. Suppose now that u has more than one neighbor in $G'[0, v]$. Since P is the longest path that contains v in $G'[0, v]$, both neighbors of u must belong to P. Then u belongs to a cycle in $G[0, v]$.

Let us prove the converse. Let P be a path in $G[0, v]$, with $v \in P$ satisfying (1) or (2). Let X be the set of vertices of P, plus, for each end-point u of P satisfying condition(1),the corresponding cycle C_u of $G'[0, v]$ containing u. Note that, since $\Delta(G) \leq 4$, X forms a community of initially inactive vertices. Indeed, each interior vertex of P (P minus its ends) has at least half of its neighbors in X, and the same holds for the vertices of the cycles added to X. If an end point u of P satisfies condition (2), it also has at least half of its neighbors in X, thus X is a community. The proof follows from lemma 4.3.9. □

Now we are ready to prove the second part of the Theorem 4.3.8.

Proof (Theorem 4.3.8 part 2.). From Lemma 4.3.10, to decide Stability it is enough to determine if v belongs to a path in $G[0]$ with ends satisfying (1) or (2).

To determine if v belongs to a path with ends satisfying (1) or (2), we build another graph $\overline{G} = (\overline{V},\ \overline{E})$ from $G[0,\ v]$ and a new vertex called ∞, where:

$$\overline{V} = V[0,\ v] \cup \{\infty\}$$

$$\overline{E} = E[0,\ v] \cup \{(u, \infty) | u \in V[0,\ v]\}$$

satisfying (1) or (2).

Notice that v is stable if and only if there are two different paths from v to ∞. Then, v is stable if and only if there is a cycle in $\overline{G}$ that contains v and ∞. Thus, we must calculate the biconnected components of G, and then decide if v and ∞ are in the same component.

Algorithm 4 Solving Stability for bootstrap percolation rule

```
Input: A graph $G = (V, E)$ and $c$ a configuration in $\{0,1\}^V$.
 1: Calculate $A[0]$, the adjacency matrix of $G[0]$.
 2: Calculate $A[0,\ v]$, the adjacency matrix of $G[0,\ v]$, using $A[0]$.
 3: if  $v$ is isolated in $A[0,\ v]$ then
 4:     return Reject
 5: end if
 6: Calculate $B$ the vector with the bi-connected component $A[0,\ v]$.
 7: Calculate $D$, the vector of degree of $G$.
 8: Define $\overline{A}$, the adjacency matrix of $\overline{G}$, using $A[0,\ v], D$ and $C$.
 9: Calculate $\overline{B}$ the vector with the bi-connected of $\overline{A}$.
10: if  $v$ and $\infty$ are in a cycle (same bi-connected component) then
11:     return Accept
12: end if
13: return Reject
```

Let $N = |V|^2$ be the size of the input. Step **1** can be done in $\mathcal{O}(\log N)$ time with $\mathcal{O}(N^2)$ processors: $|E| = \mathcal{O}(N)$ processors per vertex verify if $\{v, u\} \in E$. Step **2** can be done in $\mathcal{O}(\log^2 N)$ time with $\mathcal{O}(N)$ processors using connected component algorithm 3.2.5. Step **3-5** can be done in $\mathcal{O}(\log N)$ time with $\mathcal{O}(N)$ processors using prefix sum algorithm 3.2.4. It is enough to compute the OR on the vector non containing $A[0,\ v]_{v,v}$, $(A[0,\ v]_{v,v_1}, ..., A[0,\ v]_{v,v_{|V|}})$. Step **6** can be done in $\mathcal{O}(\log^2 N)$ time with $\mathcal{O}(N^2/\log N)$ processors using bi-connected component algorithm 3.2.6. Step **7** can be done in $\mathcal{O}(\log N)$ time with $\mathcal{O}(N)$ processors using prefix sum algorithm 3.2.4. We use $|V|$ processors by vertex. It is enough to compute the addition on each vertex u vector non containing $A[0,\ v]_{u,u}$, $(A[0,\ v]_{u,v_1}, ..., A[0,\ v]_{u,v_{|V|}})$. Step **8** can be done in $\mathcal{O}(\log N)$ time with $\mathcal{O}(N^2)$ processors: $|\overline{E}| = \mathcal{O}(N)$ processors per vertex verify if $\{v, u\} \in \overline{E}$ as step **1**. Step **9** can be done in $\mathcal{O}(\log^2 N)$ time with $\mathcal{O}(N^2/\log N)$ processors using bi-connected component algorithm 3.2.6 as step **6**. Step **10-12** can be done in $\mathcal{O}(\log N)$ time with $\mathcal{O}(1)$ processors. It is enough to verify $\overline{B}_v = \overline{B}_\infty$.

□

Note that this case includes when $G = (V, E)$ is a torus.

4.4 Monotone freezing cellular automata

Several CA applications are freezing and monotonous, such as the SIR model [36], forest fires [57] and bootstrap percolation [34], in addition to freezing threshold CA. We are interested in the results about the fixed points and the asymptotic configurations of these.

The first property is that the fixed point is invariant for the updating schemes.

Lemma 4.4.1. *Let F be a FCA monotone, σ a updating scheme and $c \in Q^{\mathbb{Z}^d}$, then a fixed point of $F^{\sigma(\infty)}$ is a fixed point for any other updating scheme, in particular, $F^{\sigma(\infty)}(c)$ is a fixed point any other updating scheme.*

Proof. Let F be a freezing cellular automaton, σ,σ' two updating schemes and $c \in Q^{\mathbb{Z}^d}$ a fixed point for σ. By definition of fixed point $\forall t : F^{\sigma(t)}(c) = c$, then as F is a freezing cellular automaton

$$\forall z \in \mathbb{Z}^d : f(c_{N(z)}) = c_z. \tag{4.1}$$

Suppose that this is not true, then let $z \in \mathbb{Z}^d$ be such that the equation 4.1 is false and let $t_z = \min\{t : z \in \sigma(t)\}$, then $F^{\sigma(t_z)}(c)_z > c_z$, concluding that $F^{\sigma(\infty)}(c)_z > c_z$.

Finally, by equation 4.1, $\forall t : F^{\sigma'(t)}(c) = c$.

□

Definition 4.4.1. A CA F is *confluent* if given σ, σ' two different updating schemes and $c \in Q^{\mathbb{Z}^d}$, then $F^{\sigma(\infty)}(c) = F^{\sigma'(\infty)}(c)$, i.e. every iteration has the same asymptotic configuration.

The following theorem shows the interaction between the monotony and freezing.

Theorem 4.4.2. *Let F be a FCA monotone, then this is confluent.*

Proof. By monotony

$$c \leq F^{\sigma(\infty)}(c) \, \wedge \, c \leq F^{\sigma'(\infty)}(c)$$

applying $F^{\sigma'}$ to the term on the left side and F^{σ} to the term on the right side and lemma 4.4.1

$$F^{\sigma'}(c) \leq F^{\sigma'}(F^{\sigma(\infty)}(c)) = F^{\sigma(\infty)}(c) \, \wedge \, F^{\sigma}(c) \leq F^{\sigma}(F^{\sigma'(\infty)}(c)) = F^{\sigma'(\infty)}(c)$$

repeating this process t times and taking $t \to \infty$

$$F^{\sigma'(\infty)}(c) \leq F^{\sigma(\infty)}(c) \, \wedge \, F^{\sigma(\infty)}(c) \leq F^{\sigma'(\infty)}(c)$$

i.e.

$$F^{\sigma'(\infty)}(c) = F^{\sigma(\infty)}(c)$$

□

Chapter 5

Universality in Freezing Cellular Automata

In the pioneering works impulsed by J. von Neumann and S. Ulam in the 50-60s, when cellular automata were formally defined for the first time, two important themes were already present: universality [13, 58, 59] and growth dynamics [60]. Growing dynamics in cellular automata were also much studied, mostly through (classes of) examples with different points of view [61, 30, 62, 36]. More recently, substantial works have been published on models of self-assembly tilings, most of which can be seen as a particular non-deterministic 2D CA where structures grow from a seed. Interestingly, the question of intrinsic universality was particularly studied in that case [63, 64].

A common feature of all these examples is that only a bounded number of changes per cell can occur during the evolution. To our knowledge, the first time that the class of CAs with that feature was considered as a whole is in [65] with a point of view of language recognition. More recently the notion of *freezing* CA was introduced in [33] which captures essentially the same idea with an explicit order on states, and a systematic study of this class (dynamics, predictability, complexity) was started. In particular it was established that the class is Turing universal (even in dimension 1).

In this chapter, we study intrinsic universality in freezing CA as a first step to understand universality in growth dynamics in general. Our central result is the construction of such intrinsically universal freezing CA: it shows that the class of freezing CA is a natural computational model with maximally complex elements which can be thought of as machines that can be 'programmed' to produce any behavior of the class. Moreover, the universal CA that we construct are surprisingly small (5 states, see Section 5.2.1) which is in strong contrast with the complicated construction known to obtain intrinsic universality for the classical self-assembly aTAM model [63]. Our contribution also lays in the negative results we prove (Theorems 5.3.1, 5.3.3 and 5.3.5): interpreting them as necessary conditions to achieve universality for freezing CA, we obtain a clear landscape of the fundamental computational or dynamical features of this class.

The chapter is organized as follows. In Section 5.1 we define the main concepts and prove that the use of context-free simulation cannot lead to universality. Section 5.2 gives a general construction scheme to obtain universal freezing CA giving three positive results in three different settings depending on the dimension, the neighborhood and the maximum number of state changes per cell. In section 5.3, we show several obstacles to the existence of universal freezing CA: dimension 1, 1 change per cell with von Neumann neighborhood in 2D, and monotonicity. Finally in section 5.4 we characterize the set of CA that can be simulated by a freezing CA, through a notion of local decreasing energy.

The content of this chapter corresponds to publication *Universality in Freezing Cellular Automata* [2]. In this chapter we will consider that the states are decreasing to be consistent with the final part of local decreasing energy.

5.1 Classical Limitation

In the context of freezing CAs, *context-free* (also called sub-automaton) universality, is prevented by the irreversibility of any computation performed by U combined with the injectivity of the coding map, as witnessed by the following theorem.

Theorem 5.1.1 (No freezing context-free universality)**.** *Let $d \in \mathbb{N}$, there is no $F \in FCA^d$ which is context-free (called previously injective) FCA_{VN_d}-universal.*

Proof. Let $d \geq 1$ be any fixed dimension. By contradiction suppose that such an universal F_u with alphabet Q_u exists and consider for any $n > 0$ the CA F_n with states $Q_n = \{-1, \dots, -n\}$ and von Neumann neighborhood with the following rule: a cell in state q changes to state r if $r < q$ and all its neighbors are in state r, otherwise it stays in state q, formally,

$$f(c_{N(u)}) = \begin{cases} r & \text{if } (c_u > r) \wedge (\forall w \in N(u) : c_u = r) \\ c_u & \text{otherwise.} \end{cases}$$

The CA F_n is $<$-freezing. By hypothesis F_u must simulate each F_n because they are all freezing CA by definition. For each n let B_n be the block size in the injection $\phi_n : Q_n \to Q_u^{B_n}$ given by simulation of F_n by F_u (it is a context free simulation so the context C is a singleton). Since Q_n is unbounded then B_n is unbounded, so we can choose n such that B_n has at least one side which is at least two times the radius r_u of the neighborhood of F_u. Without loss of generality we suppose that the left to right side of B_n is long. Consider the configuration x of F_u made by a block $\phi_n(-1)$ at position $\vec{0} \in \mathbb{Z}^d$ surrounded by blocks $\phi_n(-2)$. Since F_u on x simulates F_n on the configuration x' made of a -1 surrounded by -2, the block at position $\vec{0}$ in x must become $\phi_n(-2)$ after some time and in particular it must change: let t_0 be the first time such that $F_u^{t_0}(x)$ does not contain the block $\phi_n(-1)$ at position $\vec{0}$, and consider any position $\vec{i} \in B_n$ such that $F_u^{t_0}(x)_{\vec{i}} \neq x_{\vec{i}}$. Note that for any $\vec{k} \notin B_n$ and any time t we have $F_u^t(x)_{\vec{k}} = x_{\vec{k}}$ because cells in state -2 don't change during the evolution of x' under F_n so the corresponding blocks in the evolution of x under F_u don't change either: indeed, if such a block becomes different from $\phi_n(-2)$ at some time it will never become again $\phi_n(-2)$ (by the freezing condition on F_u and by injectivity of ϕ_n) thus contradicting the simulation of F_n by F_u through the coding ϕ_n. Therefore it holds that $F_u^{t_0-1}(x) = x$ and necessarily $t_0 = 1$ so that $F_u(x)_{\vec{i}} \neq x_{\vec{i}}$. However, since $F_u(x)_{\vec{i}}$ depends only on the $x_{\vec{i}+\vec{z}}$ for $\|\vec{z}\|_\infty \leq r_u$ and since $m_n \geq 2r_u$, it is always possible to construct a pair of configurations y of F_u and y' of F_n satisfying the following conditions:

1. $y = \overline{\phi}(y')$ (*i.e.* y is a valid encoding of y');
2. $y'_{\vec{0}} = -1$ and $F_u(x)_{\vec{i}} = F_u(y)_{\vec{i}}$;
3. any position in y' is in state -1 or -2 and has both state -1 and state -2 in its von Neumann neighborhood.

Concretely, using symmetries we can suppose without loss of generality that $\vec{i}$ belong to the left part of the block it belongs to. Then one can choose $y'_{\vec{j}} = x'_{\vec{j}}$ for $\vec{j} \in \{(-1,0), (-1,-1), (-1,1), (0,0), (0,-1), (0,1)\}$ and complete it in a greedy way to satisfy condition 3. Such a choice guaranties that $F_u(x)_{\vec{i}} = F_u(y)_{\vec{i}}$ because for any $\vec{z}$ with $\|\vec{z}\|_\infty \leq r_u$ we have $x_{\vec{i}+\vec{z}} = y_{\vec{i}+\vec{z}}$ (by the assumption that $\vec{i}$ belongs to the left part of its block and the fact the B_n is long enough from left to right). y' is a fixed point of F_n (by condition 3) so y must be a fixed point of F_u (by the freezing condition on F_u and the injectivity of ϕ_n): this contradicts condition 2 which implies $F_u(y)_{\vec{i}} \neq y_{\vec{i}}$. □

This limitation forces us to look for a more general, but not trivial, definition, that allows us to find a freezing cellular automaton.

Definition 5.1.1 (Context-sensitive)**.** Let $T > 0$, and $B \subseteq \mathbb{Z}^d$ be a d-dimensional rectangular block, with size-vector $b \in \mathbb{Z}^d$. Let $C \subsetneq \mathbb{Z}^d$ be a finite set, with $\vec{0} \in C$. Let $F = (d, Q, N, f)$ and $G = (d, Q', N', g)$ be two d-dimensional cellular automata. F *context-sensitive simulates* G with slowdown T, block B and context C if there is a injective coding map $\phi : Q_G^C \to Q_F^B$ such that the global map $\bar{\phi} : Q_G^{\mathbb{Z}^d} \to Q_F^{\mathbb{Z}^d}$ verifies:

$$\forall c \in Q_G^{\mathbb{Z}^d} : \bar{\phi}(G(c)) = F^T(\bar{\phi}(c)).$$

where $\bar{\phi}$ is defined by: for $z \in \mathbb{Z}^d, r \in B$, $\bar{\phi}(c)_{bz+r} = \phi(c|_{z+C})_r$

When G Context-sensitive simulates F with slowdown 0, block $\{0\}$, then it is said that F is a *sub-automaton* of G, as in [24, 25].

Context-sensitive simulation can get us over this hurdle as we show below; it is akin to the notion of conjugacy in symbolic dynamics [66].

Given the theorem 5.1.1, the notion of simulation used from now on will be the Context-sensitive simulation.

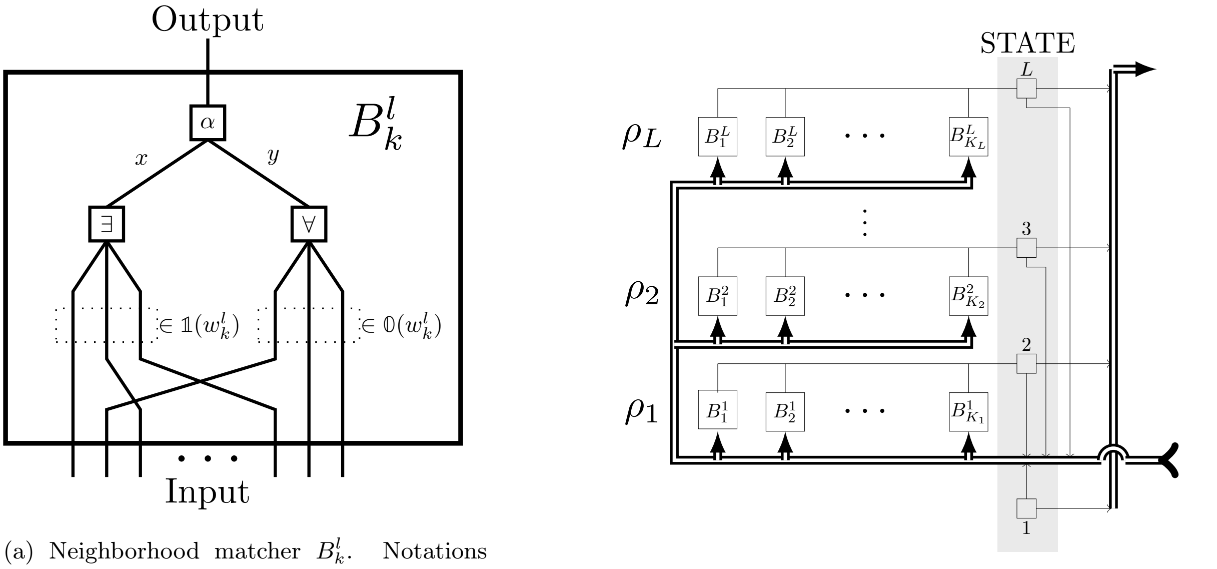
5.2 Constructing Intrinsically Universal FCA

We give a number of constructions for intrinsically universal freezing cellular automata. All of these exhibit the same running theme: if there is a environment of crossing information asynchronously, then universality can be reached. This insight yields three constructions which are concrete implementations under various technical constraints of a common abstract construction. The abstract construction can be described by: the structure of macro-cells, the mechanism to trigger state change in each macro-cell, and the wiring between neighboring macro-cells to ensure communication. At this abstract level we assume that there is a mean to cross wires without interference. Another aspect of wiring is the necessity to put delays on some wires in order to keep synchronicity of information: it is a standard aspect of circuit encoding in CAs [47, 67], which we won't address in detail here but which can be dealt with by having wires make zigzag to adjust their length as desired. The freezing condition imposes strong restrictions on the way we can code, transport and process information. We focus below on where our construction differs from the classical approach in general CAs.

Wires are Fuses. It is not possible to implement classical wires where bits of information travel freely without violating the freezing condition. In all of our constructions wires are actually fuses that can be used only once and they are usually implemented with two states: 1 stays stable without presence of neighboring 0s and 0 propagates over neighboring 1s. With that behavior our wires can be trees connecting various positions in such a way that a 0 appearing at any position is broadcasted to the whole tree. A finite wire can either be uniformly in state $b \in \{0, 1\}$ in which case all leaves 'agree' on the bit of information transported by it, or not uniform in which case information is incoherent between leaves. As it will become clear later, our constructions will use wires between adjacent blocks (or macro-cells) in the simulator CA and our encodings require that those wires are in a coherent state (uniformly $b \in \{0, 1\}$): it is precisely in this aspect that we use the power of context sensitive simulations, because the content of a block (or macro-cell) cannot be fixed independently of its neighbors in that case.

State Codification. In each macro-cell we must code in some way a (possibly very big) state that can change a (possibly very big) number of times: a classical binary encoding would violate the freezing condition so we actually use a unary coding. Given a finite set S and a quasi order $(Q, \preceq)$, let $q_0 \preceq \ldots \preceq q_{|Q|-1}$ be a linearization of $\preceq$, and let $\iota(q_i) = i$. Then let $Q_u = 1^*0^+ \cap \{0, 1\}^{|Q|}$, and $\phi \in Q \to Q_u : q \mapsto 1^{\iota(q)}0^{|Q|-\iota(q)}$. Note that for any $i < |Q|$ we have $q \preceq q' \Leftrightarrow \phi(q)_i \leq \phi(q')_i$ (where $\leq$ is the lexicographic order). Since ϕ is a bijection, for any cellular automaton F with state set Q, $\bar{\phi} \circ F \circ \bar{\phi}^{-1}$ is a cellular automaton isomorphic to F, with state set Q_u, which we call the *unary representation* of F. If F is $\preceq$-freezing, then its unary representation is $\leq$-freezing. We will use this unary encoding everywhere in the structure of our macro-cells: each state of a simulated CA F will be represented by a collection of wires representing the bits of an element of Q_u defined above. This encoding is coherent with the freezing property of the simulated CA because the fact that states can only decrease corresponds to the fact that the number of wires uniformly equal to 0 increases.

Neighborhood Matchers. The fundamental basic block of our construction is a circuit that detects a fixed pattern in the neighborhood and outputs a bit of information saying: "given this particular neighborhood pattern w, the new state of the macro-cell must be smaller than l". Our unary encoding is adapted for this because the predicate "smaller than l" for a state translates into a condition on a single bit of an element of Q_u, that is to say a single wire in our concrete representation of states. Without loss of generality we assume that $F = (\mathbb{Z}^2, Q_u, f, N)$ is a FCA with state in unary representation. Take $L = |Q_u|$, and $m = |N|$. For $l \in Q_u$, let $\{w_1^l, ..., w_{K_l}^l\} = \bigcup_{s' \leq l} f^{-1}(s') = \{n \in (Q_u)^N | f(n) \leq l\}$. Take, for some state l, $w_k^l = (q_1, ..., q_m) \in \bigcup_{s' \leq l} f^{-1}(s')$ a fixed neighborhood with output smaller than l; each state q_i is in $\{0,1\}^L$, so w_k^l is a binary word in $\{0,1\}^{mL}$.



(a) Neighborhood matcher B_k^l. Notations $\mathbb{0}(w)$ and $\mathbb{1}(w)$ stand for the set of all indexes i s.t. $w_i = 0$ and $w_i = 1$ respectively. This block triggers a 0 on the output wire exactly when the input is w_k^l.

(b) Construction of a macro-cell. Single line represent wires transporting one bit and double lines represent multi-bit wires (representing a state).

Figure 5.1: Recognizing one neighborhood (left), and wiring these *neighborhood matchers* into a *macro-cell* which computes the local function of F (right).

Given l and k, the logic gates diagram of Figure 5.1a, the *Neighborhood Matcher*, called B_k^l, outputs 0 if and only if in the input in the wires is exactly w_k^l or the output cable was already in state 0. The i-th wire joins the i-th letter in w with either the $\exists$ gate on the left if the i-th letter of w_k^l is 1 or the $\forall$ gate on the right if the i-th letter of w_k^l is 0. Gate $\exists$ triggers a 0 on wire x if at least on of its incoming wire is 0, while gate $\forall$ triggers a 0 on wire y if all incoming wires are 0. Note that both behaviors are compatible with the freezing conditions since the set of wires in state 0 can only grow during evolution. The gate α at the top triggers a 0 on the output wire if wire y is in state 0 and wire x is in state 1 (see Figure 5.1a). It is also a freezing gate, meaning that once it has triggered a 0 it will never change its state again, even if the wire x turns to state 0. Moreover this gate also turns into "trigger" state as soon as the output wire is 0.

Local Function Computation. Now we can compute the local function of F through a *macro-cell* $\mathbb{C}_F$, receiving the states $x = (x_n)_{n \in N}$ of the neighborhood as input, and yielding the next state $f(x)$ as output. For this we will divide the space into rows ρ_l for $l \in Q_u$, and some number of columns. Intuitively, the role of row ρ_l is to maintain the information "the current state of the macro-cell is less than l". For a given $l \in Q_u$, ρ_l contains all block B_k^l for $k \in \{1, \ldots, K_l\}$. The inputs are distributed to each block, and the outputs of all blocks in ρ_l are connected together by a broadcast wire. Thus, the final output in ρ_l is 0 as soon as one block K_k^l triggers, *i.e.* as soon as $f(x) \leq l$, see Figure 5.1b. Notice that once a neighborhood matcher B_k^l in row l has output 0, the output of the macro-cell it belongs to must be less than l for ever: indeed, at the time when the B_k^l was triggered to output 0 the output value of the Macro-Cell must be less than l by definition of B_k^l, after that time the output is always less than l thanks to the freezing condition on the CA being simulated. Concatenating rows in the right order, we obtain as output of the gate the correct state codification $f(x)$ for any state of the neighborhood x received as input.

Information exchanging. Given these basic blocks, one needs to embed one *macro-cell* per simulated cell on the simulator CA, and wire the inputs and outputs of neighboring macro-cells, as in Figure 5.2. The wiring between *macro-cells* depends on the neighborhood of the simulated CA. In order to clarify the presentation we will always assume that the simulated CA has a von Neumann neighborhood which is enough to achieve universality thanks to the following lemma.

Lemma 5.2.1. *For any dimension d and any $F \in FCA^d$ there is $G \in FCA^d$ with von Neumann neighborhood that simulates F.*

Proof. Consider a FCA F with state set Q, freezing order $\prec$, neighborhood $N = \{\vec{n}_1, \ldots, \vec{n}_k\} \subseteq \mathbb{Z}^d$ and local transition map $\delta : Q^N \to Q$. For a suitable choice of m and for each i ($1 \leq i \leq k$) consider a von Neumann connected path $P_i = (\vec{p}_{i,1}, \ldots, \vec{p}_{i,m})$ of length m linking $\vec{n}_i$ to $\vec{0}$: $\vec{p}_{i,1} = n_i$ and $\vec{p}_{i,m} = \vec{0}$ and $\vec{\Delta}_{i,j} = \vec{p}_{i,j} - \vec{p}_{i,j+1} \in \mathrm{VN}_d$ for $1 \leq j < m$. Now define a FCA G with von Neumann neighborhood and state set made of $mk+1$ copies of Q, denoted by projections $\pi_0, \pi_{1,1}, \ldots, \pi_{k,m}$ from $Q^{mk+1} \to Q$, and with the following behavior at each step:

- $\pi_0(G(c)_{\vec{z}}) = \delta\big(\pi_{1,m}(c_{\vec{z}}), \ldots, \pi_{k,m}(c_{\vec{z}})\big)$ if $\delta\big(\pi_{1,m}(c_{\vec{z}}), \ldots, \pi_{k,m}(c_{\vec{z}})\big) \prec \pi_0(c_{\vec{z}})$ and $\pi_0(c_{\vec{z}})$ otherwise,
- $\pi_{i,j+1}(G(c)_{\vec{z}}) = \pi_{i,j}(c_{\vec{z}+\vec{\Delta}_{i,j}})$ if $\pi_{i,j}(c_{\vec{z}+\vec{\Delta}_{i,j}}) \prec \pi_{i,j+1}(c_{\vec{z}})$ and $\pi_{i,j+1}(c_{\vec{z}})$ otherwise, for $1 \leq j < m$ and $1 \leq i \leq k$,
- $\pi_{i,1}(G(c)_{\vec{z}}) = \pi_0(c_{\vec{z}})$ if $\pi_0(c_{\vec{z}}) \prec \pi_{i,1}(c_{\vec{z}})$ and $\pi_{i,1}(c_{\vec{z}})$ otherwise, for $1 \leq i \leq k$.

Intuitively, G realizes in parallel the propagation of neighboring Q-states along paths of the form $z + P_i$ from any cell $\vec{z}$ and the application of the local transition δ in each cell using the Q-components corresponding to the end of each propagation path P_i. Let's show that G is a freezing CA that simulates F. First, it is clear that G is freezing because in any transition, any component of the state can only decrease according to $\prec$. Now consider the encoding map $\phi : Q^{\mathbb{Z}^d} \to \big(Q^{km+1}\big)^{\mathbb{Z}^d}$ defined by:

- $\pi_0(\phi(c)_{\vec{z}}) = \pi_{i,1}(\phi(c)_{\vec{z}}) = c_{\vec{z}}$ for $1 \leq i \leq k$,
- $\pi_{i,j+1}(\phi(c)_{\vec{z}}) = \pi_{i,j}(\phi(c)_{\vec{z}+\vec{\Delta}_{i,j}})$ for $1 \leq j < m$ and $1 \leq i \leq k$.

A configuration $\phi(c)$ is such that the Q-value is constant along P_i paths so only the π_0 components can change when applying G and we necessarily have $\pi_0(G(\phi(c))_{\vec{z}}) = F(c)_{\vec{z}}$ for $1 \leq i \leq k$, because F is freezing for the order $\prec$ and $\pi_{i,m}(\phi(c)_{\vec{z}}) = c_{\vec{z}+\vec{n}_i}$ by the second item above and the definition of paths P_i. Then, in $G(\phi(c))$, only the $\pi_{i,1}$ components can change and it holds that

$$\pi_{i,1}(G^2(\phi(c))_{\vec{z}}) = \pi_0(G(\phi(c))_{\vec{z}}) = F(c)_{\vec{z}}$$

for $1 \leq i \leq k$. Similarly it is straightforward to check that after $m+1$ steps the two item of the definition of ϕ are again verified and we have: $G^{m+1}(\phi(c)) = \phi(F(c))$. This shows that G simulates F and the lemma follows.

□

The von Neumann wiring between *macro-cells* in dimension 2 is shown on Figure 5.2. It is straightforward to generalize it to any dimension. Technically, thanks to Lemma 5.2.1, all the encoding map ϕ we use later have a von Neumann neighborhood context (C in Definition 5.1.1).

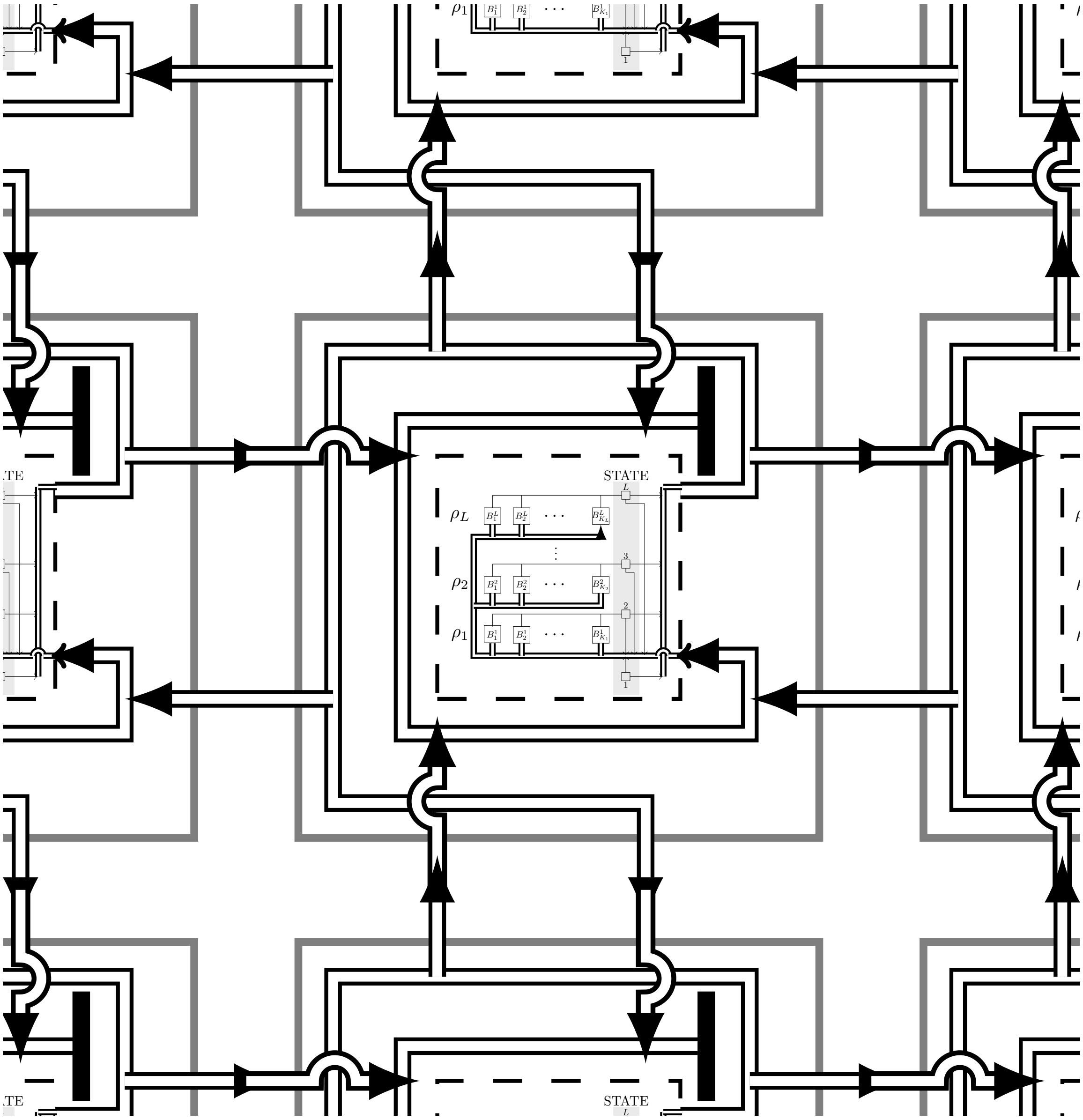


Figure 5.2: The dashed block in the middle is a macro-cell, as in figure 5.1b. The fat arrows exchange the state of each macro-cell with its neighbors.

Context-sensitive encoding. Given a configuration c of the simulated CA, the encoding is defined as follows. All wires of the construction are in a coherent state (same state along the wire). Each macro-cell holds a state of the configuration c represented in unary by the rows ρ_l described above. Wires incoming from neighboring macro-cells hold the information about neighboring states (hence the context-free encoding) which is transmitted to each block B^l_k. Inside each of these blocks the inputs arrive at gates "$\exists$" and "$\forall$" and these gates have eventually triggered a 0 on wires x and y. However, gate "α" has not yet triggered to preserve the property that the main wire of row ρ_l is coherent and represents the information on the *current* state of the macro-cell. Starting from that well encoded configuration, a simulation cycle begins by the possible triggering of "α" gates. After some time a well encoded configuration is reached again because changes coming from triggerings of α gates are broadcasted on each row, and, in each block

B^l_k, the content up to the α gate is determined by the inputs.

We can now state three variants of the construction which differ essentially in the way crossing of information is implemented.

5.2.1 A 2D, 2-change, von Neumann Neighborhood Intrinsically Universal FCA

Theorem 5.2.2. *$\exists U \in FCA_{\mathrm{VN}_2}$ with 5 states which is 2-change and FCA^2-universal.*

We can make a direct implementation of the abstract construction as a universal FCA is $U = \{\mathbb{Z}^2, \{$□, ■, 🟥, $\boxed{\leftrightarrow}$, $\boxed{\updownarrow}$, $\boxed{\alpha}$, $\colorbox{red}{$\alpha$}$, $\boxed{\exists}$, $\colorbox{red}{$\exists$}$, $\boxed{\forall}$, $\colorbox{red}{$\forall$}$$\}, VN, f_u\}$. This is a 2-change FCA with freezing order:

□, ■, $\boxed{\alpha}$, $\boxed{\exists}$, $\boxed{\forall}$ $\geq$ $\boxed{\leftrightarrow}$, $\boxed{\updownarrow}$ $\geq$ 🟥, $\colorbox{red}{$\alpha$}$, $\colorbox{red}{$\exists$}$, $\colorbox{red}{$\forall$}$.

It implements all elements of Section 5.2 and the states have the following meaning:

- □ is the quiescent background,
- ■ is a wire waiting for a signal,
- 🟥 is a signal,
- $\boxed{\leftrightarrow}$ (resp. $\boxed{\updownarrow}$) an intermediate states to manage a crossing when a first signal already passed horizontally (resp. vertically),
- $\boxed{\alpha}$ (resp. $\boxed{\exists}$ and $\boxed{\forall}$) is the α (resp. $\exists$ and $\forall$) gate waiting the conditions to trigger,
- $\colorbox{red}{$\alpha$}$ (resp. $\colorbox{red}{$\exists$}$ and $\colorbox{red}{$\forall$}$) is the α (resp. $\exists$ and $\forall$) gate once it has triggered.

All wires described by the abstract construction are made by drawing trees of degree at most 3 of VN-connected cells in state ■. The case of ■ with 4 neighbors in state ■ is reserved to manage crossings. Also gates $\exists$ and $\forall$ have unbounded fan-in in the abstract construction. Here we simulate unbounded fan-in by fan-in 2 (which is possible because the semantics of these gates is associative). More precisely, all gates ($\boxed{\alpha}$,$\boxed{\exists}$ and $\boxed{\forall}$) receive there first (resp. second) input from their left (resp. right) neighbor and send their output to the top.

The local rule is given by the following set of transitions, any cell which is in a local context not appearing in this list stays unchanged:

Normal wires: if □ $\in \{n, e, s, w\}$

- $\begin{array}{ccc} & n & \\ w & \blacksquare & e \\ & s & \end{array} \mapsto$ 🟥 if 🟥 $\in \{n, e, s, w\}$
- $\begin{array}{ccc} & n & \\ w & \blacksquare & e \\ & s & \end{array} \mapsto$ 🟥 if $\boxed{\leftrightarrow} \in \{e, w\}$ or $\boxed{\updownarrow} \in \{n, s\}$

Crossings: if $\{n, e, s, w\} \subseteq \{$■, 🟥$\}$

- $\begin{array}{ccc} & n & \\ w & \blacksquare & e \\ & s & \end{array} \mapsto \boxed{\leftrightarrow}$ if 🟥 $\in \{e, w\}$ and 🟥 $\notin \{n, s\}$

- $\begin{array}{ccc} & n & \\ w & \blacksquare & e \\ & s & \end{array} \mapsto \boxed{\updownarrow}$ if $\textcolor{red}{\blacksquare} \notin \{e, w\}$ and $\textcolor{red}{\blacksquare} \in \{n, s\}$

- $\begin{array}{ccc} & n & \\ w & \blacksquare & e \\ & s & \end{array} \mapsto \textcolor{red}{\blacksquare}$ if $\textcolor{red}{\blacksquare} \in \{e, w\}$ and $\textcolor{red}{\blacksquare} \in \{n, s\}$

- $\begin{array}{ccc} & n & \\ w & \boxed{\leftrightarrow} & e \\ & s & \end{array} \mapsto \textcolor{red}{\blacksquare}$ if $\textcolor{red}{\blacksquare} \in \{n, s\}$

- $\begin{array}{ccc} & n & \\ w & \boxed{\updownarrow} & e \\ & s & \end{array} \mapsto \textcolor{red}{\blacksquare}$ if $\textcolor{red}{\blacksquare} \in \{e, w\}$

Gates triggering: if $\{e, w\} \subseteq \{\blacksquare, \textcolor{red}{\blacksquare}\}$

- $\begin{array}{ccc} & n & \\ \textcolor{red}{\blacksquare} & \boxed{\alpha} & \blacksquare \\ & s & \end{array} \mapsto \colorbox{red}{α}$,

- $\begin{array}{ccc} & n & \\ w & \boxed{\exists} & e \\ & s & \end{array} \mapsto \colorbox{red}{$\exists$}$ if $\textcolor{red}{\blacksquare} \in \{e, w\}$,

- $\begin{array}{ccc} & n & \\ \textcolor{red}{\blacksquare} & \boxed{\forall} & \textcolor{red}{\blacksquare} \\ & s & \end{array} \mapsto \colorbox{red}{$\forall$}$

Gates output:

- $\begin{array}{ccc} & n & \\ w & \blacksquare & e \\ & s & \end{array} \mapsto \textcolor{red}{\blacksquare}$ if $s \in \{\colorbox{red}{$\alpha$}, \colorbox{red}{$\exists$}, \colorbox{red}{$\forall$}\}$

It appears that the south state in the gate triggering transitions above is not used. Moreover no transition involves the background state $\square$ and the behavior of $\boxed{\updownarrow}$ is similar to that of gate output transitions. This allows us to reduce the state set to $\{\square, \blacksquare, \textcolor{red}{\blacksquare}, \boxed{\leftrightarrow}, \boxed{\updownarrow}\}$ and to code all triggered gates by $\boxed{\updownarrow}$ and (untriggered) gates $\boxed{\alpha}$ (resp. $\boxed{\exists}$ and $\boxed{\forall}$) by a $\square$ state having at south a $\square$ (resp. $\boxed{\leftrightarrow}$ and $\boxed{\updownarrow}$). More precisely, we keep all transitions for normal wires and crossings and add the following ones which replace the gates triggering: if $\{n, e, w\} \subseteq \{\blacksquare, \textcolor{red}{\blacksquare}\}$

- $\begin{array}{ccc} & n & \\ \textcolor{red}{\blacksquare} & \square & \blacksquare \\ & \square & \end{array} \mapsto \boxed{\updownarrow}$,

- $\begin{array}{ccc} & n & \\ w & \square & e \\ & \boxed{\leftrightarrow} & \end{array} \mapsto \boxed{\updownarrow}$ if $\textcolor{red}{\blacksquare} \in \{e, w\}$,

- $\begin{array}{ccc} & n & \\ \textcolor{red}{\blacksquare} & \square & \textcolor{red}{\blacksquare} \\ & \boxed{\updownarrow} & \end{array} \mapsto \boxed{\updownarrow}$.

The gates output transitions are already realized by the behavior of $\boxed{\updownarrow}$ on normal wires. Finally, it is important to note that the crossing transition that transforms a $\boxed{\updownarrow}$ into $\textcolor{red}{\blacksquare}$ will not interfere here because it applies only when all states surrounding $\boxed{\updownarrow}$ belongs to $\{\blacksquare, \textcolor{red}{\blacksquare}\}$ and in the 3 transitions above to simulate gates, the $\boxed{\updownarrow}$ state generated will have a state among $\{\square, \boxed{\leftrightarrow}, \boxed{\updownarrow}\}$ as south neighbor. We conclude that 5 states are enough to achieve universality with von Neumann neighborhood in 2D.

5.2.2 A 3D, von Neumann neighborhood, 1-change, intrinsically universal FCA

Theorem 5.2.3. *$\exists U \in FCA_{\mathrm{VN}_3}$ with 2 states which is 1-change and FCA^3-universal.*

The abstract construction works exactly the same way in 3D, the only difference being that there are more neighbors in the 3D version of von Neumann neighborhood. Moreover, the 2D CA constructed in Section 5.2.1 can be used almost as is to obtain a 3D FCA-universal example. Indeed, all the logic circuitry of macro cell can be done in a planar way and the third dimension matters only in the wiring between 3D macro-cells.

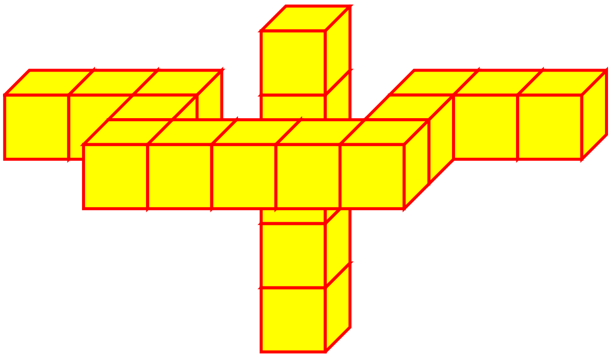


Figure 5.3: Crossing signal in 3-D.

To do so, it is sufficient to add the 3-dimensional equivalent of the normal wires transition of Section 5.2.1. Moreover, we can build a 3D FCA-universal CA which is only 1-change. This is possible by substituting all the crossings mechanics used in Section 5.2.1 for the crossing given in the Figure 5.3. Note that crossing transitions are the only place where the intermediate states $\boxed{\leftrightarrow}$ and $\boxed{\updownarrow}$ can disappear. Therefore, when removing those transitions, we get a 1-change FCA with freezing order:

□, ■ $\geq$ $\boxed{\leftrightarrow}$, $\boxed{\updownarrow}$, 🟥.

At this point states $\boxed{\leftrightarrow}$ and $\boxed{\updownarrow}$ are totally unrelated to crossings and are just used to code the behavior in the gates triggering transitions and the propagation of a 🟥 at triggered gate outputs. However a cell in 3D has 6 von Neumann neighbors, therefore there is room to code all the different behaviors using less states.

In fact 2 states are enough and FCA-universality is achieved by the 3D von Neumann FCA on $\{0, 1\}$ given by:

- a 1 surrounded by exactly two 0s becomes 0;
- 0s stay unchanged.

The 2D version of this FCA was studied in section 6.3.3 were it was shown that it can implement all necessary synchronous logical gates. By using such planar constructions in the 3D version and using the third dimension to implement asynchronous crossings as above, we can realize the abstract construction of Section 5.2.

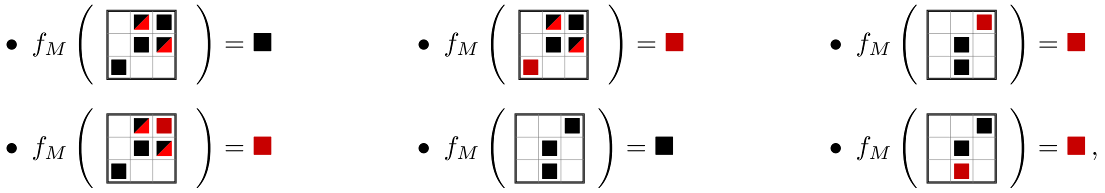
5.2.3 2D, Moore Neighborhood, 1-change, intrinsically universal FCA

Theorem 5.2.4. *$\exists U \in FCA_{\mathrm{MN}_2}$ with 4 states which is 1-change and FCA^2-universal.*

We build an intrinsically universal FCA U_M with Moore Neighborhood.

$$U_M = \mathcal{CA}(\mathbb{Z}^2, \{\square, \blacksquare, {\color{red}\blacksquare}, \boxed{\updownarrow}, \boxed{\leftrightarrow}\}, \mathrm{MN}, f_M\})\,.$$

The local function f_M is an extension of the CA defined in Section 5.2.1, adding rules using the Moore neighborhood to build a crossing as follows:

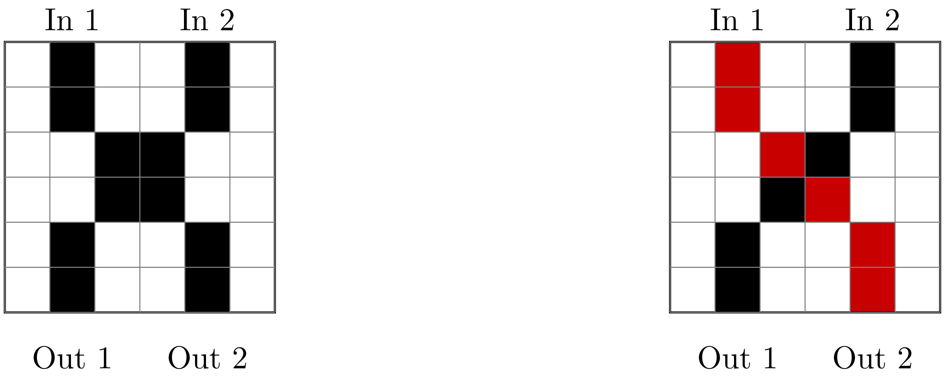


and the rotations and reflexions of these transitions, where ◪ match both ■ and 🟥 , and

$$f_M\left(\begin{array}{|c|c|c|}\hline \alpha & a & \beta \\ \hline e & c & b \\ \hline \gamma & d & \delta \\ \hline \end{array}\right) = f_u\left(\begin{array}{ccc} & \boxed{a} & \\ \boxed{e} & \boxed{c} & \boxed{b} \\ & \boxed{d} & \end{array}\right) \text{ if } \begin{array}{|c|c|c|}\hline \alpha & a & \beta \\ \hline e & c & b \\ \hline \gamma & d & \delta \\ \hline \end{array} \text{ is not in the previous cases}$$

where f_u is the 5-states CA defined in Section 5.2.1 without the crossings transition. It is therefore a 1-change CA with freezing order $\square, \blacksquare \geq {\color{red}\blacksquare}, \boxed{\leftrightarrow}, \boxed{\updownarrow}$.

With this local function the realization of crossings is given by Figure 5.4.



(a) Crossing gadget.

(b) Crossing gadget with a signal.

Figure 5.4: Crossing signal in Moore neighborhood. If the cell in In 1 (2) is in state 🟥, then after five iterations the cell in Out 2 (1) change to state 🟥 (as seen on Figure 5.4b).

As in previous constructions there is room for optimization of the number of states. For instance, one can remove the $\boxed{\leftrightarrow}$ state used in f_u to encode a type of gate. Instead we can use the space allowed by the Moore neighborhood to encode the gate by using different patterns of □ and $\boxed{\updownarrow}$ in the bottom row of the neighborhood. This give a FCA-universal example with only 4 states.

A natural candidate with only 2 states in this setting is “life without death”. It is quite possible that it is FCA-universal. However the circuitry built in [30] to show the P-completeness of this CA cannot be used directly here, in particular it does not yield an implementation of asynchronous crossing. We leave the question of FCA-universality of “life without death” open.

5.3 Obstacles to FCA-Universality

The one-dimensional case. Although one-dimensional freezing CA can be computationally universal [33], they cannot be FCA^1-universal. This is a major difference with CA in general. The intuition behind this limitation is the following: in any given 1D freezing CA, there is a bound on the number of times a zone of consecutive cells can be crossed by a signal; and above this bound, the zone becomes a blocking word preventing any information flow between left and right halves around it.

Theorem 5.3.1 (Dimension 1)**.** *There is no $F \in FCA^1$ which is FCA_{VN_1}-universal, even with context-sensitive simulation.*

Proof. Suppose that F_u is a freezing 1D CA with radius r and alphabet Q_u that can simulate any freezing 1D CA. There is a constant $M \leq |Q_u|^r$ such that the global state of a group of r consecutive cells of F_u cannot change more than M times. Consider now the CA F with states $Q_M = \{0, \ldots, M+1\}$ defined by: $F(c)_i = \min(c_i, c_{i+1})$ for any configuration c and any $i \in \mathbb{Z}$. F is a freezing CA for the natural order on integers so by hypothesis F_u simulates F using (context-sensitive) encoding map $\overline{\phi} : Q_M^{\mathbb{Z}} \to Q_u^{\mathbb{Z}}$ defined from a local map ϕ as in Definition 5.1.1. If a configuration c of F is such that

$$c_i = \begin{cases} q & \text{if } -k \leq i \leq k \\ \geq q & \text{if } i < k \\ < q & \text{if } i > k \end{cases}$$

for k larger than the radius of map ϕ, then there is t such that $\overline{\phi}(c)_{[0,r-1]} \neq F_u^t(\overline{\phi}(c))_{[0,r-1]}$: indeed if $F_u^t(\overline{\phi}(c))_{[0,r-1]}$ stays constant with t, then, considering the configuration d such that $c_i = d_i$ for $i \leq k$ and $d_i = q$ for $i > k$, we also have that $F_u^t(\overline{\phi}(d))_{[0,r-1]}$ is constant (because $F^t(d)_i$ is constant for any $i \geq -k$ and k is larger than the radius of ϕ). Moreover $\overline{\phi}(d)_i = \overline{\phi}(c)_i$ for $i < r$, therefore we should have $F_u^t(\overline{\phi}(c))_i = F_u^t(\overline{\phi}(d))_i$ for all $i < r$ and all t which would contradict the simulation of F by F_u since $F^t(c)_i$ and $F^t(d)_i$ differ for some t and $i < 0$.

Consider the following configuration of F:

$$d = {}^{\infty}(M+1) \cdot M^{k_1}(M-1)^{k_2}3^{k_3} \cdots 1^{k_M}0^{\infty}$$

where the leftmost occurrence of state M is at position $i = k_0$ for a choice of large enough $k_0, \ldots, k_M$. By the reasoning above, we must have that $\left(F_u^t(\phi(d))_{[0,r-1]}\right)_t$ must change $M+1$ times but this contradicts the definition of M. □

2D von Neumann 1-change FCA: information crossing. We will show that there is no freezing universal FCA which is 1-change and has the von Neumann neighborhood. This result is to be contrasted with the case of self-assembly tiling where an intrinsically universal system exists [63] (although with an unavoidably more technical definition of simulation). The intuition is that the propagation of state changes in such FCA produces 4-connected paths that can not be crossed in the future of the evolution because only 1 state change per cell is possible. As shown in the construction of Theorem 5.2.2, two changes per cell are enough to get rid of this limitation, even with the von Neumann neighborhood.

We will show that no 1-change von Neumann FCA can simulate the following 2-change FCA $(\mathbb{Z}^2, Q_F, \mathrm{LN}, f)$, with $Q_F = \{0, \leftarrow, \downarrow, \rlap{$\leftarrow$}\downarrow\}$ and where f is defined by

$$f(0, \downarrow, 0) = \downarrow \qquad f(0, 0, \leftarrow) = \leftarrow \qquad f(0, *, \rlap{$\leftarrow$}\downarrow) = \leftarrow \qquad f(\downarrow, *, \leftarrow) = \rlap{$\leftarrow$}\downarrow$$

and $f(a, *, *) = a$ else, where $*$ stands for any state and the arguments of f correspond to neighborhood LN in the following order: center, north, east.

Given a FCA F with von Neumann neighborhood, we call *changing path* from z to z' between configurations c and $F^t(c)$ a path $z_1, \ldots, z_n$ such that:

- $z_1 = z$,
- $z_n = z'$,
- $z_{i+1} \in \mathrm{VN}(z_i),\ \forall i = 1, \ldots, n$,
- $c_{z_i} \neq F^t(c)_{z_i},\ \forall i = 1, \ldots, n$.

We also say that a position z is *stable* in a configuration c if $F(c)_z = c_z$ and *unstable* if it is not stable.

Lemma 5.3.2 (Changing path lemma)**.** *Let z be a position in some configuration c and let $t \geq 1$ be such that $F^t(c)_z \neq c_z$, then there exists an unstable position z' in c and a changing path of length at most t from z to z' between configurations c and $F^t(c)$.*

Proof. Consider the first time t' such that $F^{t'}(c)_z \neq c_z$. If $t' = 1$ we are done because in this case z itself is unstable in c. Otherwise, z is stable in c and therefore one of its neighbors must have changed before time t'. Therefore we can apply inductively the lemma on this neighbor with a time $t < t' \leq t$ to get a changing path of length at most $t - 1$ from an unstable position to this neighbor, which we complete into a changing path of length at most t from an unstable position to z. □

Theorem 5.3.3. *There is no automaton in FCA_{VN_2} which is 1-change and able to simulate F. Therefore there is no automaton in FCA_{VN_2} which is 1-change and FCA_{VN_2}-universal.*

Proof. Suppose by contradiction that such a 1-change FCA $U = (\mathbb{Z}^2, Q, VN, f_s)$ exists. Let $\phi : Q_F^C \to Q^B$ be the encoding map ensuring the simulation of F by U in T steps:

$$\overline{\phi}(F(c)) = U^T(\overline{\phi}(c)),$$

with $\overline{\phi}(c)_{\vec{z}}$ depending exactly on cells $\lfloor \vec{z}/b \rfloor + C$ of c, where $b \in \mathbb{Z}^2$ is the size-vector of B. The injectivity of $\overline{\phi}$, the simulation and Lemma 5.3.2 ensure that there is a finite set $E \subseteq \mathbb{Z}^2$ (depending on C, b and T) such that for any configuration $c \in Q_F^{\mathbb{Z}^2}$:

- if some position z is unstable in c then some position $z' \in bz + E$ is unstable in $\overline{\phi}(c)$;
- if all positions in $z + C$ are stable in c then all positions in $bz + B$ are stable in $\overline{\phi}(c)$.

Let us consider configuration $c^n \in Q_F^{\mathbb{Z}^2}$ for any $n \geq 0$ defined by:

$$c^n(z) = \begin{cases} \downarrow & \text{if } z = (0, n), \\ \leftarrow & \text{if } z = (n^2, 0), \\ 0 & \text{else.} \end{cases}$$

By definition of F, for any t, $F^t(c^n)$ contains exactly two unstable positions: $(0, n - t)$ and $(n^2 - t, 0)$ ($\downarrow$ propagates downward, $\leftarrow$ propagates to the left eventually crossing a $\downarrow$ and everything else stays unchanged). Using the observations above and Lemma 5.3.2 we deduce that for any large enough n there exist a changing path $P_n = (z_1, \ldots, z_m)$ of length $\Omega(n)$ from $z_1 \in b \cdot (0, n) + E$ to $z_n \in b \cdot (0, -n) + E$ between configurations $\overline{\phi}(c^n)$ and $U^{2nT}(\overline{\phi}(c^n))$. Moreover, choosing n large enough, each position of the path P_n is at distance at most K from the vertical axis where K is a constant given by the simulation that do not depend on n: indeed changes between c^n and $F^{2n}(c^n)$ occur on the vertical axis or at distance at least $n^2 - 2n$ from it. Then, for a suitable choice of a $t_0 \in o(n)$ we have that:

- there is an unstable position $z = (x, y)$ in $U^{(n^2 - t_0)T}(\overline{\phi}(c^n))$ with $y > K$ and at distance $o(n)$ from the center $(0, 0)$, while all other unstable positions are at distance $\Omega(n)$ from the center;
- there is a position $z' = (x', y')$ with $y' < K$ and at distance $o(n)$ from the center such that $U^{(n^2 + t_0)T}(\overline{\phi}(c^n))_{z'} \neq \overline{\phi}(c^n)_{z'}$.

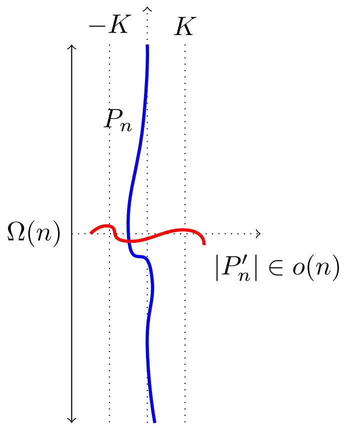


Figure 5.5: Changing paths P_n and P'_n that must cross each other.

Using Lemma 5.3.2 again, we deduce the existence of a changing path P'_n of length $o(n)$ from z to z' between configurations $U^{(n^2-t_0)T}(\overline{\phi}(c^n))$ and $U^{(n^2+t_0)T}(\overline{\phi}(c^n))$. Given the respective length and endpoints of P_n and P'_n (see Figure 5.5), they must necessarily cross each other: this is a contradiction because all positions of P_n have already made a change under the action of U after time $2nT$, so none of them can change later since U is 1-change. □

Monotone FCA: synchronous vs. asynchronous information. As for classical real function, we can consider the property of *monotonicity* in CA: given two configurations, one smaller that the other, their images by the CA compare in the same order. We are particularly interested in the case where the order on configurations is given by the order on states of a freezing CA. Several examples of such monotone FCAs were studied in literature. In particular, a simple model called *bootstrap percolation* was proposed by Chalupa in 1979 [34] to understand the properties in some magnetic materials. This model and several variants were studied from the point of view of percolation theory [61, 62], but also from the point of view of complexity of prediction [39].

The intuitive limitation of monotone freezing CAs is that they must always produce a smaller state when two signals arrive simultaneously at some cell compared to when one of the two signals arrives before the other. We now exhibit a freezing non monotone CA F that does precisely the opposite (non-simultaneous arrival produces a smaller state). Next theorem shows that F cannot be simulated by any freezing monotone CA. F is defined by $(\mathbb{Z}^2, \{0, s_1, s_2, w, \triangle\}, LN, f)$ with f given by:

- $f\left(\begin{array}{|c|c|}\cline{1-1} s_i & \multicolumn{1}{c}{} \\ \hline w & 0 \\ \hline\end{array}\right) = s_i$
- $f\left(\begin{array}{|c|c|}\cline{1-1} s_2 & \multicolumn{1}{c}{} \\ \hline s_1 & 0 \\ \hline\end{array}\right) = s_2$
- $f\left(\begin{array}{|c|c|}\cline{1-1} s_1 & \multicolumn{1}{c}{} \\ \hline \triangle & s_1 \\ \hline\end{array}\right) = s_1$
- $f\left(\begin{array}{|c|c|}\cline{1-1} 0 & \multicolumn{1}{c}{} \\ \hline w & s_i \\ \hline\end{array}\right) = s_i$
- $f\left(\begin{array}{|c|c|}\cline{1-1} s_2 & \multicolumn{1}{c}{} \\ \hline s_1 & w \\ \hline\end{array}\right) = s_2$
- $f\left(\begin{array}{|c|c|}\cline{1-1} s_1 & \multicolumn{1}{c}{} \\ \hline \triangle & w \\ \hline\end{array}\right) = s_2,$

and unspecified transitions let the state unchanged. F is $\prec$-freezing for the following order on states: $s_2 \prec s_1 \prec 0, \Delta, w$. Essentially F is a FCA sending the signals s_1 and s_2 towards south or west along the wires materialized by state w. s_2 can also move on wires made of s_1. Δ plays the role of a non-monotone local gate: when two signals arrive simultaneously, a s_1-signal is sent to the south, but when only one signal arrives, a s_2-signal is sent to the south. F cannot be simulated by any monotone FCA, hence no monotone FCA can be FCA-universal.

The key to understand the limitations of monotone FCAs is to establish that some configurations is 'above' another one for the freezing order, and then use the fact that this relation is preserved under iterations. The next lemma gives a tool to obtain such relations in the context of a simulation between FCAs.

Given a CA F and a finite set $E \subseteq \mathbb{Z}^d$, we say that a configuration y is *E-locally reachable* from a configuration x if for any $\vec{i} \in \mathbb{Z}^d$ there are configurations x^i and y^i with:

- $x^{\vec{i}}_{|\vec{i}+E} = x_{|\vec{i}+E}$;
- $y^{\vec{i}}_{|\vec{i}+E} = y_{|\vec{i}+E}$;
- $F^t(x^{\vec{i}}) = y^{\vec{i}}$ for some $t \geq 0$.

Lemma 5.3.4 (Local reachability lemma). *Let G be a $\prec$-freezing CA that (context sensitively) simulates a CA F, both of dimension d. Then there exists a finite set $E \subseteq \mathbb{Z}^d$ such that, if a configuration y is E-locally reachable by F from a configuration x, then $\overline{\phi}(y) \prec \overline{\phi}(x)$ where $\overline{\phi}$ is the encoding map given by the simulation as in Definition 5.1.1.*

Proof. Using the notations of definition 5.1.1, there exists some finite set $E \subseteq \mathbb{Z}^d$ such that for any $x, y \in Q_F^d$ and any $\vec{i} \in \mathbb{Z}^d$ it holds:

$$x|_{\vec{i}+E} = y|_{\vec{i}+E} \Rightarrow \overline{\phi}(x)|_{b\vec{i}+B} = \overline{\phi}(y)|_{b\vec{i}+B}.$$

Suppose now that y is E-locally reachable from x by F. We have in particular $\overline{\phi}(y^{\vec{i}}) \prec \overline{\phi}(x^{\vec{i}})$ because G is $\prec$-freezing and the simulation ensures that $\overline{\phi}(y^{\vec{i}})$ is in the orbit of of $\overline{\phi}(x^{\vec{i}})$ under G because $y^{\vec{i}}$ is in the orbit of $x^{\vec{i}}$ under F. From the E-locality condition and the remark above we deduce:

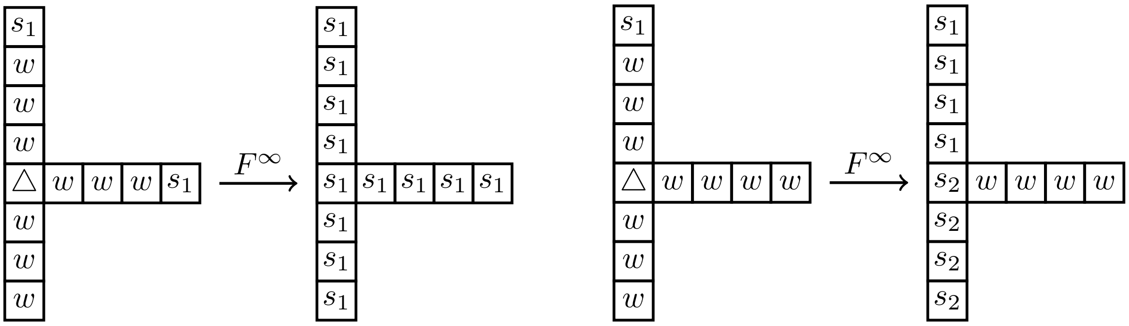
$$\forall j \in B : \ \overline{\phi}(y)_{b\vec{i}+j} \prec \overline{\phi}(x)_{b\vec{i}+j}$$

and since the relation holds for any $\vec{i}$ we finally have: $\overline{\phi}(y) \prec \overline{\phi}(x)$. □

If F is a freezing CA and c is any configuration then the limit $\lim_{t\to\infty} F^t(c)$ always exists (in the Cantor topology), is always a fixed point, and will be denoted $F^\infty(c)$ [33].

Theorem 5.3.5. *For any $d \geq 1$, there is no freezing monotone CA of dimension d which is FCA_{VN_d}-universal.*

Proof. The case of dimension 1 is already handled by Theorem 5.3.1. We do the proof for $d = 2$ using F defined above. It is straightforward to extend the argument to higher dimensions. Suppose by contradiction that there is a monotone $\prec$-freezing G that can simulate F and let E be the set given by Lemma 5.3.4 for this simulation. Let us define x^n, x^∞ and y^n, y^∞ as follows (see Figure 5.6):



(a) Configuration y^n with $n = K = 3$.

(b) Configuration y^∞ with $K = 3$.

Figure 5.6: Configurations y^n and y^∞ and the limit fixed point reached under F.

$$x^n(\vec{i}) = \begin{cases} s_2 & \text{if } \vec{i} = (0,b) \text{ with } b > -n \\ s_1 & \text{if } \vec{i} = (0,b) \text{ with } b \leq -n \\ 0 & \text{else.} \end{cases}$$

$$y^n(\vec{i}) = \begin{cases} s_1 & \text{if } \vec{i} = (0,b) \text{ with } b > K \\ w & \text{if } \vec{i} = (0,b) \text{ with } b \leq K \text{ and } b \neq 0 \\ \triangle & \text{if } \vec{i} = (0,0) \\ s_1 & \text{if } \vec{i} = (a,0) \text{ with } a > n \\ w & \text{if } \vec{i} = (a,0) \text{ with } a \leq n \text{ and } a > 0 \\ 0 & \text{else.} \end{cases}$$

where K is a large enough constant (compared to E) and x^∞ (resp. y^∞) is the limit of x^n (resp. y^n) when n goes to ∞. Choosing $n = K$ large enough (compared to E) it is straightforward to check that x^∞ (resp. y^n) is E-locally reachable from x^n (resp. y^∞). From Lemma 5.3.4, we deduce that $\overline{\phi}(x^\infty) \prec \overline{\phi}(x^n)$ and $\overline{\phi}(y^n) \prec \overline{\phi}(y^\infty)$ where $\overline{\phi}$ is the encoding map involved in the simulation of F by G. Thus we also have $G^\infty\big(\overline{\phi}(y^n)\big) \prec G^\infty\big(\overline{\phi}(y^\infty)\big)$ (by monotonicity of G), which is equivalent to $\overline{\phi}(F^\infty(y^n)) \prec \overline{\phi}(F^\infty(y^\infty))$ by the simulation. Denoting by P the half-plane of all positions (a,b) with $b < -n$, we have $F^\infty(y^\infty)_{|P} = x^\infty_{|P}$ and $F^\infty(y^n)_{|P} = x^n_{|P}$. This translates by the simulation into $\overline{\phi}(F^\infty(y^\infty))_{|P'} = \overline{\phi}(x^\infty)_{|P'}$ and $\overline{\phi}(F^\infty(y^n))_{|P'}$ $= \overline{\phi}(x^n)_{|P'}$ where P' is some half-plane contained in P (depending on parameters of the simulation). We reached a contradiction because the left-hand terms and the right-hand terms of this pair of equality compare differently with respect to $\prec$ as established above, and they cannot be all equal because $\overline{\phi}(x^n)$ is distinct from $\overline{\phi}(x^\infty)$ over P' by injectivity of $\overline{\phi}$. □

5.4 On the simulation power of FCA

A freezing-universal CA can simulate all freezing CA of same dimension, but it can simulate CAs which are not freezing. For instance, a CA with states $Q = \{0,\ldots,k\} \cup \{\iota\}$ and neighborhood $V = \{0,1\}$ that increases or decreases the local state depending on the presence of ι in the context, defined by the local map:

$$f(a,b) = \begin{cases} \iota & \text{if } a = \iota, \\ \max(a+1,k) & \text{else if } b = \iota, \\ \min(a-1,0) & \text{else.} \end{cases}$$

Such a CA is not freezing but can be simulated by a freezing CA (which can be seen directly, and is also a consequence of Theorem 5.4.2 below). However it is straightforward to check that a freezing CA cannot simulate a CA where there is no bound on the number of state change per cell.

Given a CA $F : Q^{\mathbb{Z}^d} \to Q^{\mathbb{Z}^d}$, we say that $e : Q^{\mathbb{Z}^d} \to \{0,\ldots,k\}^{\mathbb{Z}^d}$ is an *explicit local energy* for F if it is a sliding block map, *i.e.* $\forall c \in Q^{\mathbb{Z}^d}, \forall z \in \mathbb{Z}^d,\ e(c)_z = \lambda(c|_{z+N})$ for some finite $N \subseteq \mathbb{Z}^d$ and local map $\lambda : Q^N \to \{0,\ldots,k\}$, and verifies $\forall c \in Q^{\mathbb{Z}^d}, \forall z \in \mathbb{Z}^d$:

- $e(F(c))_z \leq e(c)_z$ and
- $F(c)_z \neq c_z \Rightarrow e(F(c))_z < e(c)_z$.

An explicit local energy for some CA F gives a proof that its cells can only change a bounded number of time steps their state. For instance, the example given above admits the following explicit local energy:

$$e(c)_z = \begin{cases} 0 & \text{if } c_z = \iota, \\ c_z & \text{else if } c_{z+1} \neq \iota, \\ k - c_z & \text{else.} \end{cases}$$

Such local energy maps are also naturally found in some classes of CA like FCA or nilpotent CAs (recall that F is nilpotent if F^n is a *constant* map for some n).

Lemma 5.4.1. *Any freezing CA and any nilpotent CA admits an explicit local energy.*

It turns out that we can characterize the set of CA that can be simulated by freezing CAs through the notion of explicit local energy.

Theorem 5.4.2. *A CA F admits an explicit local energy if and only if it is context-sensitive simulated by a freezing CA G.*

Proof. First suppose that F admits an explicit local energy $e : Q^{\mathbb{Z}^d} \to \{0,\dots,k\}^{\mathbb{Z}^d}$. We consider G over alphabet $Q_G = Q \times \{0,\dots,k\} \cup \{\epsilon\}$, with neighborhood the union of that of F and e, and defined as follows:

- if ϵ is not in the neighborhood and the two components of states locally look like $c \in Q^{\mathbb{Z}^d}$ and $e(c) \in \{0,\dots,k\}^{\mathbb{Z}^d}$ then the new state has components defined by $F(c)$ and $e(F(c))$;
- in any other case generate the state ϵ.

Defining the order $\preceq$ on Q_G by

- $\epsilon \preceq (q,i)$ for any $q \in Q$ and $i \in \{0,\dots,k\}$;
- $(q,i) \preceq (q',i')$ if $i < i'$;
- $x \preceq x$ for any $x \in Q_G$,

it is straightforward to check that G is $\preceq$-freezing. Let $\phi : Q^{\mathbb{Z}^d} \to (Q \times \{0,\dots,k\})^{\mathbb{Z}^d}$ be the encoding defined by $\phi(c)$ equal to c on the first component of states and $e(c)$ on the second, it is straightforward to check that $\phi \circ F = G \circ \phi$ which shows that G simulates F.

To prove the converse direction of the theorem, suppose now that a freezing CA G simulates a CA F with slowdown T and block size m. Since iterating and applying a grouping operation on a freezing CA gives a freezing CA, we can suppose without loss of generality that $m = 1$ and $T = 1$. That is to say the simulation holds through the encoding $\overline{\phi}$ coming from a local map $\phi : Q_G^V \to Q_F$ so that we have: $\overline{\phi}(F(c)) = G(\overline{\phi}(c))$. Lemma 5.4.1 ensures that G admits an explicit local energy $e : Q_G^{\mathbb{Z}^d} \to \{0,\dots,k\}^{\mathbb{Z}^d}$. One can check that $e \circ \overline{\phi} : Q_F^V \to \{0,\dots,k\}$ is the local map of an explicit local energy Ψ for F, indeed:

- $\Psi(F(c))_z = e(\overline{\phi}(F(c))_z = e(G(\overline{\phi}(c)))_z \leq e(\overline{\phi}(c))_z = \Psi(c)_z$;
- moreover, if $F(c)_z \neq c_z$ then the condition of local injectivity on ϕ (item 1 of Definition 5.1.1) ensures that $\overline{\phi}(F(c))_z \neq \overline{\phi}(c)_z$ that is to say $G(\overline{\phi}(c))_z \neq \overline{\phi}(c)_z$. e being an explicit local energy for G we deduce that: $e(G(\overline{\phi}(c)))_z < e(\overline{\phi}(c))_z$ that is to say $\Psi(F(c))_z < \Psi(c)_z$.

□

5.5 Conclusion

We have studied the freezing cellular automata from the point of view of intrinsic universality. First, we show that with the usual definition of simulation it is not possible to find an intrinsically universal freezing cellular automata, so we have introduced an adequate and nontrivial definition that allows us to find intrinsic universality.

For the construction of a intrinsically universal automata we have given an abstract structure with the sufficient components so that a cellular automata can be intrinsically universal. Eventually, we could use this same abstract structure to build more general intrinsically universal cellular automata, for example to look for intrinsic universality in bounded change cellular automata.

Using the previous construction, we have built an intrinsically universal cellular automaton with von Neumann neighborhood and three changes and we explore how the number of changes decreases if we relax the neighborhood or the number of dimensions.

We also studied the limitations of the previous results, showing that it is not possible to decrease the number of changes to find intrinsically universal cellular automata, because these automata can not cross signals. It is not possible to find intrinsically universal freezing cellular automata that are also monotonous, since they reach the same fixed point independent of the updating scheme, which leads to a contradiction trying to simulate non-monotonous automata.

Finally, we have characterized the cellular automata that can be simulated by freezing cellular automata, obtaining that they are those to which a decreasing local energy can be defined.

These results can be extended in several ways. On the one hand we can look for intrinsic universality in more general cellular automata, such as bounded changes and study if these are closed by simulation, i.e. if they can only simulate cellular automata bounded changes. If it is the case, the intrinsic universal automata would be the maximal element by simulation of this family of cellular automata, in the same way that it happens in the reversible cellular automata.

On the other hand, we can look for intrinsic universality in more particular cellular automata, as in the families of freezing cellular automata where there is no intrinsic universality, restricting the intrinsic universality to the ability to simulate automatons within this family. So we could look for a freezing, one change, von Neumann neighborhood cellular automaton capable of simulating ot other freezing, one change, von Neumann neighborhood cellular automata and analogously with monotone freezing cellular automata.

Regarding the latter, if we consider asynchronous iteration schemes, they are very close to the aTAM model and could be studied as a generalization of these or others models such as the Deterministic Tilings.

Chapter 6

On the Complexity of the Stability Problem of Binary Freezing Totalistic Cellular Automata

Consider a one or two dimensional cellular automaton on a finite torus and states *active* and *inactive* or 1 and 0.

An important problem in complexity of CA consists in predicting the future state of a cell, given an initial configuration. This decision problem is called PREDICTION [31]. In computational complexity the question is how fast we could determine the solution of PREDICTION?, and in particular if we may answer faster than simulating of the automaton. In freezing cellular automata, every initial configuration converges in at most N steps to a fixed point (where N the size of the torus), thus if we could decide PREDICTION for a sufficiently large time, then we could compute if a cell changes or remains *stable* (always in the same state). We call this problem STABILITY.

In order to find a FCA with higher complexity, the result of [33] shows that it is necessary to study FCA in more than one dimension. In this context, we should mention that D. Griffeath and C. Moore studied the *Life without Death* automaton (i.e. the game of life such that active sites remains unchanged), showing that the STABILITY problem for this rule is **P**-Complete [30]. We remark that Life without Death is a two-dimensional freezing cellular automaton with Moore neighborhood.

Moreover, in [39] the *freezing majority cellular automaton*, also known as *bootstrap percolation model* was studied in arbitrary undirected graph. In this case, an inactive cell becomes active if and only if the active cells are the most represented in its neighborhood. It was proven that STABILITY is **P**-Complete over graphs whose maximum degree is (number of neighbors) ≥ 5. Otherwise (graphs with maximum degree ≤ 4), the problem is in **NC**. This clearly includes the two dimensional case, with von Neuman neighborhood.

In this chapter we study the two simplest ways to tessellate the bi-dimensional grid: tessellation with triangles and with squares and we consider the von Neumann neighborhood. In each one of theses grids we study the family of *freezing totalistic cellular automata* (FTCA).

More precisely, we classify FTCAs in four groups:

- Simple rules: Rules that exhibit very simple dynamics which reach fixed points in a constant number of steps.
- Topological rules: Rules where the stability of a cell depends on some topological property given by the initial configuration.

- Algebraic rule: Rules where the dynamics can be accelerated, exploiting some algebraic properties given by the rule.
- Turing universal rules: Rules capable of simulating Boolean circuits and capable to simulate Turing computation.
- Fractal growing rules: Rules that produce fractal shapes growing.

The present chapter is structured as follows: in Section 6.2 , the FTCA for the triangular grid are studied. In Section 6.3, we study the FTCA on the square grid. In Section 6.4, we extend our study to FCA with rotation invariant rules. In Section 6.5, we study a weaker version of the stability problem. Finally, in Section 6.6 we give some conclusions.

The content of this chapter corresponds to publication *On the Computational Complexity of the Freezing Non-strict Majority Automata* [3] in the conference AUTOMATA 2017, and has been accepted for publication on journal *Information and Computation*, for a special issue of the same conference.

6.1 Preliminaries

In this chapter we will study the family of freezing totalistic cellular automata (FTCA) with von Neumann neighborhood and two states over the square and triangular grid. In this family, the cells in state 1 (active cells) remain active, because are freezing, and the cells in state 0 (inactive cells) are activated depending the sum of the state of their neighbors. This sum is always less or equal to $|N|$.

Let f the local function of a FTCA and $\mathcal{I}_f \subseteq \{0, 1, 2, 3, 4\}$. The number i in $\mathcal{I}_f$ means that f activates the central cell if it has i active neighbors. Thus, there is a bijection between the sets $\mathcal{I}_f$ and the FTCA with local rule f. From this bijection we denote a FTCA with local rule f as the concatenation of the elements of $\mathcal{I}_f$ increasingly. For example, the majority vote FCA in triangular grid activates a cell if it has 2 or 3 activated neighbors, then we call this rule the rule 23.

Given that the numbers of sets $\mathcal{I}_f$ is $2^{|N|+1}$ then there is 16 FTCA in the triangular grid and 32 in the square grid, but we will center our analysis on the rules where 0 is a quiescent state, i.e. if the sum of the states of the neighborhood equals to 0 remains the center cell inactive, obtaining 8 FTCA in the triangular grid and 16 in the square grid with 0 as quiescent state.

Remember that in this CA the cells in state 1 remain in state 1 forever and the cells in state 0 can change, but not every one do it.

6.2 Triangular Grid

We will start our study over the regular grid where each cell has three neighbors, as Figure 3.8a. In this topology, the sixteen FTCA are reduced to eight non-equivalent, considering the inactive state as a quiescent state. According to our classifications, the eight FTCAs in the triangular grid are grouped as follows:

- Simple rules: ϕ, 123 and 3.
- Topological rules: 2 and 23.
- Algebraic rule: 12.
- Fractal growing rules: 1 and 13.

Is easy to check that simple rules are in **NC**. For rule ϕ, we note that every configuration is a fixed point (then STABILITY for this rule is trivial). For rule 123, no site is stable unless the configuration consists of every cell inactive. We can check in time $\mathcal{O}(\log n)$ and $\mathcal{O}(n^2)$ processors whether a configuration contains an active cell using a prefix-sum algorithm (sum the states of all cells, and then decide if the result is different than 0). Finally, for rule 3 we notice that all dynamics reach a fixed point after one step. Therefore, we check if the initial neighborhood of site u makes it active in the first step (this can be decided in $\mathcal{O}(\log n)$ time in a sequential machine).

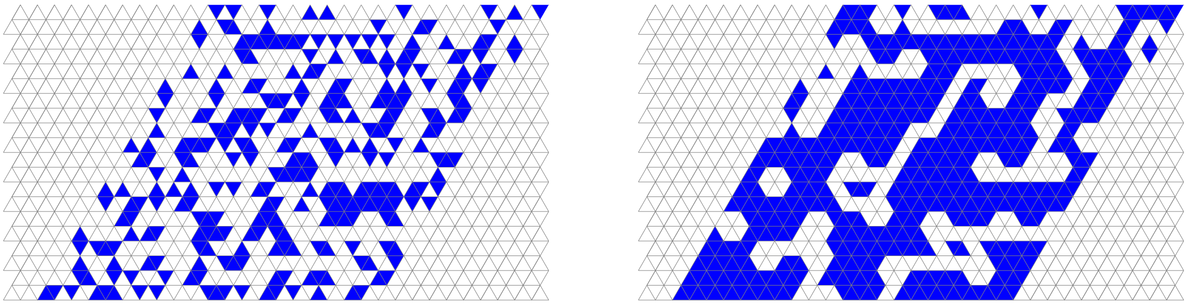
We continue our study with the Topological FTCA.

6.2.1 Topological Rules

We say that rules 2 and 23 are *topological* because, as we will see, we can characterize the stable sites according to some topological properties of the initial configurations.

As we mentioned before, rule 23 is a particular case of the freezing majority vote CA (that we called simply Majority). In [39] the authors show that STABILITY for Majority is in **NC** over any graph with degree at most 4. This result is based on a characterization of the set of stable cells, that can be verified by a fast-parallel algorithm. Thus we can apply this result to solve STABILITY for rule 23, considering the triangular grid as a graph of degree 3. Then we have the next theorem:

Theorem 6.2.1 ([39]). *There is a fast-parallel algorithm that solves* STABILITY *for 23 in time* $\mathcal{O}(\log^2 n)$ *and* $\mathcal{O}(n^4)$ *processors. Then* STABILITY *for 23 is in* ***NC***.



(a) Initial random configuration.

(b) Time 9 (fixed point).

Figure 6.1: Example of fixed point for the rule 23. The cells in state 0 in the fixed point are stable cells.

For the sake of completeness, we give the main ideas used to prove Theorem 6.2.1. This is to find a characterization of the set of stable sites.

Proposition 6.2.2 ([39]). *Consider the freezing majority vote CA defined over a graph G of degree at most 4. Let c be a configuration of G, and let $G[0]$ be the subgraph of G induced by the vertices (cells) which are inactive according to c.*

An inactive vertex u is stable if and only if,

(i) u belongs to a cycle in $G[0]$, or

(ii) u belongs to a path P in $G[0]$ where both endpoints of P are contained in cycles in $G[0]$.

Moreover, there is a fast-parallel algorithm that checks conditions (i) and (ii) in time $\mathcal{O}(\log^2 n)$ using $\mathcal{O}(4^2)$ processors.

Therefore, the proof of Theorem 6.2.1 consists in (1) noticing that a finite configuration on the triangular grid, seen as a torus, is a graph of degree 3, then it satisfies the hypothesis of Proposition 6.2.2; (2) using the algorithm given in Proposition 6.2.2 to check whether the given site u is stable.

We will use the previous result to solve the stability problem for rule 2.

Theorem 6.2.3. Stability *is in* ***NC*** *for the freezing CA* 2*.*

Proof. When we compare rule 2 and rule 23, we noticed that they exhibit quite similar dynamics. Indeed, a cell u which is stable for rule 23 is also stable for rule 2. Therefore, to solve Stability for 2 on input configuration x and cell u, we can first solve Stability for 23 on those inputs using the algorithm given by Theorem 6.2.1. When the answer of Stability for 23 is *Accept*, we know that Stability for 2 will have the same answer. In the following, we focus in the case where the answer of Stability for 23 is *Reject*, i.e. u is not stable on configuration x in the dynamics of rule 23.

Suppose that u is stable for rule 2, but is not stable for rule 23. Let t be the first time-step where u becomes active in the dynamics of rule 23. Note that, since u is stable for rule 2, necessary in step $t-1$ the three neighbors of u are active. Moreover, at least two of them simultaneously became active in time $t-1$.

Let now G be the graph representing the cells of the triangular grid covered by configuration x. Let $G[0]$ be the subgraph of G induced by the initially inactive cells, and let $G[0,u]$ be the connected component of $G[0]$ containing cell u. We claim that $G[0,u]$, in the dynamics of rule 23, every vertex (cell) in $G[0,u]$ must become active before u, i.e. in a time-step strictly smaller than t.

Claim 1: Every vertex of $G[0,u]$ is active after t applications of rule 23.

Indeed, suppose that there exists a vertex (cell) w in $G[0,u]$ that becomes active in a time-step greater than t. Call P a shortest path in $G[0,u]$ that connects u and w, and let u^* be the neighbor of u contained in P. Note that except the endpoints, all the vertices (cells) in P have at least two neighbors in P, which are inactive. Moreover, both endpoints of P are inactive at time t. Therefore, all the vertices in P will be inactive in time t. This contradicts the fact that the three neighbors of u become active before u.

Claim 2: $G[0,u]$ is a tree.

Indeed, $G[0,v]$ is connected, since it is defined as a connected component of $G[0]$ containing u. Moreover, suppose that $G[0,u]$ contains a cycle C. From Proposition 6.2.2, we know that all the cells in C are stable, which contradicts Claim 1.

Call T_u the tree $G[0,u]$ rooted on u. Let d be the depth of T_u, i.e. longest path between u and a leaf of T_u.

Claim 3: Every vertex of $G[0,u]$, except u, is active after d applications of rule 2.

Notice that necessarily a leaf of T_u has two active neighbors (because they are outside $G[0,u]$) and one inactive neighbor (its parent in T_u). Therefore, in one application of rule 2, all the leaves will become active. We will reason by induction on d. Suppose that $d=1$. Then all vertices w of T_u except u are leaves, so the claim is true. Suppose now that the claim is true for all trees of depth smaller than or equal to d, but T_u is a tree of depth $d+1$. We notice in one step the leaves are the only vertices of T_u that become active (every other vertex has two inactive neighbors). Then, after one step, the inactive sites of T_u induce a tree T'_u of depth d. By induction hypothesis, all the cells in T'_u, except u, become active after d applications of rule 2. We deduce the claim.

Let u_1, u_2, u_3 be the three neighbors of u. For $i \in \{1, 2, 3\}$, call T_{u_i} the subtree of T_u rooted at u_i, obtained taking all the descendants of u_i in T_u. Call d_i the depth of T_{u_i}. Without loss of generality, $d_1 \geq d_2 \geq d_3$.

Claim 4: u is stable for the dynamics of rule 2 but not for the dynamics of rule 23, if and only if $d_1 = d_2 \geq d_3$.

Recall that u is stable for the dynamics of rule 2 but not for the dynamics of rule 23 if and only if u has three active neighbors at time-step t, and at least two of them become active at time $t-1$. The claim follows from the application of Claim 3 to trees T_{u_1}, T_{u_2} and T_{u_3}.

We deduce the following fast-parallel algorithm solving Stability for 2: Let x be the input configuration and u the cell that we want to decide stability. First, use the fast-parallel algorithm given by Theorem 6.2.1 to decide if u is stable for the dynamics of rule 23 on configuration x. If the answer is affirmative, then we decide that (x, u) is an *Accept*-instance of Stability for 2. If the answer is negative, the algorithm looks for cycles in $G[0, u]$. If there is a cycle, then the algorithm *Rejects*, because Claim 2 implies that u cannot be stable for rule 2. If $G[0, u]$ is a tree, then the algorithm computes in parallel the depth d_v of the subtrees T_v, for each $v \in N(u)$. Finally, the algorithm accepts if the conditions of Claim 4 are satisfied, and rejects otherwise.

The steps of the algorithm are represented in Algorithm 5.

Algorithm 5 Solving Stability 2

```
Input: $x$ a finite configuration of dimensions $n \times n$ and $u$ a cell.
 1: if the answer of Stability for rule 23 is Accept on input $(x, u)$ then
 2:   return Accept
 3: else
 4:   Compute $G[0, u]$
 5:   Compute $C$ the set of cycles of $G[0, u]$
 6:   if $C \neq \emptyset$ then
 7:     return Reject
 8:   else
 9:     for all $v \in N(u)$ do in parallel
10:       Compute $d_v$ the depth of $T_v$
11:     end for
12:     if $\exists a, b, c \in (N(u)) : d_a = d_b \geq d_c$ then
13:       return Accept
14:     else
15:       return Reject
16:     end if
17:   end if
18: end if
```

Let $N = n^2$ be the size of the input. Algorithm 5 runs in time $\mathcal{O}(\log^2 N)$ using $\mathcal{O}(N^3/\log N)$ processors. Indeed, the condition of step **1** can be checked in time $\mathcal{O}(\log^2 N)$ using $\mathcal{O}(N^2)$ processors according the algorithm of Theorem 6.2.1. Step **4** can be done in time $\mathcal{O}(\log^2 N)$ using $\mathcal{O}(N^2)$ processors using a connected components algorithm given in [44]. Step **5** can be done in time $\mathcal{O}(\log^2 N)$ using $\mathcal{O}(N^3/\log N)$ processors using a bi-connected components algorithm given in [44]. Step **10** can be solved in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors using a vertex level algorithm given in [44]. Finally, Step **12** can be done in $\mathcal{O}(\log N)$ time in a sequential machine. □

6.2.2 Algebraic Rule

Now we continue with the study of rule 12. We say that this rule is *algebraic* because, as we will see, we can speed-up the dynamics using some algebraic properties of this rule. Using this speed-up we will build an algorithm that decides the stability of a cell much faster than the simple simulation of the automaton. In other words, we will show that Stability for rule 12 is in **NC**.

Let x be a finite configuration on the triangular grid, u a cell. Let v be a neighbor of u. We define a *semi-plane* S_v as a partition of the triangular grid in two parts, cut by the edge of the triangle that share cell u and v, as shown in Figure 6.2.

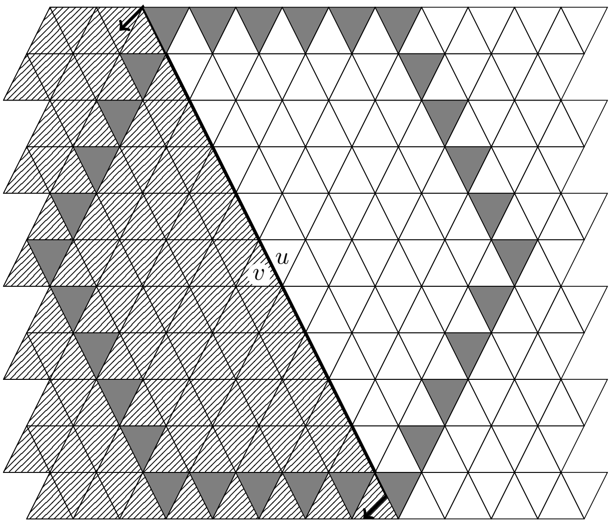


Figure 6.2: Triangular grid divided in semi-planes according to u and v. The hatch pattern represent the semi-plane S_v Gray cells are at the same distance from u.

We say that two cells v_1, v_2 are at *distance* d if a shortest path connecting v_1 and v_2 is of length d. In the following, we call D_d the set of cells at distance d from u. We can be more explicit and use the property above to give a way to speed-up the dynamics of rule 12.

Lemma 6.2.4. *Let $d \geq 2$ be the distance from u to the nearest active cell. Then the distance to the nearest cell to u in $F(c)$ is $d-1$.*

Proof. Let w be an active cell at distance d of u in configuration x, and call P a shortest (u, w)-path. Call w_1 the neighbor of w contained in P, and let w_2 be the neighbor of w_1 in P different than w (these cells exist since $d \geq 2$). Note that w_2 might be equal to u. Since P is a shortest path, w_2 is at distance $d-2$ from u. Then all the neighbors of w_2 are inactive, so w_2 it is necessarily inactive in $F(c)$. Moreover, w_1 has more than one active neighbor, and less than three active neighbors, so w_1 is active in $F(c)$. Then the distance from u to the nearest active cell in $F(c)$ is $d-1$. □

Lemma 6.2.5. *Let $d \geq 2$ be the distance from u to the nearest active cell, and let $v \in N(u)$. Then v is active after $d-1$ applications of rule 12 (i.e. $F^{d-1}(c)_v = 1$) if and only there exists an active cell in $S_v \cap D_d$.*

Proof. We reason by induction on d. In the base case, $d = 2$, suppose that S_v does not contain an active site at distance 2. Then every neighbor of v is inactive in the initial configuration, so v is inactive after one application of rule 12 (i.e. $F(c)_v = 0$). Conversely, if $F(c)_v = 0$, then every neighbor of v is initially inactive, in particular all the sites in S_v at distance 2 from u.

Suppose now that the statement of the lemma is true on configurations where the distance is d, and let c be a configuration where the distance from u to nearest active cell is $d+1$. Let c' be the configuration obtained after one application on c of rule 12 (i.e. $c' = F(c)$).

Claim 1: $F^{d-1}(c')_v = 1$ if and only if in c' there exists an active cell in $S_v \cap D_d$.

From Lemma 6.2.4, the distance from u to the nearest active cell in c' is d. The claim follows from the induction hypothesis.

Claim 2: Suppose that $F^{d-1}(c')_v = 0$. Then in c, all the cells in $S_v \cap D_{d+1}$ are inactive.

Notice that, from Claim 1, the fact that $F^{d-1}(c')_v = 0$ implies that in c' all the cells in $D_v \cap S_v$ must be inactive. Suppose, by contradiction, that there is a cell w in $S_v \cap D_{d+1}$ that is active in c. Let w' be a neighbor of w contained in $S_v \cap D_d$, and let w'' be a neighbor of w' not contained in D_{d+1} (then w' belongs to $D_d \cup D_{d-1}$). Note that w' has an active neighbor in c, but must be inactive in c'. The only option is that all the neighbors of w' are active in c, in particular w'' is active in c. This contradicts the fact the nearest active cell is at distance $d+1$ in c.

Claim 3: Suppose that $F^{d-1}(c')_v = 1$. Then there is a cell in $S_v \cap D_{d+1}$ that is active in c.

From Claim 1, the fact that $F^{d-1}(c')_v = 0$ implies that there is a cell $w \in S_v \cap D_d$ that is active in c'. Suppose by contradiction that all the cells in $S_v \cap D_{d+1}$ are inactive in c. Since w is active in c', necessarily w has at least one neighbor w' that is active in c. Since w' is not contained in $S_v \cap D_{d+1}$ (because we are supposing that all those cells are inactive in c), we deduce that w' belongs to $D_d \cup D_{d-1}$. This contradicts the fact the nearest active cell is at distance $d+1$ in c.

We deduce that $F^{d-1}(c')_v = 1$ if and only if there is a cell in $S_v \cap D_{d+1}$ that is active in c. Since $c' = F(c)$, we obtain that $F^d(c)_v = 1$ if and only if there is a cell in $S_v \cap D_{d+1}$ that is active in c. □

Theorem 6.2.6. Stability *is in* ***NC*** *for the freezing CA* 12.

Proof. In our algorithm solving Stability for 12, we first compute the distance d to the nearest active cell from u (if every cell is inactive, our algorithm trivially accepts). Then, for each $v \in N(u)$, the algorithm computes the set of cells $S_v \cap D_d$, and checks if that set contains an active cell. If it does, we mark v as *active*, and otherwise we mark v as *inactive*. Finally, the algorithm rejects if the three neighbors of u are active, and accepts otherwise. The steps of this algorithm are described in Algorithm 6

From Lemma 6.2.5, we know that v becomes active at time $d-1$ if and only if $S_v \cap D_d$ contains an active cell in the initial configuration. Since the nearest active cell from u is at distance d, necessarily after $d-1$ steps at least one of the three neighbors of u will become active. If the three neighbors of u satisfy the condition of Lemma 6.2.5, then the three of them will become active in time $d-1$, so u will remain inactive forever. Otherwise, u will have more than one and less than three active neighbors at time-step $d-1$, so it will become active at time d.

Algorithm 6 Solving Stability 12

```
Input: $x$ a finite configuration of dimensions $n \times n$ and $u$ a cell.
 1: if For all cell $w$, $x_w = 0$ then
 2:     return accept
 3: else
 4:     Compute a matrix $M = (m_{ij})$ of dimensions $2n^2 \times 2n^2$ such that
          $m_{ij}$ is the distance from cell $i$ to cell $j$.
 5:     Compute the distance $d$ to the nearest active cell from $u$.
 6:     for all $v \in N(u)$ do in parallel
 7:         Compute the set of cells $S_v \cap D_d$
 8:         if there exists $w \in S_v \cap D_d$ such that $x_w = 1$ then
 9:             Mark $v$ as active
10:         else
11:             Mark $v$ as inactive
12:         end if
13:     end for
14:     if there exists $v$ in $N(u)$ that is marked inactive then
15:         return Reject
16:     else
17:         return Accept
18:     end if
19: end if
```

Let $N = n^2$ the size of the input. This algorithm runs in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors. Indeed, the verifications on steps **1-3** and **8-10** can be done in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors using a prefix-sum algorithm. Finally, step **7** can be done in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, assigning one processor per cell and solving three inequations of kind $ax + by < c$.

□

6.3 Square Grid

Now we continue our study, considering the square grid. As we said in the Chapter 3, we can define 32 different FTCAs over this topology. Again, considering the inactive state as a quiescent state, the set of non-equivalent FTCAs is reduced to 16. According to our classifications, this list of FTCAs is grouped as follows:

- Simple rules: ϕ, 1234 and 4.
- Topological rules: 234, 3 and 34.
- Algebraic rules: 12, 123, and 124.
- Turing Universal rules: 2, 24.
- Fractal growing rules: 1, 13, 14 and 134.

In complete analogy to the triangular topology, we verify that the Stability problem in simple rules is **NC**. We will directly continue then with the Topological Rules.

6.3.1 Topological Rules

We study these rules by characterizing their fixed points and building a faster algorithm to find them. This characterization is based on the structure of the set of stable cells, called *stable sets*. Naturally, the structure of stable-sets depends on the rule.

Rules 34 and 3.

First, notice that the rule 34 corresponds to a freezing version of the majority automaton over the square grid. We remark that a finite configuration over the square grid, seen as a torus, is a regular graph of degree 4. Therefore, we can use the algorithm given in Proposition 6.2.2 to check whether a given site is stable for rule 34. We deduce the following theorem (also given in [39]).

Theorem 6.3.1 ([39]). STABILITY *is in* ***NC*** *for rule 34.*

Likewise, in analogy to the behavior of rule 2 with respect to rule 23 in the triangular grid, we can use the algorithm solving STABILITY for the rule 34 to solve STABILITY for the rule 3. Let (x, u) be an instance of problem STABILITY. Clearly, if u is stable for rule 34 we have that u is stable for rule 3. Suppose now that u is not stable for rule 34 but stable for rule 3. Let $G[0, u]$ the connected component of $G[0]$ containing u. Using the exact same proof used for rule 2 on the triangular grid, we can deduce that $G[0, u]$ is a tree, and we call T_u this tree rooted in u. Moreover, let u_1, u_2, u_3, u_4 be the four neighbors of u, and let T_{u_i} be the subtree of T_u obtained by taking all the descendants of u_i, $i \in \{1, 2, 3, 4\}$. Call d_i the depth of T_{u_i} which without loss of generality we assume that $d_1 \geq d_2 \geq d_3 \geq d_4$. We have that u is stable for rule 3 but not for rule 34 if and only if $d_1 = d_2 \geq d_3 \geq d_4$. We deduce that, with very slight modifications, Algorithm 5 solves STABILITY for rule 3. We deduce the following theorem.

Theorem 6.3.2. STABILITY *is in* ***NC*** *for rule 3.*

Rule 234.

Notice that rule 234 is the freezing version of the non-strict majority automaton, the CA where the cells take the state of the majority of its neighbors, and in tie case they decide to become active. In the following, we will show that the stability problem for this rule is also in **NC** by characterizing the set of stable sets. This time, the topological conditions of the stable sets will be the property of being tri-connected.

Theorem 6.3.3. STABILITY *is in* ***NC*** *for the freezing CA 234.*

Lemma 6.3.4. *Let $x \in \{0, 1\}^{[n]\times[n]}$ be a finite configuration and $u \in [n] \times [n]$ a site. Then, u is stable for $c = c(x)$ if and only if there exists a set $S \subseteq [n] \times [n]$ such that:*

- *$u \in S$,*
- *$c_u = 0$ for every $u \in S$, and*
- *$G[S]$ is a graph of minimum degree 3.*

Proof. Suppose that u is stable and let S be the subset of $[n] \times [n]$ containing all the sites that are stable for c. We claim that S satisfy the desired properties. Indeed, since S contains all the sites stable for c, then u is contained in S. Moreover, since the automaton is freezing, all the sites in S must be inactive on the configuration c. Finally, if $G[S]$ contains a vertex v of degree less than 3, it means that necessarily the corresponding site v has two non-stable neighbors that become 1 in the fixed point reached from c, contradicting the fact that v is stable.

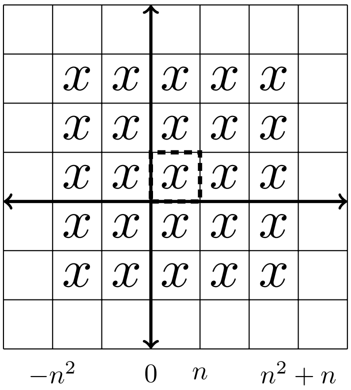


Figure 6.3: Construction of the finite configuration $D(x)$ obtained from a finite configuration x of dimension $n \times n = 2 \times 2$. Note that $D(x)$ is of dimensions 7×7.

On the other direction suppose that S contains a site that is not stable and let $t > 0$ be the minimum step such that a site v in S changes to state 1, i.e., $v \in S$ and t are such $F^{t-1}(c)_w = 0$ for every $w \in S$, and $F^t(c)_v = 1$. This implies that v has at least two active neighbors in the configuration $F^{t-1}(c)$. This contradicts the fact that v has three neighbors in S. We conclude that all the sites contained in S are stable, in particular u. □

For a finite configuration $x \in \{0,1\}^{[n]\times[n]}$, let $D(x) \in \{0,1\}^{\{-n^2-n,\dots,n^2+2n\}^2}$ be the finite configuration of dimensions $m \times m$, where $m = 2n^2 + 3n$, constructed with repetitions of configuration x in a rectangular shape, as is depicted in Figure 6.3, and inactive sites elsewhere. We also call $D(c)$ the periodic configuration $c(D(x))$.

Lemma 6.3.5. *Let $x \in \{0,1\}^{[n]^2}$ be a finite configuration, and let u be a site in $[n]\times[n]$ such that $x_u = 0$. Then u is stable for $c = c(x)$ if and only if it is stable for $D(c)$.*

Proof. Suppose first that u is stable for c, i.e. in the fixed point c' reached from c, $c'_u = 0$. Call c'' the fixed point reached from $D(c)$. Note that $D(c) \leq c$ (where $\leq$ represents the inequalities coordinate by coordinate). Since the 234 automata is monotonic, we have that $c'' \leq c'$, so $c''_u = 0$. Then u is stable for $D(c)$.

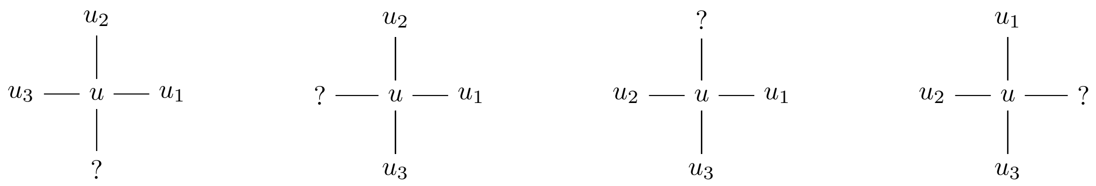
Conversely, suppose that $u \in [n] \times [n]$ is not stable for c, and let S be the set of all sites at distance at most n^2 from u. We know that in each step on the dynamics of c, at least one site in the periodic configuration changes its state, then in at most n^2 steps the site u will be activated. In other words, the state of u depends only on the states of the sites at distance at most n^2 from u. Note that for every $v \in S$, $c_v = D(c)_v$. Therefore, u is not stable in $D(c)$. □

Note that the perimeter of width n of $D(x)$ contain only inactive sites. We call this perimeter the *border* of $D(x)$, and $D(x) - B$ the *interior* of $D(x)$. Note that B is tri-connected and forms a set of sites stable for $D(c)$ thanks to Lemma 6.3.4. We call Z the set of sites w in $[m] \times [m]$ such that $D(x)_w = 0$.

Lemma 6.3.6. *Let u be a site in $[n] \times [n]$ stable for $D(c)$. Then, there exist three disjoint paths on $G[Z]$ connecting u with sites of the border B. Moreover, the paths contain only sites that are stable for $B(c)$.*

Proof. Suppose that u is stable. From Lemma 6.3.4 this implies that u has three stable neighbors. Let $0 \leq i, j \leq n$ be such that $u = (i,j)$. We divide the interior of $D(c)$ in four quadrants:

- The first quadrant contains all the sites in $D(x)$ with coordinates at the north-east of u, i.e., all the sites $v = (k,l)$ such that $k \geq i$ and $l \geq j$.



(a) Case 1 (b) Case 2 (c) Case 3 (d) Caso 4

Figure 6.4: Four possible cases for u_1, u_2 and u_3. Note that one of these four cases must exist, since u has at least three stable neighbors. From u_1 we will extend a path through the first quadrant, from u_2 a path through the second quadrant, and from u_3 a path through the third quadrant.

- The second quadrant contains all the sites in $D(x)$ with coordinates at the north-west of u, i.e., all the sites $v = (k, l)$ such that $k \leq i$ and $l \geq j$.
- The third quadrant contains all the sites in $D(x)$ with coordinates at the south-west of u, i.e., all the sites $v = (k, l)$ such that $k \leq i$ and $l \leq j$.
- The fourth quadrant contains all the sites in $D(x)$ with coordinates at the south-east of u, i.e., all the sites $v = (k, l)$ such that $k \geq i$ and $l \leq j$.

We will construct three disjoint paths in $G[Z]$ connecting u with the border, each one passing through a different quadrant. The idea is to first choose three quadrants, and then extend three paths starting from u iteratively picking different stable sites in the chosen quadrants, until the paths reach the border.

Suppose without loss of generality that we choose the first, second and third quadrants, and let u_1, u_2 and u_3 be three stable neighbors of u, named according to Figure 6.4.

Starting from u, u_1, we extend the path P_1 through the endpoint different than u, picking iteratively a stable site at the east, or at the north if the site in the north is not stable. Such sites will always exist since by construction the current endpoint of the path will be a stable site, and stable sites must have three stable neighbors (so either one neighbor at east or one neighbor at north). The iterative process finishes when P_1 reaches the border. Note that necessarily P_1 is contained in the first quadrant. Analogously, we define paths P_2 and P_3, starting from u_2 and u_3, respectively, and extending the corresponding paths picking neighbors at the north-west or south-west, respectively. We obtain that P_2 and P_3 belong to the second and third quadrants, and are disjoint from P_1 and from each other.

This argument is analogous for any choice of three quadrants. We conclude that there exist three disjoint paths of stable sites from u to the border B. □

Lemma 6.3.7. *Let u, v be two sites in $[n] \times [n]$ stable for $D(c)$. Then, there exist three disjoint (u, v)-paths in $G[Z]$ consisting only of sites that are stable for $D(c)$.*

Proof. Let u, v be stable vertices. Without loss of generality, we can suppose that $u = (i, j)$, $v = (k, l)$ with $i \leq k$ and $j \leq l$ (otherwise we can rotate x to obtain this property). In this case u and v divide the interior of $D(x)$ into nine regions (see Figure 6.5). Let $P_{u,2}, P_{u,3}, P_{u,4}$ be three disjoint paths that connect u with the border through the second, third and fourth quadrants of u. These paths exist according to the proof of Lemma 6.3.6. Similarly, define $P_{v,1}, P_{v,2}, P_{v,3}$ three disjoint paths that connect v to the border through the first, second and third quadrants of v.

Observe fist that $P_{u,3}$ touches regions that are disjoint from the ones touched by $P_{v,1}, P_{v,2}$ and $P_{v,3}$. The same is true for $P_{v,1}$ with respect to $P_{u,2}, P_{u,3}, P_{u,4}$. The first observation implies that paths $P_{u,3}$ and $P_{v,1}$ reach the border without intersecting any other path. Let w_1 and w_2 be respectively the intersections of $P_{u,3}$ and $P_{v,1}$ with the border. Let now P_{w_1,w_2} be any path in G_B connecting w_1 and w_2. We call $P_{1,3}$ the path induced by $P_{u,3} \cup P_{w_1,w_2} \cup P_{v,1}$.

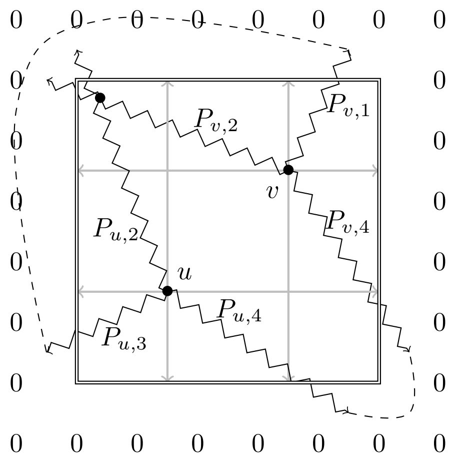


Figure 6.5: Vertices u and v divides the interior of $D(x)$ into four regions each one. Together they split the space into nine regions. According to Lemma 6.3.6, we can choose three disjoint paths connecting u and v, in such a way that each of the nine regions intersect at most one path. We use the border of $D(x)$ to connect the paths that do do not intersect in the interior of $D(x)$.

Observe now that $P_{u,2}$ and $P_{v,4}$ must be disjoint, as well as $P_{u,4}$ and $P_{v,2}$. This observation implies that $P_{u,2}$ either intersects $P_{v,2}$ or do not intersect any other path, and the same is true for $P_{u,4}$ and $P_{v,4}$. If $P_{u,2}$ does not intersect $P_{v,2}$, then we define a path $P_{2,2}$ in a similar way than $P_{1,3}$, i.e., we connect the endpoints of $P_{u,2}$ and $P_{v,2}$ through a path in the border (we can choose this path disjoint from $P_{1,3}$ since the border is tri-connected). Suppose now that $P_{u,2}$ intersects $P_{v,2}$. Let w the first site where $P_{u,2}$ and $P_{v,2}$ intersect, let $P_{u,w}$ be the u,w-path contained in $P_{u,2}$, and let $P_{w,v}$ be the w,v-path contained in $P_{v,2}$. We call in this case $P_{2,2}$ the path $P_{u,w} \cup P_{w,v}$. Note that also in this case $P_{2,2}$ is disjoint from $P_{1,3}$. Finally, we define $P_{4,4}$ in a similar way using paths $P_{u,4}$ and $P_{v,4}$. We conclude that $P_{1,3}$, $P_{2,2}$, and $P_{4,4}$ are three disjoint paths of stable sites connecting u and v in $G[Z]$.

□

Now we are ready to show our characterization of stable set of vertices.

Lemma 6.3.8. *Let $x \in \{0,1\}^{[n]\times[n]}$ be a finite configuration, and let u be a site in $[n] \times [n]$. Then, u is stable for $c = c(x)$ if and only if u is contained in a tri-connected component of $G[Z]$.*

Proof. From Lemma 6.3.5, we know that u is stable for c if and only if it is stable $D(c)$. Let S be the set of sites stable for $D(c)$. We claim that S is a tri-connected component of $G[Z]$. From Lemma 6.3.7, we know that for every pair of sites in S there exist three disjoint paths in $G[S]$ connecting them, so the set S must be contained in some tri-connected component T of $G[Z]$. Since $G[T]$ is a graph of degree at least three, and the sites in T are contained in Z, then Lemma 6.3.4 implies that T must form a stable set of vertices, then T equals S.

On the other direction, Lemma 6.3.4 implies that any tri-connected component of $G[Z]$ must form a stable set of vertices for $D(c)$, so u is stable for c. □

At this moment we are ready to study the complexity of Stability for this rule.

Proof of Theorem 6.3.3 . Let (x, u) be an input of Stability, i.e. x is a finite configuration of dimensions $n \times n$, and u is a site in $[n] \times [n]$. Our algorithm for Stability first computes from x the finite configuration $D(x)$. Then, the algorithm uses the algorithm of Proposition 3.2.7 to compute the tri-connected components of $G[Z]$, where Z is the set of sites w such that $D(x)_w = 0$. Finally, the algorithm answers *no* if u belongs to some tri-connected component of $G[Z]$, and answers *yes* otherwise.

Algorithm 7 Solving Stability 234

```
Input: $x$ a finite configuration of dimensions $n \times n$ and $u \in [n] \times [n]$ such that $x_u = 0$.
Compute the finite configuration $D(x)$ of dimensions $m \times m$ with $m = 2n^2 + 3n$
Compute the set $Z = \{w \in [m] \times [m] \ : \ D(x)_w = 0\}$.
Compute the graph $G[Z]$.
Compute the set $\mathcal{T}$ of tri-connected components of $G[Z]$.
for all $T \in \mathcal{T}$ do in parallel
    if $u \in T$ then
        return Accept
    end if
end for
return Reject
```

The correctness of Algorithm 7 is given by Lemma 6.3.8. Indeed, the algorithm answers *Reject* on input (x, u) only when u does not belong to a tri-connected component of $G[Z]$. From lemma 6.3.8, it means that u is not stable, so there exists $t > 0$ such that $F^t(c(x))_u = 1$.

Let $N = n^2$ the size of the input. Step **1** can be done in $\mathcal{O}(\log N)$ time with $m^2 = \mathcal{O}(N^2)$ processors: one processor for each site of $B(x)$ computes from x the value of the corresponding site in $B(x)$. Step **2** can be done in time in $\mathcal{O}(\log N)$ with $\mathcal{O}(N^2)$ processors, representing Z as a vector in $\{0,1\}^{m^2}$, each coordinate is computed by a processor. Step **3** can be done in time $\mathcal{O}(\log N)$ and $\mathcal{O}(N^2)$ processors: we give one processor to each site in Z which fill the corresponding four coordinates of the adjacency matrix of $G[Z]$. Step **4** can be done in time $\mathcal{O}(\log^2 N)$ with $\mathcal{O}((N^2)^4)$ processors using the algorithm of Proposition 3.2.7. Finally, steps **5** to **10** can be done in time $\mathcal{O}(\log N)$ with $\mathcal{O}(N^2)$ processors: the algorithm checks in parallel if u is contained in each tri-connected components. All together the algorithm runs in time $\mathcal{O}(\log^2 N)$ with $\mathcal{O}(N^8)$ processors. □

6.3.2 Algebraic Rules

We will now study the family of FTCA where the cells become active with one or two neighbors. We consider the rules 12, 123, 124. Of course, rule 1234 will fit in our analysis, but we already know that this rule is trivial. As we already mentioned, these rules are *algebraic* in the sense that, in order to answer the Stability problem, we will *accelerate* the dynamics using algebraic properties of these rules.

In the following, we assume that the cells are placed in the Cartesian coordinate system, where each cell is placed on a coordinate in $\mathbb{N} \times \mathbb{N}$. Moreover, without loss of generality, our decision cell is $u = (0,0)$ and the configuration c has at least one active cell. Let $\tau > 1$ be the distance from u to the first active cell. Like for rule 12 in the triangular grid, we called D_τ the set of cells at distance τ from u. Due to its shape in the squared grid, the set D_τ is called in this context the *diamond at distance τ from u.* Note that $D_\tau = \{(i,j) \in \mathbb{N}^2 : |i-j| \leq \tau\}$. We also call $d_I(\tau)$ the *diagonal at distance τ of u in the first quadrant*, defined as follows:

$$d_I(\tau) := \{(i,j) \in \mathbb{N}^2 : |i-j| = \tau \text{ and } i,j > 0\}.$$

Then, we place ourselves in the case where all the cells in $D_{\tau-1}$ are inactive.

Let c' be the configuration obtained after one step, i.e. $c' = F(c)$, where F is one of the rules in $\{12, 123, 124\}$. Notice that all the cells in $D_{\tau-1}$ will remain inactive in c'. Moreover, the states of cells in $d_I(\tau-1)$ can be computed as follows (see figure 6.6a):

$$\forall (i,j) \in d_I(\tau-1),\ c'_{i,j} = c_{i+1,j} \vee c_{i,j+1},$$

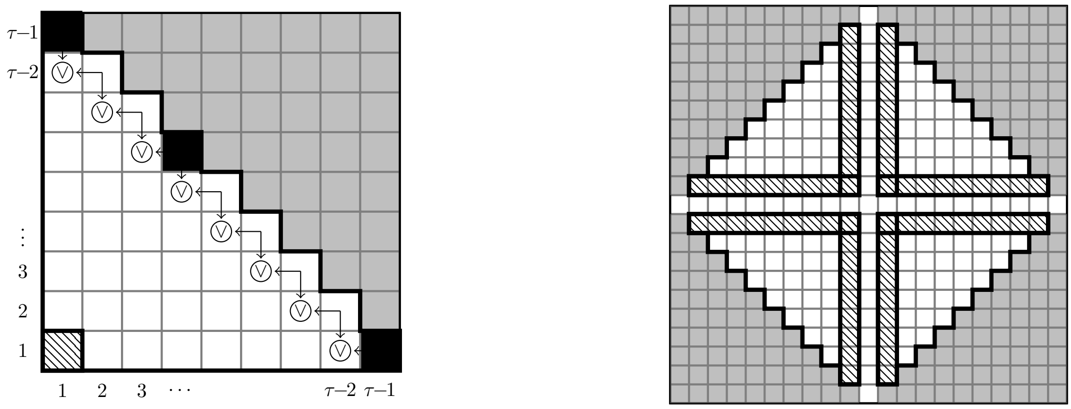
where $\vee$ is the OR operator (i.e. $c'_{i,j} = 1$ if $c_{i+1,j} = 1$ or $c_{i,j+1} = 1$). If we inductively apply this formula, we deduce:

$$F^{\tau-2}(c)_{1,1} = \bigvee_{(i,j) \in d_I(\tau)} c_{i,j}.$$

Note that if the cell $(1,1)$ is inactive at time $\tau-1$, then necessarily all the cells in $d_I(\tau)$ are inactive at time 0. Moreover, we can apply the same ideas to every cell $(i,j) \in \bigcup_{1\leq k\leq\tau} d_I(k)$ such that $i,j\geq 1$, obtaining:

$$F^{\tau-1-i}(c)_{i,1} = \bigvee_{\substack{(k,j)\,\in\, D(\tau)\\ k\geq i}} c_{k,j} \quad\text{and}\quad F^{\tau-1-j}(c)_{1,j} = \bigvee_{\substack{(i,k)\,\in\, D(\tau)\\ k\geq j}} c_{i,k}. \tag{6.1}$$

Analogously, we can define $d_{II}(\tau)$ (resp. $d_{III}(\tau)$, $d_{IV}(\tau)$) the *diagonals at distance τ of u in the second (resp. third, fourth) quadrant*, and deduce similar formulas in the other three quadrants. Concretely we can compute the states of cells $(\pm i,\pm 1)$, $i=1,...,\tau-1$ at time $\tau-i$ and the states of cells $(\pm 1,\pm j)$, $j=1,...,\tau-1$ at time $\tau-j$. These cells are represented as the hatch patterns in Figure 6.6b. We call this way of calculating cells the *OR technique*, because we compute the new cells value as the operator OR applied to the states of its neighbors.



(a) Computation of $d_I(\tau-1)$ after one application of rule F.

(b) Cells $(\pm i,\pm 1)$ and $(\pm 1,\pm j)$, where $i,j,...,\tau-1$, represented with a hatch pattern.

Figure 6.6: Computation in a diamond of size τ, where only the cells at distance τ from $(0,0)$ can be initially active.

We define the following sets of cells.

- The **north-east triangle** is the set of cells in the first quadrant between the cells in the hatch pattern (including them) and the gray zone, i.e. is the set $D_{\tau,I} = \{(i,j)\in\mathbb{N}^2 : |i-j|\leq \tau \text{ and } i,j\geq 1\}$. Analogously we define north-west, south-west and south-east triangles, and denote them $D_{\tau,II}, D_{\tau,III}$ and $D_{\tau,IV}$, respectively.

- The **north corridor** is the set of cells in the positive y-axis contained in the diamond , i.e. is the set $\{(0,i)\in\mathbb{N}^2 : 1\leq i\leq\tau\}$. Analogously we define west, south and east corridors.

Consider now a smaller diamond at distance 2 of $u=(0,0)$ depicted in Figure 6.7. As we explained, we can compute the states of cells b,d,f and h at time $\tau-1$ using the OR technique. The use of this information in order to solve STABILITY, will depend on which rule we are considering. In the following, we will show how to use this information to solve stability for rule 123, then for rule 12 and finally for rule 124.

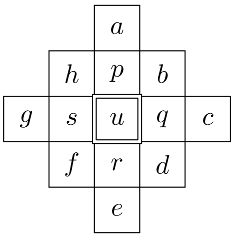


Figure 6.7: Notation for the cells in the small diamond. Cells p, q, r and s correspond to the neighbors of cell u.

Solving Stability for rule 123

Rule 123 is the simplest of the algebraic rules. Its simplicity follows mainly from the following claim. Remember that τ is the distance from u to the nearest active cell.

Claim 1: Either u becomes active at time τ or u is stable.

We know that at least one of the cells in $\{b, d, f, h\}$ will be active at time $\tau - 2$. Indeed the states of those cells depend only on the logical disjunction of the sites in the border of the diamond D_τ, and we are assuming that there is at least one active site in $D_\tau \setminus D_{\tau-1}$. Therefore, at time $\tau - 1$, necessary at least one neighbor $w \in N(u)$ will become active, since it will have more than one active neighbor, and less than four (because u is inactive at time $\tau - 1$). Suppose now that u does not become active at time τ. Since u has one active neighbor at time $\tau - 1$, the only possibility is that the four neighbors of u are active $\tau - 1$. Since the rule is freezing, u will remain stable in inactive state.

At this point, we know how to compute the states of b, d, f and h at time $\tau - 2$, and we know that the only possibility for u to become active is on time τ. Therefore, in order to decide Stability for rule 123 we need to compute the states of cells p, q, r and s at time $\tau - 1$. In the following, we show how to compute the state of site p at time $\tau - 1$. The arguments for computing cells q, r and s will be deduced by analogy (considering the same arguments in another quadrant).

Call $x^t_{(i,j)}$ the state of cell (i, j) at time t, with the convention of $x^0_{(i,j)}$ is the input state of (i, j). First, note that, for all $i \in \{0, \dots, \tau - 2\}$, the state of $(0, i)$ at time 1 will be inactive. Moreover, the state of cell $(0, \tau - 1)$ will be active if and only if at least one of its three neighbors $(-1, \tau - 1)$, $(1, \tau - 1)$ or $(0, \tau)$ is active at time 0. Then, we deduce the following formula for $x^1_{(0,\tau-1)}$:

$$x^1_{(0,\tau-1)} = x^0_{(-1,\tau-1)} \vee x^0_{(0,\tau)} \vee x^0_{(1,\tau-1)}$$

For the same reasons, we notice that at time $j > 0$ the nearest neighbor from u is in the border of $D_{\tau-j}$. Therefore,

$$x^j_{(0,\tau-j)} = x^{j-1}_{(-1,\tau-j)} \vee x^{j-1}_{(0,\tau)} \vee x^{j-1}_{(1,\tau-j)}$$

In particular

$$x^{\tau-1}_p = x^{\tau-1}_{(0,1)} = x^{\tau-2}_{(-1,1)} \vee x^{\tau-2}_{(0,1)} \vee x^{\tau-2}_{(1,1)}.$$

Remember that we know how to compute $x^{\tau-2}_{(-1,1)}$ and $x^{j-1}_{(1,1)}$ according to Equation 6.1. We deduce that we can compute ,$x^{\tau-1}_p$ as follows:

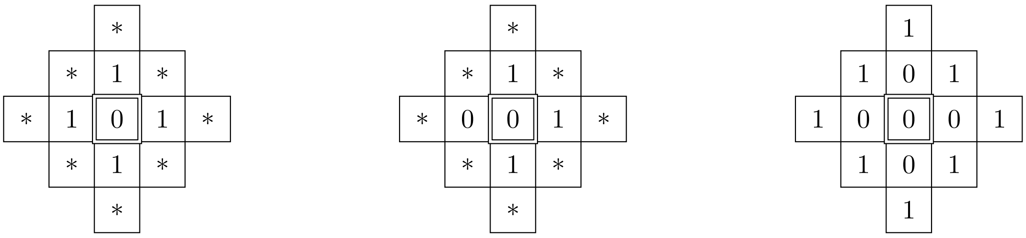
$$x^{\tau-1}_p = \bigvee_{k=1}^{\tau} x^0_{(-k,\tau-k)} \vee x^0_{(0,\tau)} \vee \bigvee_{k=1}^{\tau} x^0_{(k,\tau-k)} \tag{6.2}$$

In other words, the state of p at time $\tau - 1$ can be computed as the OR of all the cells to the north of the u contained in $D_\tau \setminus D_{\tau-1}$. Analogously we can compute $x^{\tau-1}_q$, $x^{\tau-1}_r$ and $x^{\tau-1}_s$.

Solving stability for rule 12

For rule 12 the computation of a, c, e and g is not so simple as the previous case. First of all, there is one case when cell u remains active, though we can also assume **Claim 1** for this rule.

Indeed, remember that we know that at least one of cells in $\{b, d, f, h\}$ will be active at time $\tau - 2$. Suppose that u remains inactive at time τ. There are three possibilities: (1) none of the neighbors of u will become active at time $\tau - 1$, (2) three neighbors of u become active at time $\tau - 1$; and (3) the four neighbors of u become active at time $\tau - 1$. See figure 6.8.



(a) Sum equal to 4.

(b) Sum equal to 3.

(c) Sum equal to 0.

Figure 6.8: Possibles cases of rule 12 at time τ such that u remains inactive.

Note that the in cases in Figure 6.8a and 6.8b we directly obtain that u is stable, because the decision cell is surrounded by active cells .

The case when the sum in its neighborhood is 0 is slightly more complicated. As we said, we know that at least one of $\{b, d, f, h\}$ becomes active at time $\tau - 2$. Suppose, without loss of generality, that b satisfies this condition. Moreover, we are assuming that p and q remain inactive at time $\tau - 1$. Since these cells have one active neighbor at time $\tau - 2$, the sole possibility is that cells h, a, c and d are active at time $\tau - 1$. Applying the same arguments to cells r and s, we deduce that all cells a, b, c, d, e, f, g and h will be active at time $\tau - 2$ (as depicted in Figure 6.8c). Since the rule is freezing, we deduce that cells p, q, r and s are stable, obtaining that also u is stable.

From Equation 6.1, we know how to compute the states of cells b, d, f and h at time $\tau - 2$. To decide the stability of u, we need to compute the states of p, q, r and s at time $\tau - 1$. In this case, however, the dynamics in the corridors is more complicated. In the following, we will show how to compute the east corridor (in order to compute q), depicted in Figure 6.9. We will study only this case, since the other three corridors are analogous.

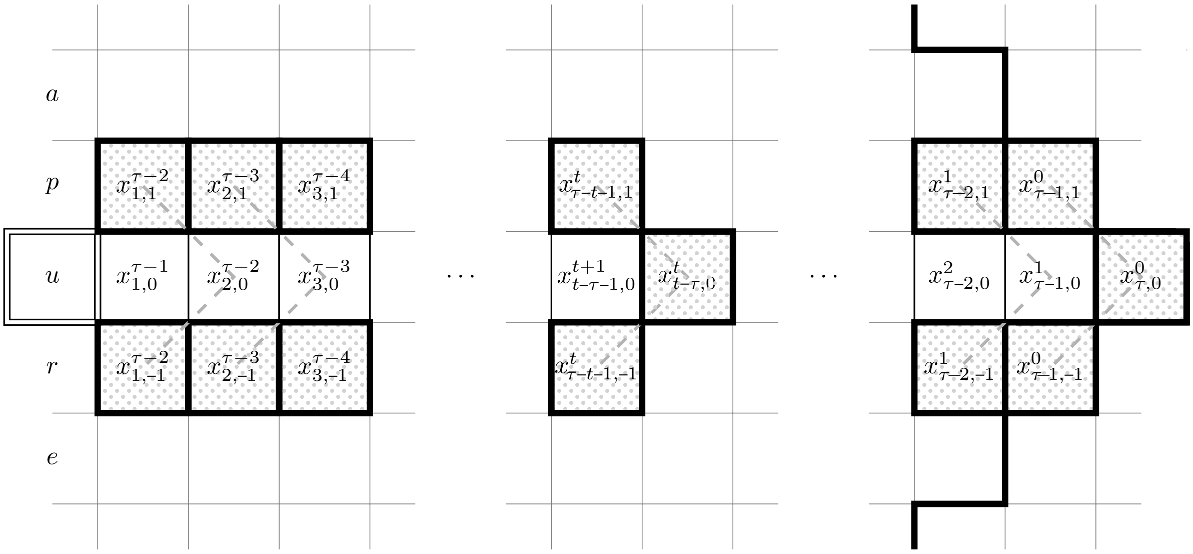


Figure 6.9: Computation of east corridor. Recall that $x^t_{(i,j)}$ is the state of cell (i,j) at time-step t. The gray cells were previously computed using the OR technique. Dashed lines connect cells which potentially change states at the same time.

Remember that, using Equation 6.1 we can compute the values of $x^{\tau-1-i}_{(i,1)}$ and $x^{\tau-1-i}_{(-i,1)}$, for every $i \in \{1, \dots, \tau-2\}$. Notice first that, if $x^{\tau-1-i}_{(i,1)} \neq x^{\tau-1-i}_{(-i,1)}$, then necessarily $x^{\tau-i}_{(i,0)} = 1$. Indeed, we know that $x^{\tau-i-1}_{(i-1,0)} = 0$ (otherwise we contradict the definition of τ). Then, $x^{\tau-1-i}_{(i,1)} \neq x^{\tau-1-i}_{(-i,1)}$ implies that at time $\tau-1-i$ the cell $(i,0)$ will have more than one and less than three active neighbors, so it will become active at time $\tau - i$.

Let i^* be the minimum value of $i \in \{1, \dots, \tau\}$ such that $x^{\tau-1-i}_{(i,1)} \neq x^{\tau-1-i}_{(-i,1)}$. If no such an i exists, then set $i^* = \tau$. Call I^* the set $\{1, \dots, i^*-1\}$. In other words, we know that $x^{\tau-1-i}_{(i,1)} = x^{\tau-1-i}_{(-i,1)}$ for every $i \in I^*$. Moreover, we also know that $x^{\tau-i^*}_{(i^*,0)} = 1$.

We now identify two situations, concerning the values of $x^{\tau-1-i}_{(i,1)}$, for $i \in I^*$.

If $x^{\tau-1-i}_{(i,1)} = x^{\tau-1-i}_{(-i,1)} = 0$ **then,** the value of $x^{\tau-i}_{(i,0)}$ will equal the value of $x^{\tau-1-i}_{(i+1,0)}$. Indeed, at time $\tau-1-i$, the cell $(i,0)$ will have three inactive neighbors ($(-i,1),(i,1),(i-1,0)$). Then it will take the same state than cell $(i+1,0)$ at time $\tau-i-1$.

If $x^{\tau-1-i}_{(i,1)} = x^{\tau-1-i}_{(-i,1)} = 1$ **then,** the value of $x^{\tau-i}_{(i,0)}$ will be the opposite than value of $x^{\tau-1-i}_{(i+1,0)}$. Indeed, at time $\tau-1-i$, the cell $(i,0)$ will have two active neighbors ($(-i,1),(i,1)$) and one inactive neighbor ($(i-1,0)$). Then cell $(i,0)$ is active at time $\tau-i$ if and only if cell $(i+1,0)$ is inactive at time $\tau-1-i$.

We imagine that a signal drives along the corridor. The signal starts at $(i^*,0)$ with value $x^{\tau-i^*}_{(i^*,1)}$. The movement of the signal satisfies that, each time it encounters an $i \in I^*$ such that $x^{\tau-1-i}_{(i,1)} = x^{\tau-1-i}_{(-i,1)} = 1$, the state switches to the opposite value. Let $z = |\{i \in I^* : x^{\tau-1-i}_{(i,1)} = x^{\tau-1-i}_{(-i,1)} = 1\}|$ (i.e. z is the number of switches). From the two situations explained above, we deduce the following lemma.

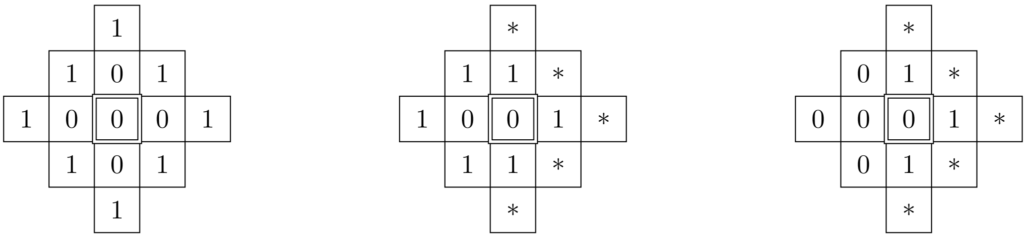
Lemma 6.3.9. *$x^{\tau-1}_{(1,0)}$ equals $x^{\tau-i^*}_{(i^*,1)}$ if z is even, and $x^{\tau-1}_{(1,0)}$ is different than $x^{\tau-i^*}_{(i^*,1)}$ when z is odd.*

Therefore, to solve Stability for rule 12, we compute the values of $x^{\tau-1}_p$, $x^{\tau-1}_q$, $x^{\tau-1}_r$, $x^{\tau-1}_s$ according to Lemma 6.3.9.

Solving Stability for rule 124

The analysis for the rule 124 is more complicated than the one we did for rule 12 and 123. In fact, one great difference is that **Claim 1** is no longer true for this rule. In other words, u might not be stable but change after time-step τ.

For this rule, the cases when the cell u remains inactive are the cases when u has zero or three active neighbors. The possible cases when the cell u remains inactive at time τ are given in Figure 6.10.



(a) Sum equal to 0.

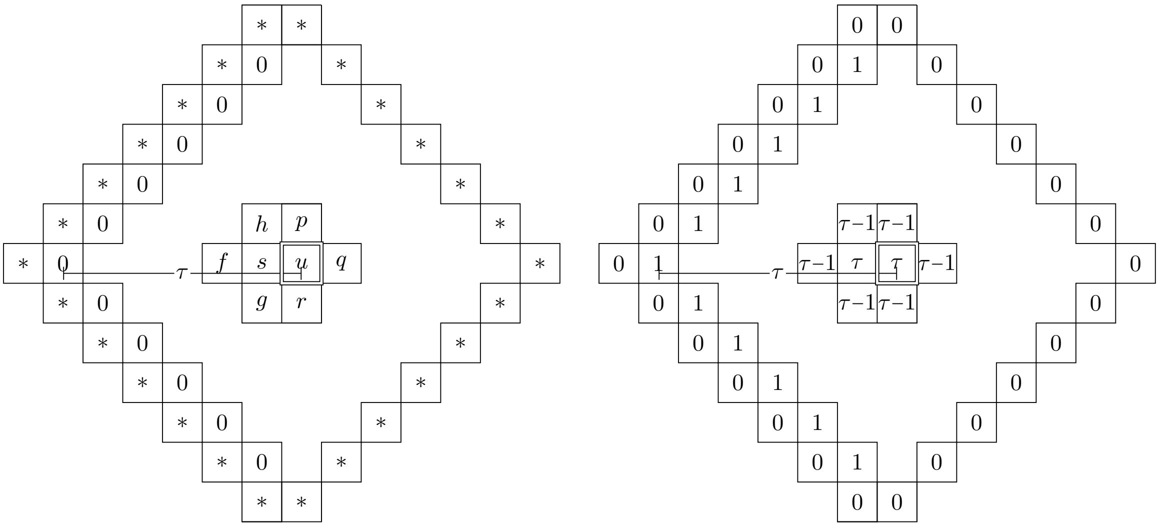
(b) Sum equal to 3.

(c) Sum equal to 3.

Figure 6.10: Possible cases of rule 124 at time τ such that u remains inactive.

The case when u has four inactive neighbors at time $\tau - 1$ is exactly the same that we explained for rule 12 (see Figure 6.10a). Suppose that u has three active neighbors, and without loss of generality assume that p, q, r are active and s is inactive. Then there are two possibilities, either s has three active neighbors (h, g, f), in which case u and s remain inactive (see Figure 6.10b) The difference with the rule 12 is that we can not decide immediately if the cell u remains inactive when the sum at time τ is 3. Indeed, in the case depicted in Figure 6.10b, it is possible that s becomes active in a time-step later than $\tau - 1$.

Thus we need to study only the case when the sum at time $\tau - 1$ of the states of neighbors of s is 0 or equivalently $x_f^{\tau-2} = x_g^{\tau-2} = x_h^{\tau-2} = 0$. Note that, by the OR technique, the fact that $x_f^{\tau-2} = x_g^{\tau-2} = x_h^{\tau-2} = 0$ means that all the cells in the left side border of D_τ are initially inactive, as shown in Figure 6.11.



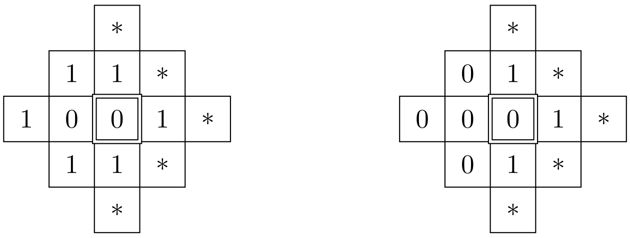
(a) Configuration at time 0.

(b) Diagram with the times when a cell can be activated.

Figure 6.11: Schedule for to compute x_u^τ if $x_f^{\tau-2} = x_g^{\tau-2} = x_h^{\tau-2} = 0$.

Knowing that $x_f^{\tau-2} = x_g^{\tau-2} = x_h^{\tau-2} = 0$, we can compute their states at time-step $\tau - 1$, considering the OR-techinique in the diamond $D_{\tau+1}$. Using this information, we can compute the state of s at time τ, i.e. compute x_s^{τ}.

Remember that we are in the case where $x_p^{\tau-1} = x_q^{\tau-1} = x_r^{\tau-1} = 1$ and $x_s^{\tau-1} = 0$. If the cell s becomes activated at time τ, then u will have four active neighbors at time τ, and it will become active. Now we suppose that s also remains inactive at time τ. Again, we have two possible cases:



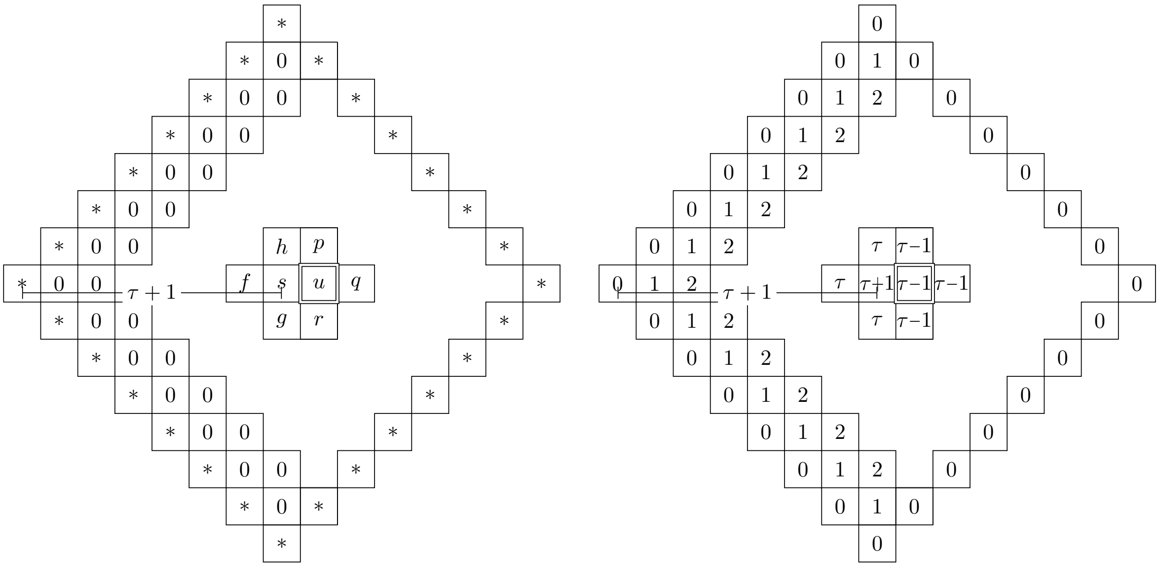
(a) Sum equal to 3.

(b) Sum equal to 3.

Figure 6.12: Possible cases of rule 124 at time $\tau + 1$ such that u remains at state 0.

In the case in Figure 6.12a (i.e., when s remains inactive at time τ because f, g and h were active at time $\tau - 1$) the cell u is stable.

For the case shown in Figure 6.12b (i.e. s remains inactive at time τ because e f, g and h were inactive at time $\tau - 1$) we must repeat the previous analysis. Indeed, we know that $x_f^{\tau-1} = x_g^{\tau-1} = x_h^{\tau-1} = 0$. The OR technique implies that every cell in the left border on the diamond $D_{\tau+1}$ (see Figure 6.11) have to be initially inactive too. In this case, however we study the next diamond $D_{\tau+1}$, shifting it one cell to the left, as the Figure 6.13.



(a) Configuration at time 0.

(b) Diagram with the times when a cell can be activated.

Figure 6.13: Schedule to compute x_s^{τ} if $x_f^{\tau-1} = x_g^{\tau-1} = x_h^{\tau-1} = 0$.

This new diamond is centered in the cell s and (considering only the sides at the north-west and south-west), consists 0f the sites at distance $\tau + 1$ from s. Again, using the OR technique, we can compute the states of cells f, g and h at time τ, and then the sate of cell s at time $\tau + 1$.

Again, if the cell s becomes active at time $\tau+1$, then the problem is solved, because u becomes active at time $\tau+2$. Further, suppose that s is not active at time $\tau+1$. Notice that this means that g, h and f must have the same state at time τ. Remember that p and r are active at time $\tau-1$, then cells h and f have at least one active neighbor at time $\tau-1$. If h, f and g are active at time τ, then s will be stable, as well as u. If h, f and g are inactive at time τ, it means that h and f have three active neighbors at time $\tau-1$, including cells $(-2,1)$ and $(-2,-1)$. Since these cells are also neighbors of g, and g remains inactive at time τ, necessarily cell $(-3,0)$ must be active at time $\tau-1$. This means that f, g and h have three active neighbors, so they are stable. Implying also that s and u are stable.

We deduce that either u becomes active at time $\tau, \tau+1$ or $\tau+2$, or u is stable.

Lemma 6.3.10. *Let F be rule* 124*. Given a finite configuration x, a cell u and τ the distance from u to the nearest active cell. Then u becomes active at time $\tau, \tau+1$ or $\tau+2$, or u is stable.*

Now we give an algorithm for to decide the STABILITY inn **NC** for the rule 124. The algorithms for rules 12 and 123 can be deduced from this algorithm.

Theorem 6.3.11. STABILITY *is in **NC** for the freezing CA* 12*,* 123 *and* 124*.*

Proof. Let (x, u) be an input of STABILITY, x is a finite configuration of dimensions $n \times n$ and u is a site in $[n] \times [n]$. The following parallel algorithm is able to decide STABILITY using the fast computation of the first neighbors of u by the OR technique. Let $N^2(u)$ be the set of cells at distance at most 2 from u. For $t \geq 0$, we call $x^t_{N(u)}$ the set of states at time t of all cells in $N(u)$.

Algorithm 8 Solving STABILITY 124

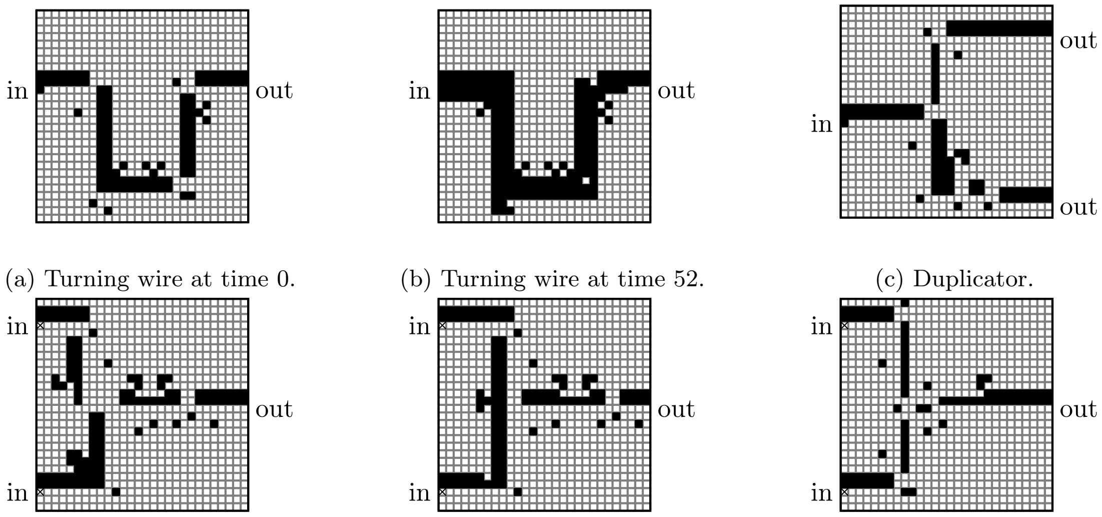
```
Input: $x$ a finite configuration of dimensions $n \times n$ and $u \in [n] \times [n]$.
 1: Compute $\tau$ the distance form $u$ to the nearest active cell in $x$.
 2: Compute $x^{\tau-2}_{N^2(u)}$ using the OR technique and the corridors.
 3: if $x^{\tau}_{u} = 1$ then
 4:     return Reject
 5: end if
 6: if $x^{\tau}_{u} = 0$ and $x^{\tau-2}_{N^2(u)}$ is as 6.10b then
 7:     return Accept
 8: end if
 9: Compute $s$ the neighbor of $u$ such that $x^{\tau-1}_{s} = 0$.
10: Compute $x^{\tau-1}_{N^2(s)}$ using the OR technique and the corridors.
11: if $x^{\tau+1}_{u} = 1$ then
12:     return Reject
13: end if
14: if $x^{\tau+1}_{u} = 0$ and $x^{\tau-1}_{N^2(u)}$ is as 6.10b then
15:     return Accept
16: end if
17: return Reject
```

Let $N = n^2$ the size of the input. Step **1** can be done in $\mathcal{O}(\log N)$ time with $\mathcal{O}(N)$ processors: one processor for each cell for to choose the active cells and to compute its distances to u, then in $\mathcal{O}(\log N)$ compute the nears cell to u. Steps **2** and **10** can be done in $\mathcal{O}(\log N)$ time with $\mathcal{O}(N)$ processors: the OR technique and the corridors can be computed with prefix sum algorithm (see proposition 3.2.4) for the computation of consecutive $\vee$ and parity of z in the corridors. The other steps can be computed in $\mathcal{O}(\log n)$ time in using a sequential algorithm. □

6.3.3 Turing Universal Rules

For the rules 2 and 24 the problem Stability is **P**-Complete by reducing a restricted version of the Circuit Value Problem [45] to this problem. Instances of circuit value problem are encoded into a configuration of the CA 2 and 24 using the idea in the proof of the **P**-completeness of Planar Circuit Value Problem (PCV) [68]. Moreover, we use an aproach given in [69], were the authors show that a two-dimensional automaton capable of simulating *wires*, *OR gates*, *AND gates* and *crossing gadgets* is **P**-Complete.

In Figure 6.14 we will give the gadgets that simulate these structures for rules 2 and 24. We remark that both rules have the same structures, because the patterns with four active neighbors never appear in the gadgets.



(a) Turning wire at time 0.

(b) Turning wire at time 52.

(c) Duplicator.

(d) AND gate.

(e) OR gate.

(f) XOR gate.

Figure 6.14: Gadgets for the implementation of logic circuits for the rules 2 and 24.

We represent the information traveling on wires which are based, roughly, on a line of active sites. Then, all the sites over (under) this line will have one active neighbor. If a cell over the line becomes active, then in the next step a neighbor of this cell will become active, so the information flows over the wire.

These constructions of the gates are quite standard. Maybe with one exception is the XOR gate. The crucial observation is that we manage to simulate the XOR using the synchronicity of information. An XOR gate consists roughly in two confluent wires. If a signal arrives from one of the two wires, the signal simply passes. If two signals arrive at the same time, the next cell in the wire will have more than two neighbors, so it will remain active.

Using the XOR gate (Figure 6.14f), one can build a planar crossing gadget, concluding the **P**-completeness constructions.

Theorem 6.3.12. Stability *is **P**-complete for the freezing CA* 2 *and* 24.

Remark: In our construction we use strongly neighborhoods composed only of inactive cells, so these constructions cannot be used for rules 02 and 024, where zero is not a quiescent state. So in these cases stability could have less complexity.

6.4 Rotation Invariant Rules

If we consider a more general family of CA where the rule is the same for neighborhoods equivalents for rotations and reflections, we obtain the same totalistic rules already studied for the triangular grid, but for the square grid we have two different cases for the neighborhoods with exactly two active neighbors:

- the cases when the two active cells are over a line (i.e. when north and south are active and its rotations) called I configurations,
- the cases when the two active cells are perpendicular (i.e. when north and east are active and its rotations) called L configurations.

In general, the complexity of a totalistic rule with number containing the digit 2 is the same as the rule exchanging the digit 2 by L in its number. Respectively the rules without the digit 2 in its number rule has the same complexity as the same rule adding I. The exception for this are the rule $I34$ and $L34$, both are analogous to 34 (see Theorem [39]) but forbidding the patterns L and I respectively in the set of stable cells.

another difference is that the rules I and $I4$, both with **NC** complexity, are not so trivially as ϕ and 4. Here is enough to study all the possibles configurations in the von Neumann neighborhood of radius 2 of the decision cell.

We have the same division of the rules: topological, algebraical, **P**-complete and fractal growing rules. This are:

Trivial: I and $I4$. It is enough to study the von Neumann neighborhood of radius 2 of the decision cell.

Topological: $I3$, $I34$, $L3$ and $L34$. We can use the same argument that for the rule 34, but checking that in the stable set of the rules with I do not appear patterns L and vice versa.

Algebraical: $1L$, $1L4$, $1L3$ and $1L34$. Given that the or technique (see figure 6.6a) use the diagonal to compute the new cells, the rules with 1 and L compute the cells in the next diagonal as the OR of its neighbors.

P-complete: L and $L4$. Is possible to build the same gadgets (figures 6.14a, 6.14c, 6.14d, 6.14e and 6.14f) for to obtain the **P**-completeness.

Fractal growing rules: $1I$, $1I4$, $1I3$ and $1I34$. This rule show a similar behavior to the rule 1. We think that solving one of this rules this will give clues for to solve the others ones.

6.4.1 Trivial Rules

We will show that it is enough to study the Von Neumann neighborhood of radius 2 to decide if a cell will change or not.

Note that the following pattern in figure 6.15

	*	*	
*	0	0	*
*	0	0	*
	*	*	

Figure 6.15: Stable pattern.

is stable, then we will study only the configurations without this pattern in the von Neumann neighborhood. These patterns are given in the following figure:

		*		
	1	*	1	
*	*	0	*	*
	1	*	1	
		*		

Case 1.

		*		
	1	*	*	
*	*	0	1	*
	1	*	*	
		*		

Case 2.

		*		
	1	*	*	
*	*	0	1	*
	*	1	*	
		*		

Case 3.

		*		
	*	*	*	
*	1	0	1	*
	*	*	*	
		*		

Case 4.

Figure 6.16: All possible Von Neumann neighborhood of radius 2, non equivalent by rotations or reflections and non containing the pattern with four 0 (figure 6.15).

Lemma 6.4.1. *Let $x \in \{0,1\}^{[n]\times[n]}$, $c = c(x)$, $u \in [n] \times [n]$ and F be the function of I. For all $t > 1, F^t(x)_u = F^2(x)_u$.*

Proof. It is enough to study the four previous cases.

Case 1:

		a_2		
	1	a_1	1	
b_2	b_1	0	c_1	c_2
	1	d_1	1	
		d_2		

We use $a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2 \in \mathbb{Z}^2$ to denote coordinates, not states.

We will consider only the non-equivalent, by symmetry, cases for $x_{a_1}, x_{b_1}, x_{c_1}, x_{d_1}, x_{a_2}, x_{b_2}, x_{c_2}, x_{d_2}$.

Sub-case 1.1 If $x_{a_1} = x_{d_1} = 1$ and $x_{b_1} = x_{c_1} = 0$, then $F(c)_z = 1$

Sub-case 1.2 If $x_{a_1} = x_{c_1} = 1$, then $\forall t \in \mathbb{N}, F^t(x)_z = 0$

Sub-case 1.3 If $x_{a_1} = x_{b_1} = x_{c_1} = 0$, then is enough to note that $F(c)_{a_1} = \neg F(c)_{a_2}$ and repeat the sub-cases 1.1 and 1.2 changing x by $F(c)$. If $F^2(x)_z = 0$ meaning that $F(c)_{a_1} = F(c)_{d_1} = 1$ or $a_2 = b_2 = c_2 = d_2 = 1$.

Case 2:

		e_1		
	1	c_1	d_1	
a	b	0	1	$*$
	1	c_2	d_2	
		e_2		

We use $a, b, c_1, c_2, d_1, d_2 \in [n] \times [n]$ for to denote coordinates, not states.

The only way to activate x_u is activating x_b to 1 and to keep $F(c)_{c_i} = 0$. To change x_b to 1 is necessary $x_a = 0$. To keep $F(c)_{c_i} = 0$ we have two ways: $e_i = 1$ or $d_i = e_i = 0$.

Case 3:

		$*$		
	1	$*$	$*$	
$*$	$*$	0	1	$*$
	$*$	1	$*$	
		$*$		

The cell u is a stable cell for any value in $*$.

Case 4:

		$*$		
	$*$	a	$*$	
$*$	1	0	1	$*$
	$*$	b	$*$	
		$*$		

We use $a, b \in \mathbb{Z}^2$ for to denote coordinates, not states.

The cell x_u is stable if $x_a = 1$ or $x_b = 1$, otherwise $F(c)_u = 1$.

□

Lemma 6.4.2. *Let F' the CA with rule $I4$, $x \in \{0,1\}^{[n]\times[n]}$ and $z \in [n] \times [n]$. For all $t > 1, F'^t(x)_u = F'^3(x)_u$.*

Proof. If $F^2(x)_z = 1$, then $F'^2(x)_z = 1$. If $F^2(x)_z = 0$, then is enough to know if all the neighbors of z are active after two iterations.

□

Theorem 6.4.3. Stability *is in* ***NC*** *for the freezing CA I and $I4$.*

Proof. Given the lemma 6.4.1 and 6.4.2 the following algorithm compute Stability for the rule I.

Algorithm 9 Solving Stability I, $I4$

Input: x a finite configuration of dimensions $n \times n$ and $z \in [n] \times [n]$ such that $x_u = 0$.

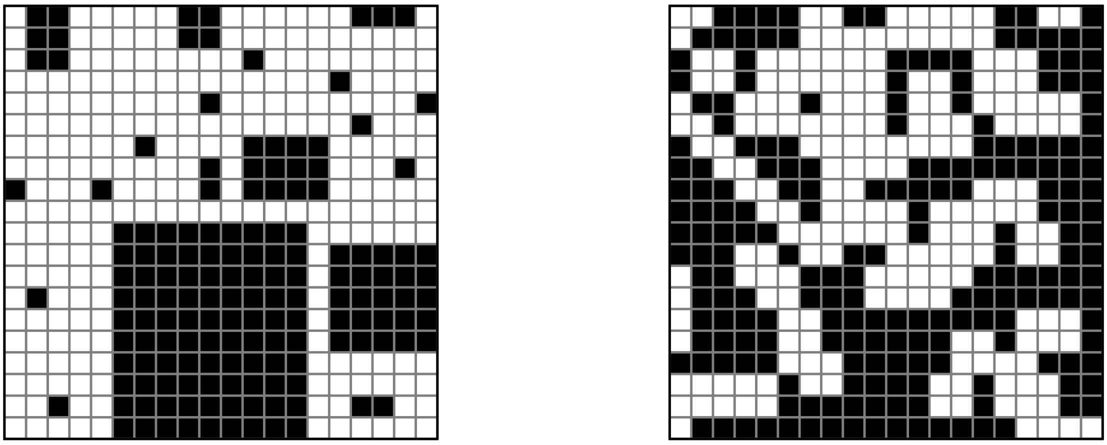
```
if $F^3(x)_u = 1$ then
    return yes
else
    return no
end if
```

□

6.4.2 Topological Rules

In [39] was showed that Stability for 34 are in **NC**. For this, was characterized the stable cells as the cells in a bi-connected component or in a paths between two bi-connected component on the graph of cells initially inactive, called $G[0]$.

We will show that the previous idea is valid for the rules $I34$ and $L34$ too. In our case it is not enough to consider the graph $G[0]$, it is necessary to add the geometrical conditions for this rules: in $I34$ if a 0 has two co-linear inactive neighbors it is not stable and if a 0 has two perpendiculars inactive neighbors it is not stable for $L34$. We will prove that considering this restriction on $G[0]$ the characterization given in [39] works also for the FCA $I34$ and $L34$.



(a) Fixed point for FCA $L34$.

(b) Fixed point for FCA $I34$.

Figure 6.17: ■ = 1. In $L34$ the stable cells are vertical or horizontal paths of inactive cells and in $I34$ are zig-zag paths of inactive cells.

Note that in $I34$ ($L34$) a cell is stable if it has at least two perpendicular (co-linear) neighbors stables.

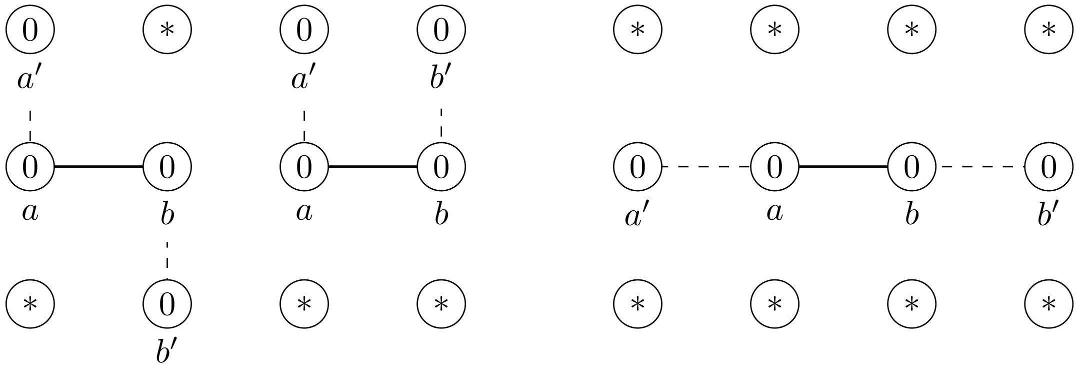
Lemma 6.4.4. *Let $x \in \{0,1\}^{[n]\times[n]}$ be a finite configuration, $u \in [n]\times[n]$ a site and F the FCA $I34$ ($L34$) then u is stable if and only if it is in a bi-connected component or in a path between two bi-connected components in $G = (V, E)$ defined by*

$$V = \{u \in x \subseteq [n]\times[n] : c_z = 0\}$$

$$E = \left\{ \{a,b\} \subseteq V^2 \middle| \begin{array}{lcl} \exists a' \in N(a)\setminus\{b\} : (a-a')\cdot(a-b) & = & 0 \\ \exists b' \in N(b)\setminus\{a\} : (b-b')\cdot(b-a) & = & 0 \end{array} \right\}$$

$$\left(\quad E = \left\{ \{a,b\} \in V^2 \middle| \begin{array}{lcl} \exists a' \in N(a)\setminus\{b\} : (a-a')\cdot(a-b) & \neq & 0 \\ \exists b' \in N(b)\setminus\{a\} : (b-b')\cdot(b-a) & \neq & 0 \end{array} \right\}\right).$$

Note that E is the set of edges in a zig-zag or “U” shape, see figure 6.18a (6.18b),(vertical or horizontal) path of cells inactive.



(a) Edges for $I34$.

(b) Edges for $L34$.

Figure 6.18: Different edges for $I34$ and $L34$. Continuous line represent the edge. Dashed lines are not necessary edges. $*$ can be 1 or 0.

Proof.
$\boxed{\Longrightarrow}$ Let $u \in [n]\times[n]$ a stable cell, then it must have two stable neighbors perpendicular (co-linear) $u_1 \neq u_{-1}$. Thus,

$$\exists u' = u'_{-1} \in N(u)\setminus\{u_1\} : (u-u')\cdot(u-u_1) = 0\ (\neq 0). \tag{6.3}$$

Since u_1 is also stable, then it also has two stable neighbors $u_1 \neq u_2$. Thus,

$$\exists u'_1 = u_2 \in N(u_1)\setminus\{u\} : (u_1-u'_1)\cdot(u_1-u) = 0\ (\neq 0). \tag{6.4}$$

From equations (6.3) and (6.4) you have that $\{u, u_1\} \in E$.

Given that u_1 is stable we can repeat this process and build an arbitrary long $(u-u_1-u_j)$-path in G, but V is finite, then we can chose j such that there is a i satisfying $u_i = u_j$. Then we have two options:

1. u is in the $(u_i - u_j)$-path, then u is in a cycle (bi-connected component) of G.
2. u is in the $(u - u_1 - u_i)$-path arriving to $(u_i - u_j)$-path.

We can repeat this analysis but starting by u_{-1} and building a path $u_{-1}...u_{-k}...u_{-l}$, where $u_{-k} = u_{-l}$. Then we obtain the same two options:

1’. u is in the $(u_{\text{-}k} - u_{\text{-}l})$-path, then u is in a cycle (bi-connected component) of G.

2'. u is in the $(u_{-1} - u_{-l})$-path arriving to $(u_{-k} - u_{-l})$-path.

If 1. or 1'. are true, then u is in a bi-connected component, if not then, by 2. and 2'., u is in a path between two bi-connected component.

$\boxed{\Longleftarrow}$ Let $c = c(x)$ and u in a bi-connected component or in a path between two bi-connected components of G called S. Note that for all vertex in S its neighbors are perpendicular (co-linear) and are inactive, then

$$F(c)|_S = c|_S \,.$$

We can repeat this over $F(c)$, obtaining

$$\forall t \in \mathbb{N} : F^t(c)\big|_S = c|_S \,.$$

in particular

$$F^t(c)_u = c_u = 0.$$

Thus u is a stable cell. □

Theorem 6.4.5. Stability *is in* ***NC*** *for the freezing CA $I34$, $I3$, $L3$ and $L34$.*

Proof. Note that G is a sub-graph of the graph $G[0]$ used in [39], then we can use the same algorithm for to find bi-connected components or paths between them on the rules $I34$, $I3$, $L3$ and $L34$. This algorithm is in NC.

□

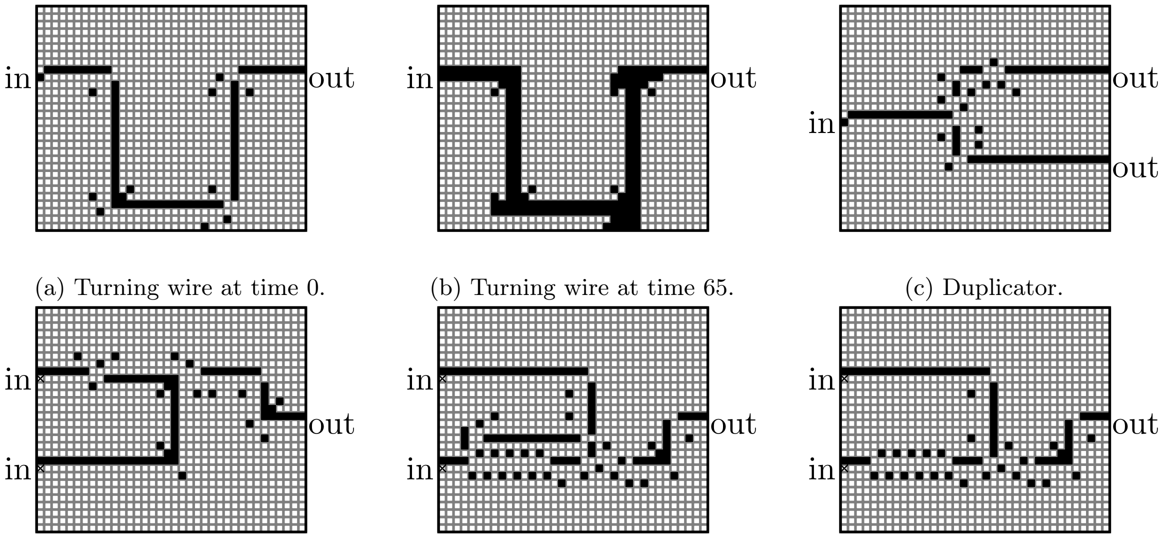
6.4.3 Algebraic Rules

Algebraic rules are the rules $1L$, $1L3$ and $1L4$ and they are analogous to 12, 123 and 124, but it is necessary to consider the fact that $I \notin \mathcal{I}$. This modifies the computation of the corridors, the cells $(\pm t, 0)$ and $(0, \pm t)$, where $0 < t < \tau$ in figure 6.9. Now the cases when the cells $(\pm t, 0)$ and $(0, \pm t)$ are in state 1 do not generate a signal.

Theorem 6.4.6. Stability *is in* ***NC*** *for the freezing CA $1L$, $1L3$ and $1L4$.*

6.4.4 Turing Universal Rules

Replacing the figure 6.14 by figure 6.19 we can build the same circuits that the cases 2 and 24, then Stability is P-complete also for the rules L and $L4$.



(a) Turning wire at time 0.

(b) Turning wire at time 65.

(c) Duplicator.

(d) AND gate.

(e) OR gate.

(f) XOR gate.

Figure 6.19: Gadgets for the implementation of logic circuits for the rules L and $L4$.

Theorem 6.4.7. Stability *is **P**-complete for L and $L4$.*

6.5 Reachability

The complexity of Stability remains open for rules such that a cell is may be activated with exactly one active neighbor (i.e., rules 1, 13, 14 and 134).

It is important to point out that if we have a fast algorithm to solve Stability then, we may apply it, given an initial configuration, to determine the fixed point reached faster than the automaton simulation.

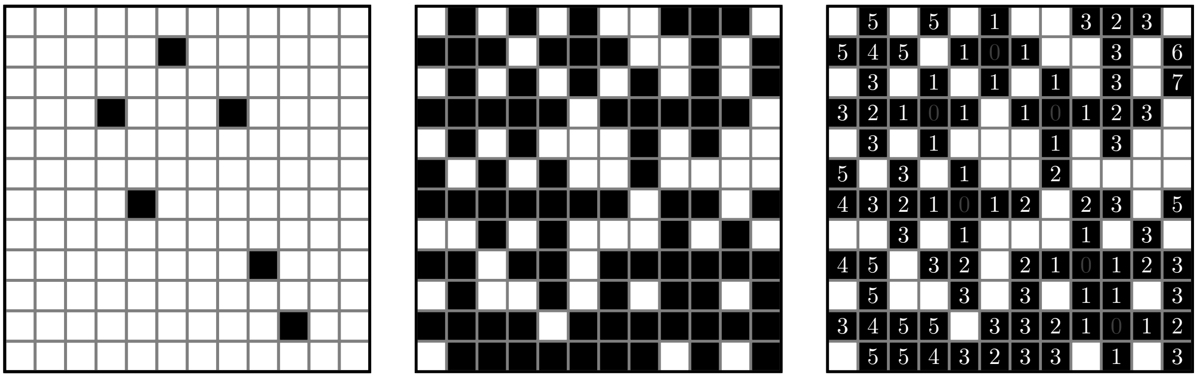
The following definitions considering a configuration x of dimensions $n \times n$, $z \in [n] \times [n]$ and $u, v \in \mathbb{1}(x)$ defined as follow:

- $\mathbb{1}(x) = \{v \in [n] \times [n] : x_v = 1\}$, the set of active cells in x.
- $d_x(u, v)$ is the length of the shortest path in $G[\mathbb{1}(x)]$. If u and v are disconnected then $d_x(u, v) = \infty$. (G, d) is a metric space.
- $d_x(v, S) = \min\limits_{s \in S} d_x(v, s)$.

Clearly, if we determine a fast algorithm for Stability then we directly may apply it for Reachability. We do not have that algorithm, but we will show that Reachability in **NC** for the rule 1.

Theorem 6.5.1. *There is a **NC** algorithm for verifying if a configuration is a fixed point obtained from another configuration, i.e.* Reachability *is in **NC**.*

Note that, given that each cell is activated with exactly one neighbor, the new active cells in $F^t(x)$ are product of a single cell in x. We call this seminal cell its *seed*.



(a) Initial configuration x.

(b) Configuration $F^6(x)$ (fixed point).

(c) Distances to $\mathbb{1}(x)$ in $F^6(x)$.

Figure 6.20: Examples of the previous definitions. The black cells are active or equivalently are in $\mathbb{1}(x)$ and $\mathbb{1}(F^6(x))$. In (c) the number meaning the distance from the cell to the cells initially active i.e. to $\mathbb{1}(x)$, then the cells in $\mathbb{1}(x)$ have the number 0. Note that this number coincides with the iteration number in which the cell is activated.

We will show a **characterization** of the images of a initial configuration using our metric showed in the figure 6.20c. After this, we will build a **NC** algorithm to decide if a configuration has this characterization.

Definition 6.5.1. Let x and x^t configurations of dimensions $n \times n$ such that $F^t(x) = x^t$, $v \in \mathbb{1}(x)$ and $z \in \mathbb{1}(x^t) \setminus \mathbb{1}(x)$.

- The cell v is *seed* of $z \in N(v)$ if $x_z = 0 \wedge F(x)_z = 1$.
- The cell v is *seed* of $z \in [n] \times [n]$ if there is a time t such that $F^{t-1}(x)_z = 0 \wedge F^t(x)_z = 1$ and the unique active neighbor of z at time $t-1$ has v as its seed.

Note that if an active cell has a seed, it is the only one. Moreover, if a cell is activated by the action of another cell, both share the same seed and the distance to the seed is 1 plus the distance from the neighbor to the seed.

We say that an inactive cell at time t is *blocked* if this cell has two or more active cells at this time. A blocked cell is stable.

Lemma 6.5.2. *Necessary conditions for $F^t(x) = x^t$ are:*

- $\mathbb{1}(x) \subseteq \mathbb{1}(x^t)$
- $\max\limits_{z \in \mathbb{1}(x^t)} d(z, \mathbb{1}(x)) \leq t.$

Corollary 6.5.2.1. *The previous lemma is valid also if $t = \infty$, it means that it is valid for the fixed point reached starting by x.*

Proof. Is enough to take $t = n^2$, because x has at most n^2 inactive cells, then, in the worst case, we need to active all the cells one by one until to activate everyone. This takes n^2 steps. □

Lemma 6.5.3. *Let x, y^T two configurations of dimensions $n \times n$ satisfying the lemma 6.5.2, where $T = \max\limits_{z \in \mathbb{1}(y^T)} d_{y^T}(z, \mathbb{1}(x))$ and for all $t = 0, ..., T$*

$$y^t_z = \begin{cases} 0 & \textit{if } d_{y^T}(z, \mathbb{1}(x)) > t \\ y^T_z & \textit{otherwise} \end{cases}.$$

Then $y^T = F^T(x)$ *if and only if*

$$\forall t = 1, ..., T, \forall z \in \mathbb{1}(y^t) \setminus \mathbb{1}(y^{t-1}) : y_z^t = 1 \Leftrightarrow |N(z) \cap \mathbb{1}(y^{t-1})| = 1 \tag{6.5}$$

Note that in the previous lemma,

1. $y^0 = x$,
2. $\mathbb{1}(x) \subseteq \mathbb{1}(y^1) \subseteq ... \subseteq \mathbb{1}(y^T)$.

Proof. By induction over T. Let $T = 1$. We will call y to y^1, then we must prove that $y = F(x)$ if and only if $\forall z \in \mathbb{1}(y) \setminus \mathbb{1}(x) : y_z = 1 \Leftrightarrow |N(z) \cap \mathbb{1}(y)| = 1$.

$\boxed{\Leftarrow}$ It is true because every new active cell in $y = F(x)$ comes from a single active cell in $\mathbb{1}(x)$.

$\boxed{\Rightarrow}$ It is enough to study the cells in $\mathbb{1}(y) \setminus \mathbb{1}(x)$ because $\mathbb{1}(x) \subseteq \mathbb{1}(y)$. Let $z \in \mathbb{1}(y) \setminus \mathbb{1}(x)$

If $y_z = 1$ this is equivalent to $|N(z) \cap \mathbb{1}(x)| = 1$, then, by definition of F, $y_z = F(x)_z = 1$.

If $y_z = 0$ we have two possibles cases:

Case $|N(z) \cap \mathbb{1}(x)| = 0$**.** In this case z has no active neighbors, then $y_z = F(x)_z = 0$.

Case $|N(z) \cap \mathbb{1}(x)| \geq 2$**.** Here z blocked, then $y_z = F(x)_z = 0$.

Then $\forall z \in [n] \times [n] : y_z = F(x)_z$.

Now we suppose that the induction hypothesis is true, i.e.

$$y^T = F^T(x) \text{ if and only if } \forall t = 1, ..., T, \forall z \in \mathbb{1}(y^t) \setminus \mathbb{1}(y^{t-1}) : y_z^t = 1 \Leftrightarrow |N(z) \cap \mathbb{1}(y^{t-1})| = 1.$$

We take the case $T + 1$, i.e. we must prove

$$y^{T+1} = F^{T+1}(x) \text{ if and only if } \forall t = 1, ..., T+1, \underbrace{\forall z \in \mathbb{1}(y^t) \setminus \mathbb{1}(y^{t-1}) : y_z^t = 1 \Leftrightarrow |N(z) \cap \mathbb{1}(y^{t-1})| = 1}_{(*)}.$$

By induction hypothesis (*) true for $t = 1, ..., T$ is equivalent to $y^T = F^T(x)$, then we need to prove

$$y^{T+1} = F(y^T) \text{ if and only if } \forall z \in \mathbb{1}(y^{T+1}) \setminus \mathbb{1}(y^T) : y_z^{T+1} = 1 \Leftrightarrow |N(z) \cap \mathbb{1}(y^T)| = 1.$$

This is true taking $T = 1$, $x = y^T$ and $y = y^{T+1}$ at the beginning of this demonstration. □

We are ready to prove that Reachability is in **NC**.

Theorem 6.5.1 . The following algorithm verifies if the configuration y^∞ is a fixed point obtained form x. For this, we check that y^∞ is a fixed point and then we check the lemma 6.5.3 for the time when the image of x become a fixed point.

Algorithm 10 Solving Reachability 1

```
Input: $x$ and $y^\infty$ a finites configurations of dimensions $n \times n$.
 1: VERIFY($x$,$y^\infty$)
 2: $D \leftarrow$ DISTANCE($x$,$y^\infty$)                          % $D(u,v) = d_T(u,v)$
 3: $D1 \leftarrow$ DISTANCE1($x$,$y^\infty$,$D$)                    % $D1(u) = d_T(u,\mathbb{1}(x))$
 4: for all $z \in [n] \times [n] \setminus \mathbb{1}(x)$ do in parallel
 5:   if $y^\infty_z = 1$ and $\sum_{\substack{z' \in N(z) \\ D1(z')=D1(z)-1}} y^\infty_{z'} = 1$ then
 6:     return Reject
 7:   else if $y^\infty_z = 0$ and $\sum_{\substack{z' \in N(z) \\ D1(z')=D1(z)-1}} y^\infty_{z'} \neq 1$ then
 8:     return Reject
 9:   end if
10: end for
11: return Accept
```

This algorithm works as follow:

Step 1: VERIFY checks that x and y^∞ satisfies $\mathbb{1}(x) \subseteq \mathbb{1}(y^\infty)$ and that y^∞ is a fixed point.

Step 2: DISTANCE computes a matrix D, where $D(z,v)$ is the distance between z and v over the graph induced by $\mathbb{1}(y^\infty)$. It can be done in time $\mathcal{O}(\log^2 N)$ using $\mathcal{O}(N^2)$ processors using the adjacency matrix and its products computed by the prefix-sum algorithm given in 3.2.4.

Step 3: DISTANCE1 computes a vector $D1$, where $D(z)$ is the distance between z and $\mathbb{1}(x)$ over the graph induced by $\mathbb{1}(y^\infty)$. It can be done in time $\mathcal{O}(\log^2 N)$ using $\mathcal{O}(N^2)$ processors using the adjacency matrix and its products computed by the prefix-sum algorithm given in 3.2.4.

Steps 5-6: Verifies that z satisfies the characterization of active cells in Lemma 6.5.3. If it does not, it answers *Reject*. There is a constant number of operations, then it can be done in $\mathcal{O}(\log N)$ time in a sequential machine.

Steps 7-8: Verifies that z satisfies the characterization of inactive cells in Lemma 6.5.3. If it does not, it answers *Reject*. There is a constant number of operations, then it can be done in $\mathcal{O}(\log N)$ time in a sequential machine.

Step 11: *Accepts* if all the cells satisfies Lemma 6.5.3.

□

6.6 Concluding Remarks

6.6.1 Summary of our results

In this chapter we have studied the complexity of the Stability problem for the Freezing Totalistic Cellular Automata (FTCA) on the triangular and square grid with von Neumann neighborhood and two states. We find different complexities for these FTCA, including a **P**-complete case on square grid. For the rules where Stability is in **NC** we have considered two approaches: a topological approach (theorems 6.2.3, 6.2.1, 6.3.1, 6.3.2, and 6.3.3) and an algebraic approach (theorems 6.2.6 and 6.3.11).

Rule	Stability	Theorem
ϕ	$\mathcal{O}(1)$	Trivial
3	**NC**	Trivial
2	**NC**	Thm 6.2.3
23	**NC**	Thm 6.2.1
12	**NC**	Thm 6.2.6
123	**NC**	Trivial

Table 6.1: Summary of rules and their complexity of Stability on triangular grid.

Rule	Stability	Theorem	Rule	Stability	Theorem
4	**NC**	Trivial	234	**NC**	Thm 6.3.3
3	**NC**	Thm 6.3.2	12	**NC**	Thm 6.3.11
34	**NC**	Thm 6.3.1	124	**NC**	Thm 6.3.11
2	**P**-Complete	Thm 6.3.12	123	**NC**	Thm 6.3.11
24	**P**-Complete	Thm 6.3.12	1234	$\mathcal{O}(1)$	Trivial

Table 6.2: Summary of rules and their complexity of Stability on square grid.

Also we study the cases when the rules are not totalistic, but they are invariant by rotation and reflection, obtaining the following table:

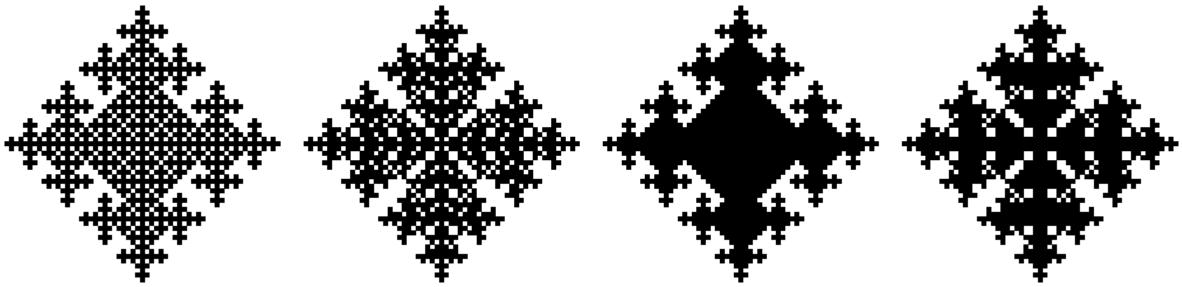
Rule	Stability	Theorem	Rule	Stability	Theorem
I	$\mathcal{O}(1)$	Trivial	$L3$	**NC**	Thm 6.4.5
$I4$	$\mathcal{O}(1)$	Trivial	$L34$	**NC**	Thm 6.4.5
$I3$	**NC**	Thm 6.4.5	$1L$	**NC**	Thm 6.4.6
$I34$	**NC**	Thm 6.4.5	$1L4$	**NC**	Thm 6.4.6
L	**P**-Complete	Thm 6.4.7	$1L3$	**NC**	Thm 6.4.6
$L4$	**P**-Complete	Thm 6.4.7	$1L34$	**NC**	Thm 6.4.6

Table 6.3: Summary of invariant by rotation rules and their complexity of Stability on square grid.

6.6.2 About Fractal-Growing Rules

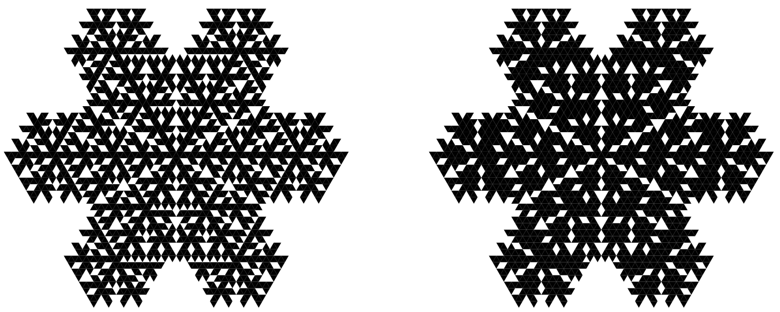
In this chapter we have not included a study of fractal growing rules. In fact, the complexity of Stability remains open for these rules, even for fractal rules defined over a triangular grid.

To obtain an intuition about the dynamical complexity of those rules, see Figures 6.21 and 6.22, where starting with only the center active we obtain a fractal behavior.



(a) Rule 1 (b) Rule 13 (c) Rule 14 (d) Rule 134

Figure 6.21: Examples of different rules with the similar fractal dynamics starting with a single active cell on square grid.



(a) Rule 1 (b) Rule 13

Figure 6.22: Examples of different rules with the similar fractal dynamics starting with a single active cell on triangular grid.

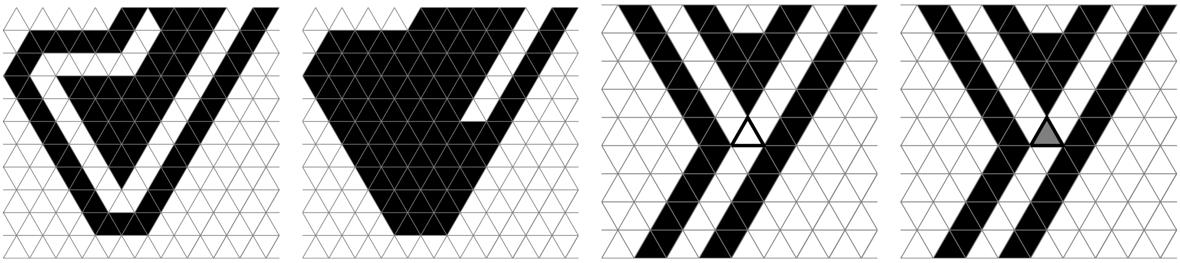
It is important to remark that non-freezing version of rule 13 is the usual XOR between the four neighbors, which is a linear cellular automaton. Using a prefix-sum algorithm, we can compute any step of a linear cellular automaton, so the *non-freezing* rule 13 is in **NC**. Although we might imagine that adding *freezing* property simplifies the dynamics of a rule, the non-linearity of rule 13 increases enough the difficulty to prevent us to characterize its complexity.

6.6.3 On P-Completeness on the triangular grid

It is important to point out that for triangular grid (despite rule 1 or 13 might be candidates), we do not exhibit a rule such that Stability is **P**-complete. The reader might think that, like in the one-dimensional case, every freezing rule defined in a triangular grid is **NC**. This is not the case. Moreover, three states (that we call 0, 1, 2) suffice to define a **P**-complete FTCA. The general freezing property means that states may only grow (so, in this case state 2 is stable). In this context, for a triangular grid, consider the following the local function.

$$f\left(x_u, \sum_{z \in N+u} x_z\right) = \begin{cases} 1, & \text{if } x_u = 0 \land (\sum_{z\in N(u)} x_z = 2 \lor \sum_{z \in N(u)} x_z = 12) \\ 1, & \text{if } x_u = 10 \land \sum_{z \in N(u)} x_z = 11 \\ x_u, & \text{otherwise} \end{cases}$$

The proof of **P**-Completeness follows similar arguments than the ones we used for rules 2 and 24 in the squared grid (Theorem 6.3.12), i.e. reducing the Circuit Value Problem (CVP) to Stability on this rule. Instances of CVP are encoded into a configuration of this FTCA using the idea in the construction of the logical gates. In figures 6.14d and 6.23d we exhibit the gadgets.



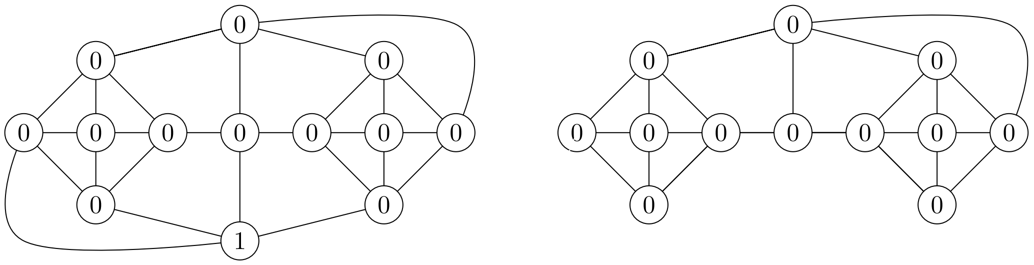
(a) Wire at time 0. (b) Wire at time 30. (c) AND gate. (d) XOR gate.

Figure 6.23: Gadgets for the implementation of logic circuits. The thick line marks the cell that makes the calculation from signals. The color code is: ▲ : 1, ▲ : 10 and △ : 0

6.6.4 Limitation on the complexity for the rule 234

We showed that the Stability problem for the two-dimensional freezing non-strict majority automaton is in **NC**. This question was posed in [31] and [39].

In [39] it is shown that the prediction problem for freezing non-strict majority automaton on an arbitrary graph of degree at most four is **P**-complete, and in graphs of degree at most three is in **NC**. The authors conjectured that this problem is in **NC** on any planar topology. We remark that if we remove the hypothesis that the topology is a grid, then our characterization of stable sets (Theorem 6.3.8) is no longer true, even for planar regular tri-connected graphs of degree four, as we can see in Figure 6.24.



(a) Initial configuration. (b) Graph obtained after to remove the active cells.

Figure 6.24: Non-tri-connected graph of inactive cells, but stable.

This opens a way to study for a characterization of the graphs in which the theorem is true.

6.6.5 About non-quiescent rules

Finally, it is convenient to say a word about rules where cells become active with zero active neighbors, i.e., rules where state 0 is not quiescent. Clearly, after one step for those rules, every cell will have at least one active neighbor. Then, their complexity is at most the complexity of the same rule, not considering the case of zero active neighbors as an activating state. For example, consider rule 034 in the square grid. After one step of rule 034, the dynamics are exactly the same that the one of rule 34. Therefore, rule 034 is in **NC**. Although, there are some interesting cases. First, notice that rule 01 is trivial (in the triangular or square grids), because after only one step the rule reaches a fixed point. This contrasts with rule 1, which is a Fractal-Growin rule. Second, consider rule 02 or 024 in the square grid. We know that rule 2 and 24 are **P**-Complete. However, the reader can verify that the gadgets used to reduce CVP to Stability do not work for rule 02 and 024. This fact opens the possibilty that rules 02 and 024 belong to **NC**.

Chapter 7

Fast-Parallel Algorithms for Freezing Totalistic Asynchronous Cellular Automata

In this chapter, we relax the synchronizing hypothesis of cellular automata and consider *freezing asynchronous cellular automata* (FACA), where cells evolve one by one, following a predefined order called *updating scheme.*

An active research topic in the context of the study of CA, is the prediction problem, i.e. anticipate the future state of a cell given an initial configuration. In the context of ACA, this problem can be translated into finding a sequential updating scheme that changes the state of a cell. Our objective is to study the prediction in the context of the *Computational Complexity Theory.* More precisely, our objective is to classify the prediction problem of a ACA (CA) in one of the following clases: **P** of problems solvable in polynomial time on a deterministic Turing machine; **NC** of problems that can be solved by a fast-parallel algorithm, and **NP** the problems that can be solved in polynomial time in a non-determinstic Turing machine. It is known that $\mathbf{NC} \subseteq \mathbf{P} \subseteq \mathbf{NP}$, and the prediction problem is at most in **NP**. It is a wide-believed conjecture that the inclusion are proper, then this classification give us an idea of the complexity of these automata.

In particular we are interested in studding freezing ACA. It is direct that, in freezing ACA, every initial configuration consisting of N cells reaches a fixed point in $\mathcal{O}(N^2)$ steps, on every updating scheme. However, there are cells that remain inactive regardless of the chosen updating scheme. These cells are called *stable.* We call AsyncStability the problem of deciding, given an initial configuration, if a given cell is stable on any updating scheme.

The *freezing majority cellular automaton*, also known as *bootstrap percolation model* [39] was studied in arbitrary undirected graph. In this case, an inactive cell becomes active if and only if the active cells are the most represented in its neighborhood. In that chapter, it is shown that in these ACA any updating scheme converges at the same fixed point and it was proved that AsyncStability is **P**-Complete over graphs such that its maximum degree (number of neighbors) ≥ 5. Otherwise (graphs with maximum degree ≤ 4), the problem is in **NC**. This clearly includes the two dimensional grid with von Neumann neighborhood.

In this chapter, we consider a family of two-state two dimensional *Freezing Totalistic Asynchronous Cellular Automaton* (FTACA) defined on a triangular and square grid.

We show that for every graph in this family the problem AsyncStability is in **NC**. We show this result following two approaches:

- Infiltration approach: FTACA where there is a connected set S of inactive cells such that, if any set in the perimeter of S becomes active, then for every cell in the connected set there is an updating scheme that activates it (we say that the set was infiltrated).
- Monotone approach: We use a result of [39] that relates the behavior of monotone rules on asynchronous updating schemes with respect to the same rules in synchronous updating schemes.

This chapter is structured as follows: First, in Section 7.1, we give the main definitions and notations. In Sections 7.2 and 7.3 we study the complexity of AsyncStability in the triangular grid and the square grid respectively. In Sections 7.2.1 and 7.3.1 we study the complexity of FTACA using the infiltration approach in the triangular grid and the square grid respectively. In Sections 7.2.2 and 7.3.2, we study the complexity of FTACA using the monotone approach in the triangular grid and the square grid respectively. Finally, in Section 7.6 we give some conclusions.

Part of the content of this chapter correspond to publication *Fast-Parallel Algorithms for Freezing Totalistic Asynchronous Cellular Automata* [4].

7.1 Definitions

We will study the family of two-state freezing totalistic asynchronous cellular automata (FTACA), over the triangular or square grids, with von Neumann neighborhood. In this family, the active cells remain active, because the rule is freezing, and the inactive cells become active depending only in the sum of their neighbors.

Let F be a FTACA. We can identify F with a set $\mathcal{I}_F \subseteq \{0,1,2,3\}$ for the triangular grid and $\mathcal{I}_F \subseteq \{0,1,2,3,4\}$ for the square grid such that, for every configuration c and site u:

$$f(c_{N(u)}) = \begin{cases} 1 & \text{if } (c_u = 1) \vee (\sum_{v \in N(u)\setminus\{u\}} c_v \in \mathcal{I}_F), \\ 0 & \text{otherwise.} \end{cases}$$

We will name the FTACA according to the elements contained in $\mathcal{I}_F$, as the concatenation of the elements of $\mathcal{I}_F$ in increasing order (except when $\mathcal{I}_F = \emptyset$, that we call ϕ). For example, let Maj be the freezing majority vote CA, where an inactive cell becomes active if the majority of its neighbors is active. Note that Freezing Majority-Vote Cellular Automata is the rule 23 in this notation for triangular grid and for square grid Freezing (non) Strict Majority-Vote Cellular Automata is the rule 34 (234).

We deduce that there are 2^4 different FTCA for the triangular grid and 2^5 different for square grid, each of them represented by the corresponding set $\mathcal{I}_F$. We will focus our analysis on the FTACA where the inactive state is a quiescent state, which means that the inactive sites where the sum of their neighborhoods is 0 remain inactive. Therefore, we will consider initially only 8 different FTACA.

7.2 Triangular grid

Here we study the complexity of AsyncStability for the FTACA on a triangular grid, obtaining that in all these rules this problem is in **NC**. We grouped these rules according the strategy used to prove its complexity: 1, 12 and 13 we use the infiltration technique; 2, 23 we use the monotony and the rules ϕ, 3 and 123 are trivial.

7.2.1 The infiltration technique

In this section we study the rules where an inactive cell becomes active with one active neighbor. This cases include the rules 1, 12 and 13. For all these rules we define, for an initial configuration x, V_{+1} the set of all cells in $T(n)$ that *need exactly one active neighbor to be activated*, formally

$$V_{+1} = \{v \in [n] \times [n] : x_v = 0 \ \wedge \sum_{w \in N(v)} x_w + 1 \in \mathcal{I}_{\mathcal{F}}\}.$$

We define G_{+1} as the graph induced by V_{+1}. Also we define B_{+1}, called *boundary* of G_{+1}, as the cells in the complement of V_{+1} with at least one neighbor in V_{+1}, formally

$$B_{+1} = \{v \notin V_{+1} : V_{+1} \cap N(v) \neq \emptyset\}.$$

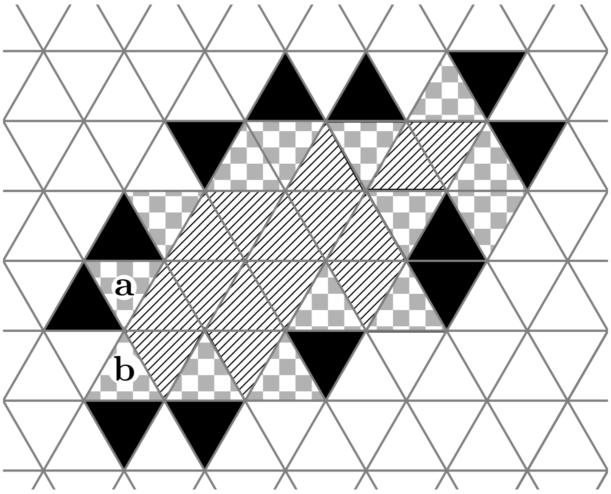


Figure 7.1: ▨: Cells in V_{+1}. ▩: Cells in B_{+1}. ■: Active cells . Example of V_{+1} and B_{+1} for rule 1. The cell (a) is not in V_{+1} because it can not evolve. The cell (b) is not in V_{+1} because it evolves immediately.

Without loss of generality, we suppose that G_{+1} is connected and contains u (otherwise, we restrict to the connected component of G_{+1} containing u). The following lemma explains what happens if a cell infiltrate the border.

Lemma 7.2.1. *If there is an updating scheme such that some cell in B_+ becomes activated, then there is a updating scheme activating u.*

Proof. Roughly, if a boundary cell v becomes active (it infiltrates V_{+1}), then, by connectivity, we choose the shortest (v, u)-path of cells in V_+. Then, choosing an updating scheme activating the cells of the $(v - u)$-path one by one from v we activate u. □

Now we will see that it is enough to check the information of the neighborhood of each border cell to know if this cell is stable or it infiltrates V_{+1}. We are not interested in the cells that become active with three active neighbors, because this cells can belong to B_+ and also they cannot affect its neighbors, because every one of this is already active and can not evolve.

Lemma 7.2.2. *To decide if a boundary cell $v \in B_+$ will become active or it is stable depends only of $N(v)$.*

Proof. Let $v \in B_{+1}$. We will consider the following facts:

- By definition of B_{+1}, v has at least one inactive neighbor, the neighbor in V_{+1}.
- Given that $v \notin V_{+1}$, then it has at least one active neighbor, otherwise $v \in V_{+1}$.
- If v has exactly one active neighbor, then v evolves in one time-step and we are done.
- If v has exactly two active neighbors, then v evolves in one iteration and we decide (because $2 \in \mathcal{I}_F$) or it does not evolve, then v is stable and we decide (because $2 \notin \mathcal{I}_F$).

We deduce the lemma.

□

Theorem 7.2.3. *AsyncStability is in* ***NC*** *for the rules* 1, 12 *and* 13.

Proof. Let (x, u) be an input of AsyncStability, i.e. x is a finite configuration of dimensions $n \times n$, and u is a site in $[n] \times [n]$. Our algorithm for AsyncStability first check if the neighborhood of u is a stable pattern or can evolve in one step, then computes V_{+1} and G_{+1}. Then, the algorithm computes the connected components of G_{+1} and restricts G_{+1} to the connected component containing u and then compute B_{+1}. Finally, the algorithm answers *Reject* if there is a vertex $v \in B_{+1}$ that can be activated. Otherwise answer *Accept.*

This algorithm works too on the rules changing with three active neighbors, because to activate a boundary cells with three neighbors implies to have a active cell in V_{+}.

Algorithm 11 AsyncStability solving 1, 12 and 13

```
Input: $x \in \{0,1\}^{T(n)}$ and $u \in T(n)$ such that $x_u = 0$.
 1: if $N(u)$ is a stable pattern then
 2:     return Accept
 3: end if
 4: if $f(x_{N(u)}) = 1$ then
 5:     return Reject
 6: end if
 7: Compute the $V_{+1} = \{v \in \mathbb{Z}^2 : x_v = 0 \ \wedge \ |x_{N(v)}|_1 + 1 \in \mathcal{I}_F\}$.
 8: Compute the graph $G_{+1} = G[V_{+1}]$.
 9: Compute the connected components of $G_{+1}$, $\{C_i\}_{i=1}^{M}$ .
10: Redefine $V_{+1} = C_i : u \in C_i$.
11: Compute the $B_{+1} = \{v \notin V_{+1} : V_{+1} \cap N(v) \neq \emptyset\}$.
12: for all $v \in B_{+1}$ do in parallel
13:     if $f(x_{N(v)}) = 1$ then
14:         return Reject
15:     end if
16: end for
17: return Accept
```

Let $N = n^2$ the size of the input.

Steps 1-6 are computed easily in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors.

Step 7 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, 1 processor by cell $v \in [n] \times [n]$ and it tests that $|x_{N(v)}|_1 + 1 \in \mathcal{I}_F$.

Step 8 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, 1 processor by edge (u, v) in the grid (there is $\mathcal{O}(N)$ edges) and it add (u, v) to the edges of G_{+1} if u and v are in V_{+1}.

Step 9 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors by proposition 3.2.5.

Step 10 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, 1 processor by cell v in each connected component, it tests that $v = u$ and defines i as the index of the connected component containing u. Each processor remove from V_+ its vertex is not in C_i.

Step 11 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, 1 processor by cell $v \notin V_+$ and it tests that $V_{+1} \cap N(v) \neq \emptyset$.

Steps 12-16 are computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, 1 processor by cell $v \in B_+$ and it tests that $f(x_{N(v)}) = 1$. If there is a cell that verifies this condition return *Reject*.

□

7.2.2 Monotone rules

Given that we know the complexity of Stability for 23 then AsyncStability has at the most the same complexity.

Theorem 7.2.4. AsyncStability *is in* ***NC*** *for the rule* 23.

Proof. In [39] is shown an algorithm solving Stability in time $\mathcal{O}(\log^2 n)$ with $\mathcal{O}(n^3/\log n)$ processors. Roughly the stable cells are characterized as the cells in a bi-connected component (cycles) or a cell in a path between two cycles in the graph induced by inactive cells. The complexity of the algorithm is then given by the complexity of computing bi-connected components. More information about this can be found in [46].

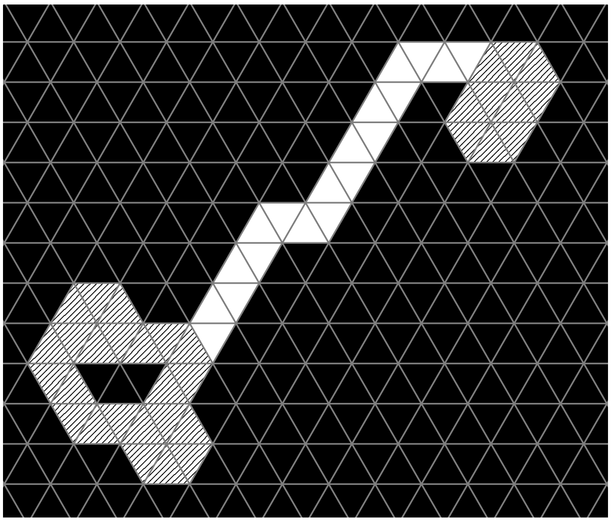


Figure 7.2: Example of a fixed point for the rule 23. Cells with lines are cells in a bi-connected component of inactive cells. The white cells are cells in a path of inactive cells connecting two bi-connected component of inactive cells.

□

Moreover, we can use this fact to know the complexity of AsyncStability in its non-monotone versions, the ACA 2.

Lemma 7.2.5. *Let x be a configuration and u a cell with at least one inactive neighbor. Then u is stable for the rule 23 if and only if u is stable for the rule 2.*

Proof. Note that a site u that is stable for rule 23 is directly stable for rule 2. Indeed an updating scheme that activates u for rule 2 also activates u on rule 23. Suppose now that u is not stable for rule 23, and let σ be an updating scheme such that after t time-steps u becomes active. Moreover, we pick σ such that t is minimum. Since t is minimum, we can assume that every cell that is updated before u is initially inactive, and switches from inactive to active. Moreover, note that a cell with three active neighbors does not affect the dynamics of other cells, because active cells remain active. Therefore, we assume that every cell updated in σ before u had exactly two inactive neighbors. Finally, suppose that in $t-1$ cell u had three active neighbors. Since we are assuming that u had at least one inactive neighbor, it means that there is a time step in $0, \dots, t-2$ in which u had two active neighbors. This contradicts the minimality of t. We deduce that u had exactly two active neighbors at time t. Therefore u becomes active on rule 2 updated according to σ.

□

The previous lemma shows that it is possible to use the algorithm to solve AsyncStability for 23 to solve AsyncStability for the rule 2.

Theorem 7.2.6. AsyncStability *is in* ***NC*** *for the rule* 2.

Proof. The following algorithm verifies the conditions to deduce the rule 2 from the rule 23, then solves stability for the rule 23.

Algorithm 12 AsyncStability solving 2

```
Input: $x \in \{0,1\}^{T(n)}$ and $u \in T(n)$ such that $x_u = 0$.
1: if $N(u)$ has three active neighbors then
2:     return Reject
3: end if
4: To solve AsyncStability for the rule 23.
5: return The same answer obtained in the previous line.
```

Let $N = n^2$ the size of the input.

Steps 1-3 are computed easily in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, because is to compute the sum of the states of three cells.

Steps 4 are computed in time $\mathcal{O}(\log^2 n)$ using $\mathcal{O}(n^3/\log n)$ processors, because is the complexity of AsyncStability for the rule 234, see theorem 7.2.4.

□

7.3 Square grid

Here we study the complexity of AsyncStability for the FTACA on a square grid, obtaining that in some rules this problem is in **NC**. We grouped these rules according to the strategy used to prove its complexity: 1, 12, 13, 14, 123, 134 and 124 use the infiltration technique; 23, 234, 3 and 34 use the monotony and the rules ϕ, 4 and 1234 are trivial.

7.3.1 The infiltration technique

In this section we study the rules where an inactive cell becomes active with one active neighbor. This cases include the rules 1, 12, 13, 14, 123, 134 and 124. As well as for the triangular grid we we define, for an initial configuration x, V_{+1} the set of all cells in $\mathbb{Z}^2$ that *need exactly one active neighbor to be activated*, formally

$$V_{+1} = \{v \in [n] \times [n] : x_v = 0 \ \wedge \ \sum_{w \in N(v)} x_w + 1 \in \mathcal{I}_{\mathcal{F}}\},$$

and we define G_{+1} as the graph induced by V_{+1}. Also we define the *boundary* B_{+1} as the cell in the complement of V_{+1} with at least one neighbor in V_{+1}, formally

$$B_{+1} = \{v \notin V_{+1} : V_{+1} \cap N(v) \neq \emptyset\}.$$

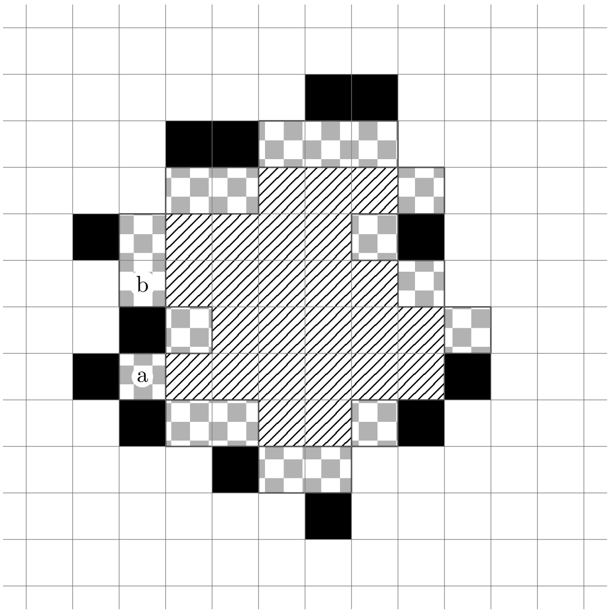


Figure 7.3: ▨: Cells in V_{+1}. ▩: Cells in B_{+1}. ■: Active cells . Example of V_{+1} and B_{+1} for rule 1. The cell (a) is not in V_{+1} because it can not evolve. The cell (b) is not in V_{+1} because it evolves immediately.

Without loss of generality, we suppose that G_{+1} is connected and containing u (otherwise, we restrict to the connected component of G_{+1} containing u). Note that the infiltration lemma (Lemma 7.2.1) is valid too for the square grid, then we need to show that border cells only depend on their neighborhood to decide if they are stable or not.

Lemma 7.3.1. *For the rules* 1, 12, 13, 14, 123, 134 *and* 124 *at least one of the following is true for each boundary cell* $v \in B_{+1}$*:*

- *To decide if this boundary cell will become active or if it is stable depends only on* $N(v)$*.*
- *A cell in* V_{+1} *becomes active.*

Proof. Let $v \in B_{+1}$. We will consider the following facts:

- The cell v has not four active neighbors, in the opposite case meaning then at least one neighbor in V_{+1} becomes active. Thus we will not consider the case when v has four active neighbors.
- By definition of B_{+1}, v has at least one inactive neighbor, the neighbor in V_{+1}.
- Given that $v \notin V_{+1}$, then we need to explore the following cases, where v has at least one active neighbor. $v \in V_{+1}$ otherwise.

– If v has exactly one active neighbor, then v evolves in one time-step and we are done, because always $1 \in \mathcal{I}_F$.

– If v has exactly two active neighbors, then v evolves in one iteration and we decide (i.e. 2 belongs to $\mathcal{I}_F$) or it does not evolve. Note that if $v \in B_{+1}$ and does not evolve, then $3 \notin \mathcal{I}_F$, otherwise $v \in B_{+1}$. Thus v is stable and we decide.

– If v has exactly three active neighbors, then it becomes active (i.e. 3 belongs to $\mathcal{I}_F$) or remains inactive, (i.e. $3 \notin \mathcal{I}_F$), then v is stable. Remember that if v has four active neighbors, then at least one cell in V_{+1} was activated.

Thus we deduce the lemma.

□

Theorem 7.3.2. *AsyncStability is in* ***NC*** *for the rules* 1, 12*,* 13*,* 14*,* 123*,* 134 *and* 124*.*

Proof. Let (x, u) be an input of AsyncStability, i.e. x is a finite configuration of dimensions $n \times n$, and u is a site in $[n] \times [n]$. Our algorithm for AsyncStability first checks if the neighborhood of u is a stable pattern or can evolve in one step, then computes V_{+1} and G_{+1}. Then, the algorithm computes the connected components of G_{+1} and restricts G_{+1} to the connected component containing u and then computes B_{+1}. Finally, the algorithm answers *Reject* if there is a vertex $v \in B_{+1}$ that can be activated. It answers *Accept* otherwise.

This algorithm works on the rules changing with three four neighbors too, because if a boundary with four active neighbors was activated, then there is a active cell in V_{+1}.

Algorithm 13 AsyncStability solving 1, 12, 13, 14, 123, 134 and 124

```
Input: $x \in \{0,1\}^{[n]\times[n]}$ and $u \in [n] \times [n]$ such that $x_u = 0$.
 1: if $x_{N(u)}$ is a stable pattern then
 2:     return Accept
 3: end if
 4: if $f(x_{N(u)}) = 1$ then
 5:     return Reject
 6: end if
 7: Compute the $V_{+1} = \{v \in \mathbb{Z}^2 : x_v = 0 \;\wedge\; |x_{N(v)}|_1 + 1 \in \mathcal{I}_F\}$.
 8: Compute the graph $G_{+1} = G[V_{+1}]$.
 9: Compute the connected components of $G_{+1}$, $\{C_i\}_{i=1}^{M}$ .
10: Redefine $V_{+1} = C_i : u \in C_i$.
11: Compute the $B_{+1} = \{v \notin V_{+1} : V_{+1} \cap N(v) \neq \emptyset\}$.
12: for all $v \in B_{+1}$ do in parallel
13:     if $f(x_{N(v)}) = 1$ then
14:         return Reject
15:     end if
16: end for
17: return Accept
```

Let $N = n^2$ the size of the input.

Steps 1-6 are computed easily in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors.

Step 7 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, 1 processor by cell $v \in [n] \times [n]$ and it checks that $|x_{N(v)}|_1 + 1 \in \mathcal{I}_F$.

Step 8 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, 1 processor by edge (u, v) in the grid (there is $\mathcal{O}(N)$ edges) and it adds (u, v) to the edges of G_{+1} if u and v are in V_{+1}.

Step 9 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors by proposition 3.2.5.

Step 10 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, 1 processor by cell v in each connected component, it checks that $v = u$ and define i as the index of the connected component containing u. Each processor removes from V_+ its vertex if it is not in C_i.

Step 11 is computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, 1 processor by cell $v \notin V_+$ and it checks that $V_{+1} \cap N(v) \neq \emptyset$.

Steps 12-16 are computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors. It is used 1 processor by cell $v \in B_+$ and this processor checks that $f(x_{N(v)}) = 1$. If there is a cell that verifies this condition, then returns *Reject*. *Accepts* otherwise.

□

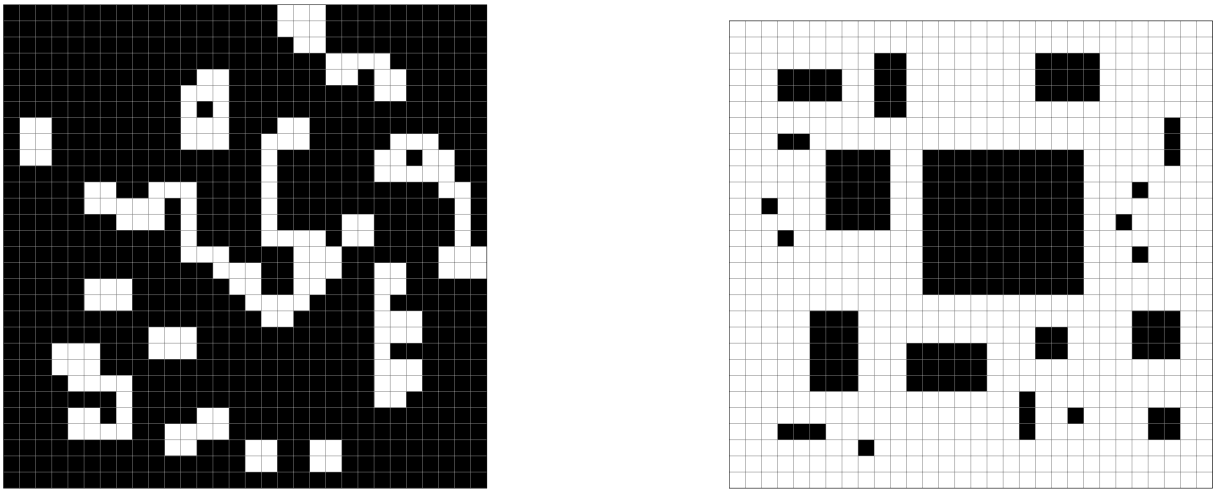
7.3.2 Monotone rules

Given that we know the complexity of Stability for the monotone rules on square grid then AsyncStability has at most the same complexity.

Theorem 7.3.3. AsyncStability *is in* ***NC*** *for the rule* 234 *and* 34.

Proof. In [39] is shown an algorithm solving Stability in time $\mathcal{O}(\log^2 n)$ with $\mathcal{O}(n^3/\log n)$ processors for the rule 34 non strict majority FCA. Roughly the stable cells are characterized as the cells in a bi-connected component (cycles) or a cell in a path between two cycles in the graph induced by inactive cells. The complexity of the algorithm is given by the complexity of computing bi-connected components.

For the rule 234 we use the algorithm in theorem 6.3.3, it is based on the characterization of the stable cells, it is



(a) Fixed point for rule 34.

(b) Fixed point for rule 234.

Figure 7.4: Example of fixed points for the rule 34 and 234. For the rule 34 the stable cells are bi-connected components of inactive cells or path of inactive cells connecting bi-connected components of inactive cells. For the rule 234 the stable cells are a tri-connected components of inactive cells.

□

We can extend this result to the non-monotone ACA 23 and 3 using the monotony of 234 and 34 respectively.

Lemma 7.3.4. *Let x a configuration and u a cell with at least one inactive neighbor. Then u is stable for the rule 234 (34) if and only if u is stable for the rule 23 (3).*

Proof. We will call F_{23} and F_{234} the FCA with rules 23 and 234 respectively.

It is direct that if u is stable for rule 234 then it is stable for rule 23 and note that for any upgrade scheme rules 23 and 234 are only different for cells with four active neighbors.

Suppose that there is a stable u cell for 23 but not for 234, i.e. there is a σ updating scheme and a t time such that $F_{234}^{\sigma(t)}(c)_u = 1$ and $F_{23}^{\sigma(t)}(c)_u = 0$. So $F_{23}^{\sigma(t)}(c)$ has four stable neighbors. Be t' the least time where $F_{23}^{sigma(t')}(c)_u$ has three active neighbors. Note that $F_{234}^{\sigma(t')}(c)_u$ also has three active neighbors. If we define σ' as σ until the time $t'-1$ and $\sigma'(t') = u$, we get an updating scheme that changes to u for rule 234, contradicting the fact that u is stable for 234.

For rules 34 and 3 is analogous. □

From the previous lemma we get that the problem AsyncStability is **NC** for the rules 23 and 3.

Theorem 7.3.5. AsyncStability *is in* ***NC*** *for the rule* 23 *and* 3.

Proof. The following algorithm verify the conditions to deduce the rule 2 from the rule 23, then solve stability for the rule 23.

Algorithm 14 AsyncStability solving 23 (3)

```
Input: $x \in \{0,1\}^{[n]\times[n]}$ and $u \in [n]\times[n]$ such that $x_u = 0$.
1: if $x_{N(u)}$ has four active neighbors then
2:     return Reject
3: end if
4: To solve AsyncStability for the rule 234 (34).
5: return The same answer obtained in the previous line.
```

Let $N = n^2$ the size of the input.

Steps 1-3 are computed easily in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, because it is computing the sum of the states of its neighbors.

Steps 4 are computed in time $\mathcal{O}(\log^2 n)$ using $\mathcal{O}(n^3/\log n)$ processors., because is the complexity of AsyncStability for the rule 234 (34), see theorem 7.3.3.

□

7.4 Rules with 0 non quiescent

If we consider the rules where 0 is not quiescent the infiltration approach can be modified to compute AsyncStability in **NC** too. For the infiltration approach is enough to remove the inactive cells with only inactive cell in its neighborhood to V_{+1}. For monotone approach note that for all configuration c and updating scheme σ we have $F_{23}^{\sigma}(c) \leq F_{023}^{\sigma}(c)$.

Theorem 7.4.1. AsyncStability *is in* ***NC*** *for the rules* 01*,* 012 *and* 013 *in the triangular grid and for the rules* 01*,* 012*,* 013*,* 014*,* 0123*,* 0134 *and* 0124 *in the square grid.*

Proof. Let F be one of the rules of the theorem and u the decision cell. We calculate V_{+1} as if rule 0 were quiescent. If in V_{+1} there is a cell with all its neighbors inactive, so u is not stable, otherwise the algorithm for the rule with 0 quiescent will give the correct answer, because there are no cells with all its neighbors inactive in V_{+1} nor in B_{+1}, because all its cells have at least one active cell.

The following algorithm is able to solve AsyncStability using the algorithms for rules with 0 as a quiescent state.

Algorithm 15 AsyncStability solving 01, 012 and 013 in the triangular grid and 01, 012, 013, 014, 0123, 0134 and 0124 in the square grid.

```
Input: $x \in \{0,1\}^{[n]\times[n]}$ and $u \in [n]\times[n]$ such that $x_u = 0$.
1: Compute the $V_{+1} = \{v \in \mathbb{Z}^2 : x_v = 0 \ \wedge \ |x_{N(v)}|_1 + 1 \in \mathcal{I}_F\}$.
2: for all  $v \in V_{+1}$ do in parallel
3:     if $|x_{N(v)}|_1 = 0$ then
4:         return Reject
5:     end if
6: end for
7: To solve AsyncStability for the rule same rule, but with 0 as quiescent state.
8: return The same answer obtained in the previous line.
```

Let $N = n^2$ the size of the input.

Step 1 is computed easily in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, because is to compute the sum of three or four cell.

Steps 3-6 are computed in time $\mathcal{O}(\log N)$ using $\mathcal{O}(N)$ processors, one per cell to compute the sum of its neighbors.

Steps 7 is computed in time $\mathcal{O}(\log^2 n)$ using $\mathcal{O}(n^3/\log n)$ processors, by theorems 7.2.4, 7.2.6, 7.3.3 and 7.3.5.

□

7.5 NP completeness

We will show that using the FTCA 2 on the tri-dimensional square grid, the AsyncStability problem is **NP**-complete. It is **NP** because we can verify if a given updating scheme activates a cell in a polynomial time for any FTACA. Note that if we consider the rule 2 in a tri-dimensional space, we can build the logic gates AND and OR and duplicate signals over a plane simulating a two dimensional FTCA 2. This work also in an asynchronous way. Instead of using an XOR gate for wire crossings, we can simply use the third dimension to pass one wire over the other, similar to Figure 4.6.

We will build an abstract system similar to CNF formulas, but considering three state instead of two, we will prove that this has system a **NP**-complete problem and we will use them to prove that AsyncStability is **NP**-complete too.

We consider the following states $\mathbb{T} = \{(0,1),(1,0),(0,0)\}$. To give an intuition, we will call $\mathbb{0} = (0,1), \mathbb{1} = (1,0)$ and $e = (0,0)$, then $\mathbb{T} = \{\mathbb{0}, \mathbb{1}, e\}$. Also consider the operators on $\mathbb{T}$ defined by the table 7.1.

X	Y	$X \Vee Y$
$\mathbb{0}$	$\mathbb{0}$	$\mathbb{0}$
$\mathbb{0}$	$\mathbb{1}$	$\mathbb{1}$
$\mathbb{1}$	$\mathbb{0}$	$\mathbb{1}$
$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$
$\mathbb{0}$	e	e
$\mathbb{1}$	e	$\mathbb{1}$
e	$\mathbb{0}$	e
e	$\mathbb{1}$	$\mathbb{1}$

X	Y	$X \Vee Y$
$(0,1)$	$(0,1)$	$(0,1)$
$(0,1)$	$(1,0)$	$(1,0)$
$(1,0)$	$(0,1)$	$(1,0)$
$(1,0)$	$(1,0)$	$(1,0)$
$(0,1)$	$(0,0)$	$(0,0)$
$(1,0)$	$(0,0)$	$(1,0)$
$(0,0)$	$(0,1)$	$(0,0)$
$(0,0)$	$(1,0)$	$(1,0)$

(a) The OR on $\mathbb{T}$. In the right table, the first component of $X \Vee Y$ is computed as $X_1 \vee Y_1$ and the second one is computed as $X_2 \wedge Y_2$.

X	Y	$X \Wedge Y$
$\mathbb{0}$	$\mathbb{0}$	$\mathbb{0}$
$\mathbb{0}$	$\mathbb{1}$	$\mathbb{0}$
$\mathbb{1}$	$\mathbb{0}$	$\mathbb{0}$
$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$
$\mathbb{0}$	e	$\mathbb{0}$
$\mathbb{1}$	e	e
e	$\mathbb{0}$	$\mathbb{0}$
e	$\mathbb{1}$	e

X	Y	$X \Wedge Y$
$(0,1)$	$(0,1)$	$(0,1)$
$(0,1)$	$(1,0)$	$(0,1)$
$(1,0)$	$(0,1)$	$(0,1)$
$(1,0)$	$(1,0)$	$(1,0)$
$(0,1)$	$(0,0)$	$(0,1)$
$(1,0)$	$(0,0)$	$(0,0)$
$(0,0)$	$(0,1)$	$(0,1)$
$(0,0)$	$(1,0)$	$(0,0)$

(b) The AND on $\mathbb{T}$. In the right table, the first component of $X \Wedge Y$ is computed as $X_1 \wedge Y_1$ and the second one is computed as $X_2 \vee Y_2$.

X	$\neg X$
$\mathbb{0}$	$\mathbb{1}$
$\mathbb{1}$	$\mathbb{0}$
e	e

X	$\neg X$
$(0,1)$	$(1,0)$
$(1,0)$	$(0,1)$
$(0,0)$	$(0,0)$

(c) The negation on $\mathbb{T}$. The first and second components of $\neg X$ are compute as the negations of the first and second component of X respectively.

Table 7.1: Logical operator on $\mathbb{T}$. The left tables has the values X and Y as symbols and the right tables has the values X and Y as pairs.

We can interpret e as an unknown value. In the usual logic for any value of e, $e \vee 1 = 1$, see row 8 in Figure 7.1a, but the value of $e \vee 0$ depends on e, so if e is unknown, then $e \vee 0$ is also unknown, "$e \vee 0 = e$", see row 7 in Figure 7.1a.

We call to $\mathbb{C}$ a $\mathbb{T}$-CNF formula if these use the symbols $\Vee$, $\Wedge$ and $\neg$ and exchanging this symbols by $\vee$, $\wedge$ and $\neg$ respectively we obtain a CNF formula. If C is a CNF formula, we call $\mathbb{C}$ a $\mathbb{T}$-CNF formula obtained by the previous exchange.

Also we define $\phi : \{0,1\} \to \mathbb{T}$ as $\phi(0) = \mathbb{0}$ and $\phi(1) = \mathbb{1}$ and for all n, $\overline{\phi} : \{0,1\}^n \to \mathbb{T}^n$ is the function that applies ϕ component by component.

These definitions induce the following decision problem analogous to SAT, called $\mathbb{T}$-SAT.

Satisfiability Problem on $\mathbb{T}$ ($\mathbb{T}$-SAT)
Input: A $\mathbb{T}$-CNF formula.
Question: Is there an input value such that the output value is $\mathbb{1}$?

Looking only the rows non containing e is easy to check that if we only consider inputs $\mathbb{0}$ and $\mathbb{1}$ for the $\mathbb{T}$-CNF formulas, then SAT and $\mathbb{T}$-SAT are equivalent.

The following lemma says that if X satisfies a $\mathbb{T}$-CNF formula, then if we replace an entry of X with a e value with the value 1 and call it X', then X' also satisfies our $\mathbb{T}$-CNFformula. So if a $\mathbb{T}$-CNF formula is satisfactory, we can always find X' without e entries that satisfies our $\mathbb{T}$-CNF formula.

Lemma 7.5.1. *Let $\mathbb{C}$ a $\mathbb{T}$-CNF formula induced by C, then if we consider only inputs without e values, then the formula C is satisfiable if and only if the $\mathbb{T}$-formula is satisfiable. Moreover, if the answers of $\mathbb{T}$-SAT with formula $\mathbb{C}$ is* yes *by X without e values, then the answers of SAT with formula C is* yes *by $\overline{\phi}^{-1}(X)$, i.e. $C(\overline{\phi}^{-1}(X)) = 1$.*

Lemma 7.5.2. *Let $\mathbb{C}$ a $\mathbb{T}$-CNF formula $X = (X^1, ..., X^n)$ satisfies $\mathbb{C}$. Let*

$$\overline{X}^i = \begin{cases} X^i & \textit{if } X^i \neq e \\ \mathbb{1} & \textit{otherwise.} \end{cases},$$

then $\overline{X}$ also satisfies $\mathbb{C}$.

Proof. Let $\mathbb{C}$ a $\mathbb{T}$-CNF formula and the answers of $\mathbb{T}$-SAT is *yes* by $X = (X^1, ..., X^n)$, i.e. $\mathbb{C}(X) = \mathbb{1} = (1, 0)$, then there are C_1, C_2 CNF formulas such that $C_1(x1, x2) = 1$ and $C_2(x1, x2) = 0$, where $xi = (X^1_i, ..., X^n_i)$. Moreover, C_1 is the CNF formula exchanging the symbols $\Vee$ and $\Wedge$ by $\vee$, $\wedge$ respectively, the literals X^i by $x1^i$ and $\boldsymbol{\neg}\, X^i$ by $x2^i$. Analogously, we define C_2.

Note that C_1 and C_2 are monotone circuit, then if the output of each of these circuit is 1 for an input, then is 1 also for an input obtained by exchanging 0 input values by 1 input values.

Finally, to exchange e by $\mathbb{1}$ is equivalent to add 1 values in inputs of C_1, then the first component of the output of $\mathbb{C}$ is 1 too. It is not possible that the second component of the output of $\mathbb{C}$ changes to 1 too, because this means that by composition of $\Vee$ and $\Wedge$ and $\boldsymbol{\neg}$ we can obtain an output $(1, 1)$, but all these operators are closet on $\mathbb{T}$.

□

Proposition 7.5.3. *$\mathbb{T}$-SAT is **NP**-complete.*

Proof. $\mathbb{T}$-SAT is in **NP** because if a $\mathbb{T}$-CNF formula is satisfiable and we know the input that satisfies the $\mathbb{T}$-CNF formula, then we need to compute the outputs of this formula. We can do it in polynomial time because circuit value problem is in **P**.

Let C a CNF formula, we need to prove that C is satisfiable if and only if $\mathbb{C}$ is satisfiable too.

If C is satisfiable, then there is an input x such that $c(x) = 1$, then, by lemma 7.5.1, $\mathbb{C}(X) = \mathbb{1}$ and $\mathbb{C}$ is satisfiable.

Conversely, if $\mathbb{C}$ is satisfiable, then there is an input X such that $\mathbb{C}(X) = \mathbb{1}$. If all the values X^i are $\mathbb{0}$ or $\mathbb{1}$, then, by lemma 7.5.1, $C(\overline{\phi}^{-1}(X)) = 1$ and the C is satisfiable. Otherwise, by lemma 7.5.1, we can build an input $\overline{X}$ such that all the values $\overline{X}^i$ are $\mathbb{0}$ or $\mathbb{1}$. Thus, $C(\overline{\phi}^{-1}(\overline{X})) = 1$ and the C is satisfiable.

We use polynomial time to build the $\mathbb{T}$-CNF formula, because it is simply build a CNF formula copy of the original and another copy but with the AND gates exchanged with OR gates.

This proves the **NP**-completeness of $\mathbb{T}$-SAT. □

We will use this new **NP**-complete problem to prove the **NP**-completeness of rule 2 on a square tri-dimensional grid, but for this we need the following gadget.

Lemma 7.5.4. *There is a gadget that allows to generate at most one signal in one of two directions for every updating scheme.*

Proof. With no loss of generality, let σ an updating scheme that in each iteration the iterated cell changes, except when the fixed point is reached.

Note that σ should begin by iterating the a or c cell in the Figure 7.5. By symmetry we will study the case when the first iterated cell is the cell a, i.e. $\sigma(1) = a$. If the second iterated cell is c then a fixed point is reached. So, after iterating the a cell it is only possible to iterate b, $\sigma(2) = b$. This prohibits iterate c and allows to iterate only d, $\sigma(3) = d$. Finally this allows to continue iterating cells until change the cell x, but no way to iterate that has iterated a, b and d can change the cell y. □

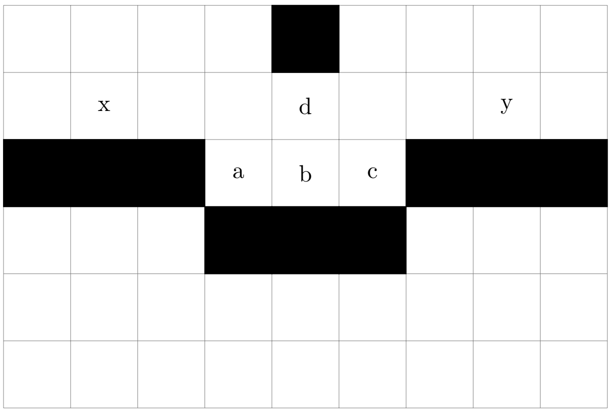


Figure 7.5: Gadget sending at most one signal for one of two outputs.

Theorem 7.5.5. *AsyncStability is **NP**-complete for the rule 2 on a square tri-dimensional grid.*

Proof. Let's reduce $\mathbb{T}$-SAT to AsyncStability. Let $\mathbb{C}$ be a $\mathbb{T}$-CNF formula with literals $X^1, ..., X^2$. We will simulate your C_1 and C_2 formulas of the lemma 7.5.2 by building them in two layers in a three-dimensional configuration. Each one of the formulas C_1 and C_2 are constructed according to the theorem 6.3.12 on a different parallel layer.

Note that the wiring of each layer is the same and only the gates change. So,

- $\mathbb{1}$ is represented by a signal on layer 1 and the absence of signal on layer 2,
- $\mathbb{0}$ is represented by a signal on layer 2 and the absence of signal on layer 1 and
- e is represented by absence of signal on both layers.

Now we need to introduce signals in layers 1 and 2.

For each literal X^i we put a gadget as in Figure 7.5 and connect its output 1 with the positions of the inputs of the literals $x1i$ and output 2 with the inputs of the literals $x2i$. So each pair of wires in layers 1 and 2 has at most one signal.

The construction in polynomial time is due to the fact that the construction of each circuit can be done in polynomial time. □

Note that $\mathbb{T}$-SAT is **NP**-complete also if the question is *Is there an input value such that the output value is* $\mathbb{0}$*?*, but the question *Is there an input value such that the output value is e?* is trivial, it is enough to choose all the inputs with value e.

Observes that if a $\mathbb{T}$-CNF formula is satisfied by X containing a variable with state e means that any choice of states of this variable also satisfies the $\mathbb{T}$-CNF formula.

7.6 Concluding Remarks and Future Works

In this chapter we proved that the AsyncStability problem is in **NC** for all Freezing Totalistic Asynchronous Cellular Automata (FTACA) in the triangular and square grid with the von Neumann Neighborhood. There are 16 FTACA on the triangular grid and 32 on the square grid. We focused our study on 8 rules on the triangular grid and 16 on the square grid, where the inactive state is quiescent. There are some rules that are trivial (ϕ, 123 and 3 on triangular grid and ϕ, 1234 and 4 on square grid). Rules 1, 12 and 13 are in **NC** by the infiltration approach and the rules 2 and 23 are in **NC** by the monotone approach.

Problems where the complexity of AsyncStability remains open are those where the complexity of Stability is P-complete, i.e. they are rules 2 and 24 on the square grid. In these cases we can no longer use the XOR gate, because different updating scheme produce different outputs for the same inputs. we have used the XOR gate to build a crossover and simulate non-planar Boolean circuits. That way, we can only build monotone planar circuits, then we can only solve the *planar and monotone circuit value problem*, but this problem is in **NC** [68]. We study the rule 4 in dimension three in order to avoid the problem to crossing signals and we prove that in this cases AsyncStability is **NP**-complete, the higher complexity for AsyncStability.

If we consider the rules where 0 is not quiescent the infiltration approach and monotone approach can be modified to compute AsyncStability in **NC** too. For the infiltration approach is enough to remove the inactive cells with only inactive cell in its neighborhood to V_+, because now this cell can not 1 neighbor to active.

Another way to explore is to find different kinds of FTACA, with other Neighborhoods, more states or dimension, where we can find FTACA with higher complexity, as FTACA where the complexity of AsyncStability its **NP**-complete.

Chapter 8

Conclusion et travaux futurs

Dans ce travail, nous avons étudié la complexité des automates cellulaires coagulants de différents points de vue. D'une part, nous avons étudié s'il existe un seul automate cellulaire coagulant montrant tous les comportements que l'on peut trouver sur ces automates cellulaires. Nous en trouvons un et nous avons étudié ses propriétés et ses limites.

D'autre part, nous avons étudié la complexité d'un automate cellulaire coagulant avec des états et des quartiers très limités. Nous avons trouvé un quartier de von Neumann à deux états gelant des automates cellulaires totalistes avec une complexité "**P** "-complète pour STABILITY.

On peut conclure que, malgré l'apparente limitation de la propriété coagulation, cette famille d'automates cellulaires présente un comportement très riche et intéressant.

8.1 Universalité des automates cellulaires coagulants

Nous avons commencé notre étude des automates cellulaires coagulants par la recherche d'un automate cellulaire coagulant intrinsèquement universel, c'est-à-dire capable de simuler tout autre automate cellulaire coagulant. Nous avons montré qu'avec la définition habituelle de la simulation, il n'existe pas d'automate de ce type. Nous étudions donc une notion plus générale de la simulation appelée simulation contextuelle. Selon cette notion, où la macrocellule simulante dépend non seulement de la cellule simulée, elle dépend aussi de ses voisins, il a été possible de trouver un automate cellulaire coagulant intrinsèquement universel. Il a deux changements et il n'est pas possible d'en trouver un avec moins de changements, parce que les automates cellulaires coagulants avec un seul changement ne peuvent pas croiser les signaux, alors il ne peut pas simuler un automate cellulaire avec des signaux de passage, car il y a aussi les automates cellulaire coagulants qui changent deux ou plus. De plus, nous avons utilisé le deuxième changement uniquement pour croiser les signaux.

Nous avons construit un automate cellulaire coagulant intrinsèquement universel avec un seul changement en utilisant la troisième dimension pour faire les croisements. Puisque les circuits utilisés pour construire l'automate cellulaire coagulant intrinsèquement universel étaient possibles avec l'automate cellulaire coagulant avec la règle 2, alors avec sa version tridimensionnelle nous avons trouvé un automate cellulaire coagulant intrinsèquement universel avec quartier von Neumann et seulement deux états, donc avec seulement un changement.

Une autre façon d'obtenir un automate cellulaire coagulant intrinsèquement universel avec un changement sans ajouter d'autres dimensions, était d'agrandir un peu son voisinage, dans ce cas nous avons vu qu'avec le quartier Moore nous avons pu faire des croix planes en utilisant les diagonales de ce quartier, comme les fous sur les places avec différentes couleurs ne touchent pas.

Lorsque nous avons imposé plus de conditions aux automates cellulaires coagulants intrinsèquement universels, nous avons également rencontré des obstacles. Il n'existe pas d'automates cellulaires coagulants monotones intrinsèquement universels. La coagulation et la monotonie nous donnent un AC confluent, c'est-à-dire, où chaque schéma de mise à jour se termine au même point fixe. S'il y avait eu un automate cellulaire coagulant monotone intrinsèquement universel, il doit simuler des automates cellulaires coagulants non monotones, avec lesquels une contradiction a été construite.

Enfin, nous avons trouvé une caractérisation des automates cellulaires qui peut être simulée en fonction du contexte par des automates cellulaires coagulants : les CA ne sont pas nécessairement des automates coagulants mais ont un comportement très semblable. Elle se caractérise par l'existence d'une énergie locale explicite, une fonction qui attribue à chaque quartier une valeur numérique (l'énergie) qui diminue à mesure que la dynamique de l'AC progresse. Comme pour les automates cellulaires coagulants dans ces cas, il y a une quantité locale qui est perdue à chaque itération.

Quelques sujets qui pourraient être suivis seraient la recherche d'un plus petit automate cellulaire coagulant intrinsèquement universel avec quartier von Neumann, c'est-à-dire qui a le moins d'états possible. Puisqu'il doit avoir deux changements, cela signifie que le nombre minimum d'états qu'il peut avoir est de trois. On pourrait également rechercher l'universalité intrinsèque dans les sous-classes coagulant : par exemple en recherchant un automate cellulaire monotone et ccoagulant intrinsèquement universel pour la classe des automates cellulaires coagulants monotones et un automate cellulaire coagulant à un changement intrinsèquement universel pour l'automate cellulaire à un changement.

8.2 Sur la complexité du problème de stabilité des automates cellulaires binaires coagulants totalistiques

Nous avons concentré notre étude sur une famille très simple d'automates cellulaires coagulants : les automates cellulaires coagulants totalisateurs avec voisinage von Neumann et deux états. Nous étudions cette famille dans deux types de grille : la grille tessellée avec des carrés et avec des triangles, où chaque cellule a respectivement 3 et 4 voisins, donc dans chaque cas nous avons 16 et 32 règles différentes, mais au début nous avons considéré seulement les règles où l'état inactif est quiescent. Nous identifions ces règles avec la concaténation de tous les nombres de sorte que la règle active la cellule centrale avec ce nombre de voisins actifs.

Ici, nous nous sommes intéressés à la complexité informatique du problème de savoir si une cellule va changer ou non, ce que nous appelons Stability.

Dans les deux grilles, nous regroupons les règles en fonction des approches que nous utilisons pour résoudre Stability:

- Règles simples : sont celles où Stability est décidée trivialement. Par exemple, règle 4 pour la grille carrée, où la cellule est stable à moins qu'elle n'ait initialement 4 voisins actifs.
- Règles topologiques : ce sont celles où la stabilité est donnée par la structure qui a l'ensemble des cellules inactives. Par exemple la règle 34 dans la grille carrée, un ensemble de cellules inactives où toutes les cellules ont au moins deux voisins à l'intérieur, est un ensemble stable, car après l'application de la règle aucune cellule n'est activée. Ici, nous utilisons les algorithmes **NC** pour calculer les composants connectés, bi-connectés et tri-connectés.
- Règles algébriques : sont celles où la valeur d'une cellule inactive peut être calculée par des opérations algébriques du reste pour accélérer le calcul. Par exemple la règle 12 pour la grille carrée, une cellule change avec un ou deux voisins actifs ou l'équivalent pour calculer le OU des voisins, même s'il y a plusieurs voisins inactifs connectés il est possible de calculer la valeur comme le OU des voisins les plus éloignés. Ici nous utilisons les algorithmes **NC** pour calculer la somme du préfixe.

- Règles fractales : ce sont celles où dans la dynamique on observe un comportement fractal. Par exemple, la règle 1 pour la grille carrée, commencée avec une seule cellule active, montre une croissance similaire à celle d'un flocon de neige.

Dans la grille triangulaire nous avons montré que Stability est dans **NC** pour toutes les règles sauf les règles fractales. Pour obtenir une complexité maximale, nous avons ajouté un état supplémentaire à la règle 2 pour simuler des circuits logiques comme la règle 2 dans la grille carrée.

Pour la grille carrée, nous avons deux règles où le problème de stabilité est **P**-complet. C'est la plus grande complexité pour Stability sur les automates cellulaires coagulants. Ce sont les règles 2 et 24, où il était possible de simuler des circuits logiques.

Etant donné la difficulté d'étudier Stability dans les règles fractales, nous avons étudié un problème plus simple, à savoir la "Reachability, si avec deux configurations on peut atteindre l'une de l'autre par itérations de l'automate cellulaire. Si nous pouvons résoudre Stability alors nous pouvons résoudre Reachability. Nous avons montré que dans le cas de la règle 1 Reachability est en **NC**.

Pour le reste des règles où l'état inactif n'est pas quiescent, il a été possible de lier la complexité de calcul de Stability par la complexité de la même règle mais avec l'état inactif comme quiescent. La complexité des règles 02 et 024 reste ouverte (les versions de 2 et 24 où 0 n'est pas un état quiescent), où la complexité pourrait éventuellement diminuer.

8.3 Algorithmes parallèles rapides pour les automates cellulaires asynchrones totalistes et coagulants

Dans ce dernier chapitre, nous étudions la même famille d'automates cellulaires coagulants sur la même grille que le chapitre précédent, mais nous modifions le schéma de mise à jour par un schéma asynchrone. Nous nous sommes donc intéressés au calcul de la complexité de calcul de la version de Stability pour les automates cellulaires asynchrones, appelée problème AsyncStability, où une cellule est stable si elle est stable pour tout schéma de mise à jour.

Dans les deux grilles, nous regroupons les règles en fonction des approches que nous utilisons pour résoudre la AsyncStability:

- Règles simples : sont ces règles où AsyncStability est décidée trivialement.
- Règle d'infiltration : sont les règles qui activent une cellule avec exactement un voisin actif. Nous résolvons AsyncStability en cherchant une cellule dans le composant connecté de cellules inactives se connectant à la cellule de décision qu'il est possible d'activer. s'il existe, nous disons qu'une cellule s'infiltre et ceci peut "voyager" sur le composant connecté de cellules inactives se connectant à la cellule de décision jusqu'à leur activation.
- Règles monotones : ce sont les règles qui sont monotones, puis elles sont confluentes. Alors nous avons pu résoudre la Stability sur sa version synchrone et donner la même réponse.

Pour la grille triangulaire, AsyncStability est dans **NC** pour toutes les règles. Tandis que pour la grille carrée, nous savons que AsyncStability in **NC** pour toutes les règles sauf les règles 2 et 24. Notez que ce sont les règles **P**-completes pour le cas synchrone. Pour ces cas ouverts, nous étudions ses versions tridimensionnelles, où nous pouvons construire des portes logiques et utiliser la troisième dimension pour croiser les signaux. Ensuite, avec un gadget spécial, c'est l'équivalent de trouver un schéma de mise à jour changeant une cellule sur une configuration (simulant un circuit logique) que de trouver une affectation satisfaisante à une formule booléenne. Puis AsyncStability dans **NP**-complet pour les règles 2 et 24 dans sa version tridimensionnelle. C'est la plus grande complexité pour AsyncStability sur les automates cellulaires coagulants.

La complexité de AsyncStability pour les règles 2 et 24 en deux dimensions reste ouverte. Aussi il devrait être intéressant ce qui se passe dans d'autres quartiers ou supprimer l'hypothèse totaliste sur les règles étudiées.

8.4 Remarques finales et problèmes ouverts

Malgré leur limite apparente par le fait que chaque cellule ne peut que croître, les automates cellulaires coagulants présentent une grande diversité et complexité. Que nous pouvons les ordonner de la manière suivante, du moins complexe au plus complexe : 1D automates cellulaires coagulants, 2D automates cellulaires coagulants 2D avec le voisinage von Neumann et un changement, et le reste des automates cellulaires coagulants.

- Les automates cellulaires coagulants 1D sont les moins complexes puisque la Stability est au maximum NLOGSPACE ;
- les automates cellulaires coagulants 2D von Neumann voisinage et un changement, parce qu'il a un automate cellulaire coagulant où Stability est **P**-complète, mais ils n'en ont pas un intrinsèquement universel ;
- le reste des automates cellulaires coagulants où il existe un automate cellulaire coagulant intrinsèquement universel.

Si nous relâchons le synchronisme dans les schémas de mise à jour, nous obtenons que la plupart des automates cellulaires coagulants étudiés ont une faible complexité dans la dimension deux, mais si nous les étudions dans la dimension trois, nous obtenons qu'il y en a un où le problème de stabilité est **NP**-complet.

Il y a encore quelques questions à clore dans l'élaboration de cette thèse, telles que :

- **de trouver un automate cellulaire coagulants intrinsèquement universel en dimension deux avec seulement 3 états (le nombre minimum d'états possibles).**

  Dans la construction des automates cellulaires coagulants intrinsèquement universels, nous utilisons un grand nombre d'états mais seulement deux changements par cellule. A partir de là, nous avons construit un système de circuit logique capable de simuler la fonction locale de n'importe quel automate cellulaire coagulant. Nous connaissons déjà un automate cellulaire coagulant capable de simuler n'importe quelle fonction Booblean et en particulier la fonction locale d'un automate cellulaire coagulant. Cet automate est la règle 2, mais, comme nous le savons déjà, il n'a qu'un seul changement et ne peut être intrinsèquement universel. Le problème est que pour croiser les informations, cet automate utilise un gadget construit en utilisant la porte XOR, où les signaux d'entrée doivent arriver en même temps pour simuler le passage. Après le passage d'un signal, il est inutilisable. Ceci pourrait être résolu en ajoutant un troisième état à la règle 2 et ainsi obtenir un automate cellulaire coagulant intrinsèquement universel avec seulement 3 états et un voisinage von Neumann.
- **rechercher un automate cellulaire coagulant intrinsèquement universel pour la classe des automates cellulaires d'un seul changement**

  Puisque les automates cellulaires coagulants avec un seul changement par cellule et le voisinage de von Neumann n'ont pas d'automates intrinsèquement universels, alors nous pourrions étudier ces automates comme une famille indépendante d'automates cellulaires et voir s'il existe un automate cellulaire avec un seul changement capable de simuler tout autre automate cellulaire coagulant avec seulement un changement. Une façon de montrer que ce n'est pas possible serait de trouver un comportement qui "deviendra plus complexe" à mesure que le nombre d'états augmente, de sorte qu'il ne puisse pas être simulé par des automates avec moins d'états, donc tout candidat à être intrinsèquement universel aurait un automate avec plus d'états qu'il ne peut simuler.

- **déterminer la complexité des règles fractales :**

  Pour donner une idée de la difficulté de ces règles, nous allons analyser la règle 1, mais sans utiliser le voisinage de von Neumann, en utilisant le voisinage pour simuler le diagramme espace-temps des automates cellulaires 1D dans des automates cellulaires 2D.

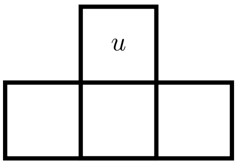


Figure 8.1: Quartier simulant sur grille 2D un automate cellulaire 1D des premiers voisins, où u est la cellule centrale.

Avec ce voisinage, la règle 1 simule le diagramme espace-temps des automates cellulaires élémentaires 22 (ECA 22). L'ECM 22 a été classé par Wolfram dans la classe 3 [70], où la classe 4 contient les automates les plus complexes. Avec cette version 1D de la règle 1, nous pouvons explorer les résultats des études statistiques de cette règle [71, 72] ou la théorie du langage [73].

Chapter 9

Conclusion and Future Works

In this work we have studied complexity of freezing cellular automata from different points of view. On the one hand, we have studied if there is a single freezing cellular automaton showing all the behaviors that we can find on this cellular automata. We find one and we have studied its properties and limitations.

On the other hand, we have studied how complex a freezing cellular automata with very limited states and neighborhood can be. We found a two state von Neumann neighborhood freezing totalistic cellular automata with a **P**-complete STABILITY complexity.

We can conclude that, despite the apparent limitation of the freezing property, this family of cellular automata shows a very rich and interesting behavior.

9.1 Universality in Freezing Cellular Automata

We began our study of the freezing cellular automata with the search for an intrinsically universal freezing cellular automata, that is, capable of simulating any other freezing cellular automata. We showed that with the usual definition of simulation there is no such automaton. Thus we study a more general notion of simulation called context-sensitive simulation. Under this notion, where the simulating macro cell not only depend on the simulated cell, also depends on its neighbors, it was possible to find an intrinsically universal freezing cellular automaton. It has two changes and it is not possible to find one with fewer changes, because the freezing cellular automata with only one change cannot cross signals, then it cannot simulate a cellular automaton crossing signals, as some freezing cellular automata with two or more changes. Moreover, we have used the second change just to cross signals.

We built an intrinsically universal freezing cellular automaton with only one change using the third dimension to make the crossings. Since the circuitry used to build the intrinsically universal freezing cellular automaton was possible to do with the freezing cellular automaton with rule 2, then with its three-dimensional version we have found an intrinsically universal freezing cellular automaton with von Neumann neighborhood and only two states, therefore with only one change.

Another way to obtain an intrinsically universal freezing cellular automaton with one change without adding another dimensions, was to enlarge a little its neighborhood, in this case we saw that with the Moore neighborhood we have been able to make plane crosses using the diagonals of this neighborhood, similar to how bishops in squares with different color never touch.

When we imposed more conditions on the intrinsically universal freezing cellular automata candidate we also encounter impediments. There is no monotone intrinsically universal freezing cellular automata. Freezing and monotony give us a confluent CA, that is, where every updating scheme ends at the same fixed point. If there had been a monotone intrinsically universal freezing cellular automaton, it must simulate non-monotone freezing cellular automata, with which a contradiction was constructed.

Finally, we found a characterization of cellular automata that can be simulated context-sensitive by freezing cellular automata: the CA are not necessarily freezing, but have a very similar behaviors. It is characterized by the existence of an explicit local energy, a function that assigns to each neighborhood a numerical value (the energy) that decreases as the dynamics of the CA advances. Similar to the freezing cellular automata in these cases there is a local quantity that is lost an each iteration.

Some topics that could be followed would be to look for an smallest intrinsically universal freezing cellular automata with von Neumann neighborhood, i.e. that has as few states as possible. Since it must have two changes, it means that the minimum number of states it can have is three. One could also look for intrinsic universality in sub-classes of freezers: for example in looking for an monotone and freezing cellular automata intrinsically universal for the monotone freezing class and a one change freezer cellular automata intrinsically universal for the one change freezing cellular automata.

9.2 On the Complexity of the Stability Problem of Binary Freezing Totalistic Cellular Automata

We have focused our study on a very simple family of freezing cellular automatons: the totalistic freezing cellular automata with von Neumann neighborhood and two states. We study this family in two types of grid: the tessellated grid with squares and with triangles, where each cell has 3 and 4 neighbors respectively, so in each case we have 16 and 32 different rules, but initially we have considered only the rules where the inactive state is quiescent. We identify these rules with the concatenation of all the numbers such that the rule activates the central cell with that number of active neighbors.

Here we were interested in the computational complexity of the problem of knowing if a cell is going to change or not, which we call Stability. In both grids we group the rules according to the approaches we use to solve Stability:

- Simple rules: are those where Stability is decided trivially. For example, rule 4 for the square grid, where the cell is stable unless it initially has 4 active neighbors.
- Topological rules: they are those where the stability is given by the structure that has the set of inactive cells. For example rule 34 in the square grid, a set of inactive cells where all cells have at least two neighbors within, is a stable set, because after applying the rule no cell is activated. Here we use the **NC** algorithms to compute connected, bi-connected and tri-connected components.
- Algebraic rules: are those where the value of an inactive cell can be calculated by algebraic operations of the rest to speed up the calculation. For example rule 12 for the square grid, a cell changes with one or two active neighbors or equivalent to calculate the OR of the neighbors, also if there are several inactive neighbors connected it is possible to calculate the value as the OR of the farthest neighbors. Here we use the **NC** algorithms to compute the prefix sum.
- Fractal rules: they are those where in the dynamics a fractal behavior is observed. For example, rule 1 for the square grid, started with just one active cell, shows a growth pattern similar to a snowflake.

In the triangular grid we have shown that Stability is in **NC** for all rules except in the fractal rules. To achieve maximum complexity we added an extra state to rule 2 to simulate logic circuits like rule 2 in the square grid.

For the square grid we have two rules where the stability problem is **P**-complete. This is the higher complexity for Stability on freezing cellular automata. These are rules 2 and 24, where it was possible to simulate logic circuits.

Given the difficulty of studying Stability in the fractal rules, we have studied a simpler problem, namely Reachability, whether given two configurations it is possible to reach one from the other by iterations of the cellular automaton. If we can solve Stability then we can solve Reachability. We have shown that in the case of rule 1 Reachability is in **NC**.

For the rest of the rules where the inactive state is not quiescent, it was possible to bound the computational complexity of Stability by the complexity of the same rule but with the inactive state as quiescent. The complexity for the rules 02 and 024 remains open (the versions of 2 and 24 where 0 is not a quiescent state), where possibly the complexity could decrease.

9.3 Fast-Parallel Algorithms for Freezing Totalistic Asynchronous Cellular Automata

In this last chapter we study the same family of freezing cellular automata on the same grid that the previous chapter, but we change the updating scheme by an asynchronous one. Thus we were interested in computing the computational complexity of the version of Stability for asynchronous cellular automata, called AsyncStability problem, where a cell is stable is it is stable for any updating scheme.

In both grids we group the rules according to the approaches we use to solve AsyncStability:

- Simple rules: are those rules where AsyncStability is decided trivially.
- Infiltration rule: are those rules that activate a cell with exactly one active neighbor. We solve AsyncStability searching some cell in the connected component of inactive cells connecting to the decision cell that is possible to be activated. if there exists, we say that a cell infiltrate and this can "travel" on the connected component of inactive cells connecting to decision cell until activate them.
- Monotone rules: are those rules that are monotone, then these are confluent. Then we have been able to solve Stability on its synchronous version and give the same answer.

For the triangular grid, AsyncStability is in **NC** for all the rules. While for the square grid, we know that AsyncStability in **NC** for all the rules except the rules 2 and 24. Note that this are the **P**-complete rules for the synchronous case. For these open cases we study its three-dimensional versions, where we can built logics gates and use the third dimension to cross signals. Then, with a special gadget, it is equivalent to find a updating scheme changing a cell on a configuration (simulating a logical circuit) that to find a satisfying assignment to a boolean formula. Then AsyncStability in **NP**-complete for the rules 2 and 24 in its tri-dimensional version. This is the higher complexity for AsyncStability on freezing cellular automata.

The complexity of AsyncStability for the rules 2 and 24 in two dimension remains open. Also it should be interesting what happens in others neighborhoods or remove the totalistic hypothesis on the studied rules.

9.4 Final remarks and open problems

Despite their apparent limitation by the fact that each cell can only grow, freezing cellular automata show great diversity and complexity. That we can order these in the following way from the least complex to the most complex: 1D freezing cellular automata, 2D freezing cellular automata with von Neumann neighborhood and one change, and the rest of the freezing cellular automata.

- The 1D freezing cellular automata are the least complex since Stability is at most NLOGSPACE;

- the freezing cellular automata 2D von Neumann neighborhood and one change, because it has a freezing cellulare automaton where Stability is **P**-complete, but they do not have one intrinsically universal;
- the rest of freezing cellular automata where there is an intrinsically universal freezing cellular automata.

If we relax the synchronism in the update schemes, we obtain that most of the freezing cellular automata studied have a low complexity in dimension two, but if we study them in dimension three we obtain that there is one where the stability problem is **NP**-complete.

There are still some questions to be closed about in the development of the this thesis, such as:

- **to find an intrinsically universal freezing cellular automaton in dimension two with only 3 states (the minimum number of possible states).**

  In the construction of the intrinsically universal freezing cellular automata we use a large number of states but only two changes per cell. From this, we built a logic circuit system capable of simulating the local function of any freezing cellular automata. We already know a freezing cellular automata capable of simulating any Booblean function and in particular the local function of any freezing cellular automata. This automata is rule 2, but, as we already know, it has only one change and can not be intrinsically universal. The problem is that to cross information this automata uses a gadget built using XOR gate, where the input signals must arrive at the same time to simulate the crossing. After a signal passes through a crossing, it is unusable. This could be solved by adding a third state to rule 2 and thus obtain an intrinsically universal freezing cellular automata with only 3 states and von Neumann neighborhood.

- **to look for an intrinsically universal freezing cellular automaton for the class of the cellular automata of only one change**

  Since freezing cellular automata with only one change per cell and von Neumann neighborhood do not have an intrinsically universal automata, then we could study these automata as an independent family of cellular automata and see if there is a cellular automata with only one change capable of simulating any other freezing cellulare automata with only one change. A way to show that this is not possible would be to find a behavior that will "become more complex" as the number of states increases, so that it cannot be simulated by automata with fewer states, so any candidate to be intrinsically universal would have a automata with more states that it can not simulate.

- **to determinate the complexity of fractal rules:**

  To give an idea of the difficulty of these rules, we will analyze rule 1, but not using the von Neumann neighborhood, using the neighborhood to simulate the space-time diagram of 1D cellular automats in 2D cellular automatas.

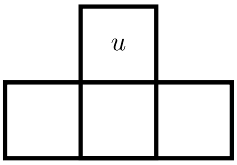


Figure 9.1: Neighborhood simulating on 2D grid a 1D first neighbors cellular automaton, where u is the center cell.

With this neighborhood rule 1 simulates the space-time diagram of the elementary cellular automata 22 (ECA 22). The ECM 22 has been classified by Wolfram as Class 3 [70], where class 4 contains the most complex automata. With this 1D version of rule 1 we can explore results about statistical studies of this rule [71, 72] or language theory [73].

Personal Bibliography

[1] D. Maldonado, A. Moreira, and A. Gajardo. Universal time-symmetric number-conserving cellular automaton. In J. Kari, editor, *Cellular Automata and Discrete Complex Systems - 21st IFIP WG 1.5 International Workshop, AUTOMATA 2015, Turku, Finland, June 8-10, 2015. Proceedings*, volume 9099 of *Lecture Notes in Computer Science*, pages 155–168. Springer Berlin Heidelberg, 2015.

[2] F. Becker, D. Maldonado, N. Ollinger, and G. Theyssier. Universality in freezing cellular automata. In F. Manea, R. G. Miller, and D. Nowotka, editors, *Sailing Routes in the World of Computation - 14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30 - August 3, 2018, Proceedings*, pages 50–59, Cham, 2018. Springer International Publishing.

[3] E. Goles, D. Maldonado, P. Montealegre, and N. Ollinger. On the computational complexity of the freezing non-strict majority automata. In *Cellular Automata and Discrete Complex Systems - 23rd IFIP WG 1.5 International Workshop, AUTOMATA 2017, Milan, Italy, June 7-9, 2017, Proceedings*, pages 109–119, 2017.

[4] E. Goles, D. Maldonado, P. Montealegre-Barba, and N. Ollinger. Fast-parallel algorithms for freezing totalistic asynchronous cellular automata. In G. Mauri, S. El Yacoubi, A. Dennunzio, K. Nishinari, and L. Manzoni, editors, *Cellular Automata - 13th International Conference on Cellular Automata for Research and Industry, ACRI 2018, Como, Italy, September 17-21, 2018, Proceedings*, pages 406–415, Cham, 2018. Springer International Publishing.

Bibliography

[5] M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, October 1970.

[6] J. Hardy, Y. Pomeau, and O. de Pazzis. Time evolution of a two-dimensional classical lattice system. *Phys. Rev. Lett.*, 31:276–279, Jul 1973.

[7] J. Hardy, O. de Pazzis, and Y. Pomeau. Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions. *Phys. Rev. A*, 13:1949–1961, May 1976.

[8] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, July 1961.

[9] A. Church. A Set of Postulates for the Foundation of Logic.

[10] E. L. Post. Formal reductions of the general combinatorial decision problem. *Am. J. Math.*, 65:197–215, 1943.

[11] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[12] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[13] J. V. Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Champaign, IL, USA, 1966.

[14] A. R. Smith. Simple computation-universal cellular spaces and self-reproduction. In *9th Annual Symposium on Switching and Automata Theory (swat 1968)(FOCS)*, volume 00, pages 269–277, 10 1968.

[15] K. Lindgren and M. G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4(3), 1990.

[16] M. Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.

[17] J. Albert and K. C. II. A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1(1), 1987.

[18] B. Durand and Z. Róka. *The Game of Life: Universality Revisited*, pages 51–74. Springer Netherlands, Dordrecht, 1999.

[19] N. Ollinger and G. Richard. Four states are enough! *Theor. Comput. Sci.*, 412(1-2):22–32, 2011.

[20] A. Moreira. Universality and decidability of number-conserving cellular automata. *Theoretical Computer Science*, 292(3):711 – 721, 2003. Algorithms in Quantum Information Prcoessing.

[21] J. Durand-Lose. Intrinsic universality of a 1-dimensional reversible cellular automaton. In R. Reischuk and M. Morvan, editors, *STACS*, volume 1200 of *Lecture Notes in Computer Science*, pages 439–450. Springer, 1997.

[22] K. Morita. Universality of one-dimensional reversible and number-conserving cellular automata. In E. Formenti, editor, *AUTOMATA JAC*, volume 90 of *EPTCS*, pages 142–150, 2012.

[23] A. Gajardo, J. Kari, and A. Moreira. On time-symmetry in cellular automata. *J. Comput. System Sci.*, 78(4):1115–1126, 2012.

[24] M. Delorme, J. Mazoyer, N. Ollinger, and G. Theyssier. Bulking I: an abstract theory of bulking. *Theoret. Comput. Sci.*, 412(30):3866–3880, 2011.

[25] M. Delorme, J. Mazoyer, N. Ollinger, and G. Theyssier. Bulking II: classifications of cellular automata. *Theoret. Comput. Sci.*, 412(30):3881–3095, 2011.

[26] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. 117:285–285, 05 1965.

[27] J. Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449 – 467, 1965.

[28] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[29] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[30] D. Griffeath and C. Moore. Life without death is P-Complete. Working papers, Santa Fe Institute, 1997.

[31] C. Moore. Majority-vote cellular automata, ising dynamics, and p-completeness. Working papers, Santa Fe Institute, 1996.

[32] T. Neary and D. Woods. P-completeness of cellular automaton rule 110. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Automata, Languages and Programming*, pages 132–143, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[33] E. Goles, N. Ollinger, and G. Theyssier. Introducing Freezing Cellular Automata. In *Cellular Automata and Discrete Complex Systems, 21st International Workshop (AUTOMATA 2015)*, volume 24 of *TUCS Lecture Notes*, pages 65–73, Turku, Finland, June 2015.

[34] J. Chalupa, P. L. Leath, and G. R. Reich. Bootstrap percolation on a bethe lattice. *Journal of Physics C: Solid State Physics*, 12(1):L31, 1979.

[35] T. Ghisu, B. Arca, G. Pellizzaro, and P. Duce. An improved cellular automata for wildfire spread. *Procedia Computer Science*, 51:2287 – 2296, 2015. International Conference On Computational Science, ICCS 2015.

[36] M. Fuentes and M. Kuperman. Cellular automata and epidemiological models with spatial dependence. *Physica A: Statistical Mechanics and its Applications*, 267(3):471–486, 1999.

[37] M. J. Patitz. An introduction to tile-based self-assembly. In J. Durand-Lose and N. Jonoska, editors, *Unconventional Computation and Natural Computation*, pages 34–62, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[38] P. Bak, K. Chen, and C. Tang. A forest-fire model and some thoughts on turbulence. *Physics Letters A*, 147(5-6):297 – 300, 1990.

[39] E. Goles, P. Montealegre-Barba, and I. Todinca. The complexity of the bootstraping percolation and other problems. *Theoretical Computer Science*, 504:73–82, 2013.

[40] S. Wolfram. Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55(3):601–644, July 1983.

[41] G. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Math. Syst Theory.*, 3(4):320–375, 1969.

[42] F. Robert. *Discrete iterations: a metric study.* Springer series in computational mathematics. Springer-Verlag, 1986.

[43] M. Sipser. *Introduction to the Theory of Computation.* Cengage Learning, Boston, MA, USA, 3 edition, 2012.

[44] J. JáJá. *An Introduction to Parallel Algorithms.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1992.

[45] R. Greenlaw, H. Hoover, and W. Ruzzo. *Limits to Parallel Computation: P-completeness Theory.* Oxford University Press, Inc., New York, NY, USA, 1995.

[46] J. JáJá and J. Simon. Parallel algorithms in graph theory: Planarity testing. *SIAM J. Comput.*, 11(2):314–328, 1982.

[47] E. R. Banks. Universality in cellular automata. In *SWAT (FOCS)*, pages 194–215. IEEE Computer Society, 1970.

[48] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[49] E. R. Banks. Information processing and transmission in cellular automata. Technical Report AITR-233, MIT Artificial Intelligence Laboratory, 1971.

[50] N. Ollinger and G. Richard. A particular universal cellular automaton. In *Proceedings International Workshop on The Complexity of Simple Programs, CSP 2008, Cork, Ireland, 6-7th December 2008.*, pages 205–214, 2008.

[51] A. Maruoka and M. Kimura. Condition for injectivity of global maps for tessellation automata. *Information and Control*, 32(2):158 – 162, 1976.

[52] K. C. II, J. Pachl, and S. Yu. On the limit sets of cellular automata. *SIAM Journal on Computing*, 18(4):831–842, 1989.

[53] K. Sutner. Model checking one-dimensional cellular automata. *J. Cellular Automata*, 4:213–224, 2009.

[54] J. Kari. The nilpotency problem of one-dimensional cellular automata. *SIAM J. Comput.*, 21(3):571–586, 1992.

[55] S. Kirkpatrick, W. W. Wilcke, R. B. Garner, and H. Huels. Percolation in dense storage arrays. *Physica A: Statistical Mechanics and its Applications*, 314(1):220 – 229, 2002. Horizons in Complex Systems.

[56] H. Amini. Bootstrap percolation in living neural networks. *Journal of Statistical Physics*, 141(3):459–475, Nov 2010.

[57] I. Karafyllidis and A. Thanailakis. A model for predicting forest fire spreading using cellular automata. *Ecological Modelling*, 99(1):87 – 97, 1997.

[58] J. H. Holland. A universal computer capable of executing an arbitrary number of subprograms simultaneously. In A. W. Bukrs, editor, *Essays on Cellular Automata*, pages 264–276. U. of Illinois Press, 1970.

[59] J. W. Thatcher. Universality in the von neumman cellular model. In A. W. Bukrs, editor, *Essays on Cellular Automata*, pages 132–186. U. of Illinois Press, 1970.

[60] S. Ulam and R. Schrandt. On recursively defined geometric objects and patterns of growth. Technical report, Los Alamos Scientific Laboratory, 1967.

[61] J. Gravner and D. Griffeath. Cellular automaton growth on z2: Theorems, examples, and problems. *Advances in Applied Mathematics*, 21(2):241 – 304, 1998.

[62] B. Bollobás, P. Smith, and A. Uzzell. Monotone cellular automata in a random environment. *Combinatorics, Probability and Computing*, 24(4):687–722, 2015.

[63] D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods. The tile assembly model is intrinsically universal. In *FOCS 2012 Proceedings*, pages 302–310, 2012.

[64] P. Meunier, M. J. Patitz, S. M. Summers, G. Theyssier, A. Winslow, and D. Woods. Intrinsic universality in tile self-assembly requires cooperation. In *SODA 2014 Proceedings*, pages 752–771, 2014.

[65] R. Vollmar. On cellular automata with a finite number of state changes. In W. Knödel and H.-J. Schneider, editors, *Parallel Processes and Related Automata*, volume 3 of *Computing Supplementum*, pages 181–191. Springer Vienna, 1981.

[66] D. A. Lind and B. Marcus. *An Introduction to Symbolic Dynamics and Coding.* Cambridge U. Press, New York, NY, USA, 1995.

[67] N. Ollinger. Universalities in cellular automata. In *Handbook of Natural Computing*, pages 189–229. Springer, 2012.

[68] L. M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–29, July 1977.

[69] E. Goles, P. Montealegre, K. Perrot, and G. Theyssier. On the complexity of two-dimensional signed majority cellular automata. *J. Comput. Syst. Sci.*, 91:1–32, 2018.

[70] *A New Kind of Science.* Wolfram Media Inc., Champaign, Ilinois, US, United States, 2002.

[71] J. G. Zabolitzky. Critical properties of rule 22 elementary cellular automata. *Journal of Statistical Physics*, 50(5):1255–1262, Mar 1988.

[72] P. Grassberger. Long-range effects in an elementary cellular automaton. *Journal of Statistical Physics*, 45(1):27–39, Oct 1986.

[73] J. Zhisong and W. Yi. Complexity of limit language of the elementary cellular automaton of rule 22. *Applied Mathematics-A Journal of Chinese Universities*, 20(3):268–276, Sep 2005.

Diego MALDONADO

Universalité et complexité des automates cellulaires coagulants

Résumé :
Les automates cellulaires forment une famille bien connue de modèles dynamiques discrets, introduits par S. Ulam et J. von Neumann dans les années 40. Ils ont été étudiés avec succès sous différents points de vue : modélisation, dynamique, ou encore complexité algorithmique. Dans ce travail, nous adoptons ce dernier point de vue pour étudier la famille des automates cellulaires coagulants, ceux dont l'état d'une cellule ne peut évoluer qu'en suivant une relation d'ordre prédéfinie sur l'ensemble de ses états. Nous étudions la complexité algorithmique de ces automates cellulaires de deux points de vue : la capacité de certains automates coagulants à simuler tous les autres automates cellulaires coagulants, appelée universalité intrinsèque, et la complexité temporelle de prédiction de l'évolution d'une cellule à partir d'une configuration finie, appelée complexité de prédiction. Nous montrons que malgré les sévères restrictions apportées par l'ordre sur les états, les automates cellulaires coagulants peuvent toujours exhiber des comportements de grande complexité.
D'une part, nous démontrons qu'en dimension deux et supérieure il existe un automate cellulaire coagulants intrinsèquement universel pour les automates cellulaires coagulants en codant leurs états par des blocs de cellules ; cet automate cellulaire effectue au plus deux changements d'états par cellule. Ce résultat est minimal en dimension deux et peut être amélioré en passant à au plus un changement en dimensions supérieures.
D'autre part, nous étudions la complexité algorithmique du problème de prédiction pour la famille des automates cellulaires totalistiques à deux états et voisinage de von Neumann en dimension deux. Dans cette famille de 32 automates, nous exhibons deux automates de complexité maximale dans le cas d'une mise à jour synchrone des cellules et nous montrons que dans le cas asynchrone cette complexité n'est atteinte qu'à partir de la dimension trois. Pour presque tous les autres automates de cette famille, nous montrons que leur complexité de prédiction est plus faible (sous l'hypothèse $\mathbf{P} \neq \mathbf{NP}$).
Mots clés: Universalité, Complexité, Automates Cellulaires Coagulants, Systèmes Dynamiques Discrets

Universality and complexity on freezing cellular automata

Abstract :
Cellular automata are a well know family of discrete dynamic systems, defined by S. Ulam and J. von Neumann in the 40s. The have been successfully studied from the point of view of modeling, dynamics and computational complexity. In this work, we adopt this last point of view to study the family of freezing cellular automata, those where the state of a cell can only evolve following an order relation on the set of states. We study the complexity of these cellular automata from two points of view, the ability of some freezing cellular automata to simulate every other freezing cellular automata, called intrinsic universality, and the time complexity to predict the evolution of a cell starting from a given finite configuration, called prediction complexity. We show that despite the severe restriction of the ordering of states, freezing cellular automata can still exhibit highly complex behaviors.
On the one hand, we show that in two or more dimensions there exists an intrinsically universal freezing cellular automaton, able to simulate any other freezing cellular automaton by encoding its states into blocks of cells, where each cell can change at most twice. This result is minimal in dimension two and can be even simplified to one change per cell in higher dimensions.
On the other hand, we extensively study the computational complexity of the prediction problem for totalistic freezing cellular automata with two states and von Neumann neighborhood in dimension two. In this family of 32 cellular automata, we find two automata with the maximum complexity for classical synchronous cellular automata, while in the case of asynchronous evolution, the maximum complexity can only be achived in dimension three. For most of the other automata of this family, we show that they have a lower complexity (assuming $\mathbf{P} \neq \mathbf{NP}$).
Keywords: Universality, Complexity, Freezing Celular Automata, Discrete Dynamical System.

**LIFO - Bâtiment IIIA, Rue Léonard de Vinci, B.P. 6759,
F-45067, ORLEANS Cedex 2**