Reconnaissance de langages par automates cellulaires

Véronique Terrier

Habilitation à diriger des recherches

présentée le 4 avril 2011 à

L'université de Caen Basse-Normandie

par

Véronique Terrier

Reconnaissance de langages par automates cellulaires

Rapporteurs

Yuri Gurevich, Microsoft Research / Université du Michigan
Nicolas Ollinger, Université de Provence
Laurent Vuillon, Université de Savoie

Jury

Christian Choffrut, Université Paris Diderot
Marianne Delorme, Université de Lyon
Étienne Grandjean, Université de Caen
Jacques Mazoyer, Université de Lyon
Nicolas Ollinger, Université de Provence
Laurent Vuillon, Université de Savoie
Jean-Baptiste Yunès, Université Paris Diderot

Sommaire

1 Introduction

Les automates cellulaires (AC) ont été introduits il y a une soixantaine d'années par von Neumann et Ulam qui cherchaient à définir les caractéristiques d'un système formel apte au calcul universel et à l'auto-reproduction. Leur utilité a été rapidement reconnue dans des domaines variés comme la physique et la biologie, pour modéliser des phénomènes complexes. Avec des travaux initiés par Hedlund, c'est en tant que classe particulière de systèmes dynamiques discrets qu'ils deviennent objet d'étude [20]. En plus de ces directions de recherche toujours très actives, les AC se sont vite imposés comme un modèle incontournable du calcul massivement parallèle. Les premières constructions algorithmiques ont mis en évidence, sur des diagrammes espace-temps, la richesse combinatoire et les possibilités remarquables d'organiser et de synchroniser l'information propres à ces machines parallèles [1, 39, 16]. Cette puissance de calcul a conduit à s'interroger sur les performances de ces machines et à chercher à mesurer leurs capacités et leurs limites. C'est ainsi que l'approche reconnaissance de langages a été développée et que l'étude des classes de complexité des AC a démarré [9, 3, 36, 37].

Mes travaux s'inscrivent dans ce dernier courant de recherche. Je m'intéresse aux questions de complexité sur les AC, avec une attention particulière aux petites classes de complexité : calcul en temps réel (i.e. temps minimal) et en temps linéaire ; en effet, c'est pour ces classes que la puissance de calcul est remarquable par rapport au mode séquentiel. Avec pour objectif de préciser la puissance de ce modèle et de mieux comprendre ce qu'est un calcul parallèle, trois tendances majeures se dégagent de mes recherches : l'étude des limites de ce modèle, la comparaison avec d'autres modèles de calcul et la question de l'influence de certains paramètres comme la dimension ou le voisinage sur ses capacités de reconnaissance.

Avant de présenter ces trois points, je vais rappeler quelques définitions.

2 Définitions

Une première qualité des AC est que leur structure est homogène et leur description est simple et bien formalisée.
Un *automate cellulaire* est un assemblage régulier de machines élémentaires (appelées cellules). Ces cellules sont disposées uniformément sur une ligne, un plan ou un espace de dimension quelconque et sont connectées mutuellement de façon locale et homogène. Elles fonctionnent comme des automates finis tous identiques qui évoluent en parallèle et de manière synchrone. Initialement, toute cellule est dans un état fixé. Puis à chaque étape, chacune des cellules met à jour son propre état conformément aux règles de transition de l'automate et des états reçus des cellules connectées.

Formellement, un automate cellulaire est identifié par un quadruplet $(d, Q, \mathcal{V}, \delta)$ où :

- d est la dimension de l'espace, les cellules sont alors indexées par $\mathbb{Z}^d$,
- Q est l'ensemble fini des états que peuvent prendre les cellules,
- $\mathcal{V} \subseteq \mathbb{Z}^d$ est un ensemble fini ordonné appelé voisinage qui spécifie les liens de communication,
- $\delta : Q^{\mathcal{V}} \to Q$ est la fonction de transition locale.

Si $\langle c, t\rangle$ dénote l'état de la cellule c au temps t, alors au temps suivant on a :
$\langle c, t+1\rangle = \delta\left(\langle c+v_1, t\rangle, \cdots, \langle c+v_n, t\rangle\right)$ où $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$.

Dépendant du *voisinage*, le déplacement des informations peut s'effectuer dans toutes les directions de l'espace comme avec le voisinage de von Neumann $\{v \in \mathbb{Z}^d : \Sigma|v_i| \leq 1\}$ et le voisinage de Moore $\{v \in \mathbb{Z}^d : \max|v_i| \leq 1\}$ ou être restreint sur une ou toutes les directions comme avec le voisinage unidirectionnel de von Neumann $\{-v : v \in \mathbb{N}^k \text{ et } \Sigma v_i \leq 1\}$ et le voisinage unidirectionnel de Moore $\{-v : v \in \mathbb{N}^k \text{ et } \max v_i \leq 1\}$.
On parle de *communication bidirectionnelle* pour les voisinages qui permettent de transmettre l'information partout dans l'espace et de *communication unidirectionnelle* pour les voisinages qui n'autorisent le déplacement de l'information que dans un sens pour chaque direction.

Les AC, grâce à leur plasticité, peuvent opérer sur des mots de dimension quelconque. D'ordinaire, les *données* traitées sont *unidimensionnelles* (des mots standards) ou *bidimensionnelles* (des images ou figures rectangulaires). Dans la perspective des AC en tant que *reconnaisseurs de langage*, il faut préciser d'une part comment les données sont entrées et d'autre part comment les résultats des calculs sont récupérés.
Par commodité, les symboles des mots d'entrée sont supposés faire partie de l'ensemble des états de l'automate. Les deux *modes d'entrée* usuels sont le mode parallèle et le mode séquentiel. Dans le *mode parallèle*, tous les symboles de l'entrée sont disposés conformément à la structure du mot sur les cellules de l'automate au temps initial. Avec des

données unidimensionnelles, le *mode séquentiel* consiste à choisir une cellule spécifique qui reçoit successivement les symboles de l'entrée.
Pour déterminer le résultat du calcul, une cellule est distinguée comme *cellule de sortie* et deux sous-ensembles d'états sont spécifiés : celui des *états d'acceptation* et celui des *états de rejet*. L'entrée de la cellule de sortie dans un de ces états finaux marque la terminaison du calcul. À noter que le choix de la cellule de sortie est arbitraire dans le cas des communications bidirectionnelles, mais, dans le cas des communications unidirectionnelles, cette cellule doit pouvoir accéder à tous les symboles de l'entrée.
Par suite, on dit qu'un AC *reconnaît* un langage L, si, sur toute entrée w, la cellule de sortie entre dans un état d'acceptation si $w \in L$ ou dans un état de rejet si $w \notin L$ à un certain temps t_f ; et pour tout temps $t < t_f$, la cellule de sortie n'est ni dans un état d'acceptation, ni dans un état de rejet.

On cherche naturellement à mesurer les ressources temps et espace consommées par le calcul d'un AC. La *complexité en temps* $t : \mathbb{N}^k \to \mathbb{N}$ d'un AC reconnaissant un langage de dimension k est la fonction définie par :

$$t(n_1, \cdots, n_k) = \max_{\substack{w \text{ mot de taille} \\ (n_1, \cdots, n_k)}} \{t : \text{ l'AC accepte ou rejette l'entrée } w \text{ en } t \text{ étapes}\}$$

On s'intéresse tout particulièrement aux petites complexités. En premier, la fonction *temps réel* $rt(n_1, \cdots, n_k)$ figure le temps minimal nécessaire à la cellule de sortie pour recevoir toutes les informations d'une entrée de taille $(n_1, \cdots, n_k)$. Cette fonction dépend des paramètres de l'AC (sa dimension, son voisinage, son mode d'entrée, la position de sa sortie) et spécifie la borne inférieure de temps consommé par ce type d'AC. Autre fonction de petite complexité, la fonction *temps linéaire* correspond à la fonction temps réel multipliée par une constante strictement supérieure à 1.
L'espace consommé $s : \mathbb{N}^k \to \mathcal{P}(\mathbb{Z}^d)$ par un AC de dimension d reconnaissant un langage de dimension k est la fonction définie par :

$$s(n_1, \cdots, n_k) = \bigcup_{\substack{w \text{ mot de taille} \\ (n_1, \cdots, n_k)}} \{c : \text{ la cellule } c \text{ participe au calcul de l'AC sur l'entrée } w\}$$

Par commodité, on ne considère dans la suite que des AC dont le calcul est borné en espace minimal. Cet *espace minimal* dépend des caractéristiques de l'AC et correspond à l'ensemble des cellules utilisées par les calculs en temps réel sur ce type d'AC.

Associées à un type d'AC, les *classes de complexité temps/espace* identifient les familles de langages reconnus par ce type d'AC en espace minimal et en temps inférieur à une certaine fonction. Vu les variétés d'automates, il est facile de s'égarer dans les multiples classes de langages qui en dérivent. Dans la suite, les classes de langages bidimensionnels examinées seront mentionnées avec toutes leurs caractéristiques. Et pour les classes de

langages unidimensionnels, on utilisera les notations suivantes.
On distingue quatre types classiques d'AC unidimensionnels opérant sur des langages unidimensionnels. Selon le mode d'entrée et le voisinage choisis, ils sont notés PCA, POCA, SCA, SOCA où la première lettre "P" ou "S" figure le mode d'entrée (parallèle ou séquentiel) et l'occurrence de "O" (comme one-way) indique si le voisinage est uni- ou bi-directionnel. Leur extension à des réseaux de dimension arbitraire k sont notés k-PCA, k-POCA, k-SCA, k-SOCA.

Étant donné un type d'automate X $\in$ {PCA, POCA, SCA, SOCA, k-PCA, k-POCA, k-SCA, k-SOCA, ...}, on définit les classes de complexité temps/espace suivantes.

- X(t) : la classe des langages unidimensionnels reconnus en espace minimal et temps au plus t par des AC de type X.

Les classes les plus significatives sont celles des langages reconnus en temps réel et en temps linéaire (et a fortiori en espace minimal).

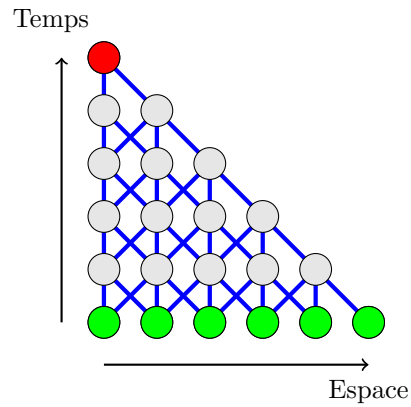
- RX : la classe temps réel des AC de type X.
- LX : la classe temps linéaire des AC de type X.

À l'autre extrémité, la classe des langages reconnus en temps quelconque est notée comme le type afférent.

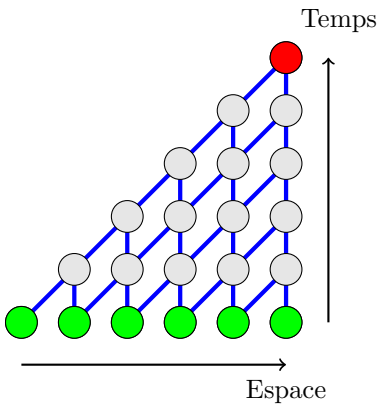
- X : la classe des langages reconnus en espace minimal par des AC de type X.

À noter que pour les types d'AC avec communication bidirectionnelle, ces classes de complexité en espace coïncident avec celles des machines de Turing. En particulier, PCA est la classe espace linéaire des machines de Turing déterministes.

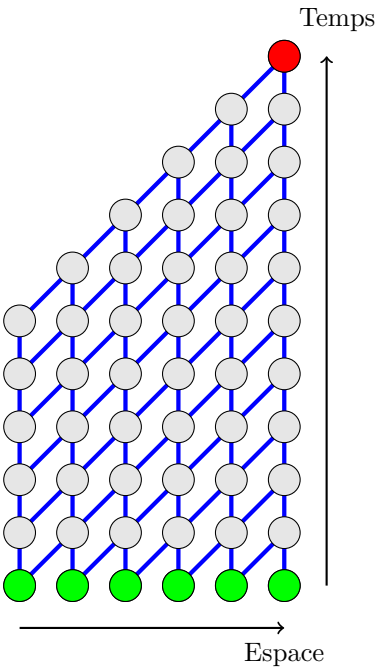
Une notion clé est celle de *graphes de dépendances*. Ces graphes reflètent la structure spatio-temporelle spécifique à un type d'AC qui opère avec une complexité déterminée (voir Figure 1). Nombre de simulations et limitations connues pour les classes d'AC se traduisent géométriquement grâce aux graphes de dépendances. Par nature, un graphe de dépendances est orienté et sans cycle. Les sommets sont les sites qui participent au calcul et les arcs figurent les dépendances induites par le voisinage entre les sites.



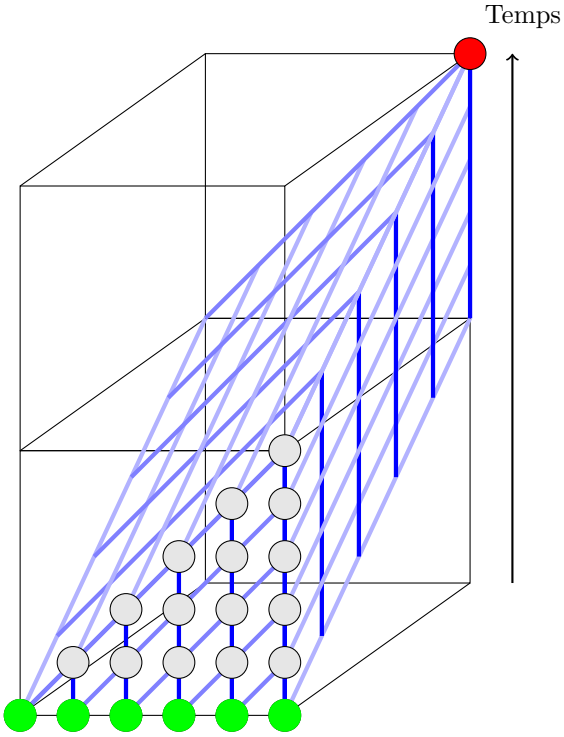
(a) RPCA : temps réel, entrée parallèle, voisinage bidirectionnel $\{-1, 0, 1\}$



(b) RPOCA : temps réel, entrée parallèle, voisinage unidirectionnel $\{-1, 0\}$



(c) LPOCA : temps linéaire, entrée parallèle, voisinage unidirectionnel $\{-1, 0\}$



(d) 2-RPOCA : espace de dimension 2, temps réel, entrée parallèle, voisinage unidirectionnel $\{(-1, 0), (0, -1), (0, 0)\}$

Figure 1: Graphes de dépendances

3 Préciser les limites du modèle

Notre ignorance est grande lorsqu'il s'agit de séparer les classes de complexité des AC. En premier, la question de savoir si les AC travaillant en espace linéaire (et temps quelconque) sont plus puissants que les AC travaillant en temps réel, reste piteusement sans réponse. Or une réponse positive à cette question est fortement improbable puisqu'une des conséquences est l'égalité de P et PSPACE. Mais à l'inverse, on ne sait pas si une réponse négative implique l'inclusion stricte de P dans PSPACE, ce qui justifierait au moins les difficultés rencontrées. Et comme pour tous les autres modèles de calcul, les outils font défaut pour établir des hiérarchies strictes sur les classes de complexité lorsqu'on ne fait varier qu'une ressource (ici le temps) et en laissant fixe une autre ressource (ici l'espace).
D'autres questions plus spécifiques aux AC comme celle de déterminer si en dimension 1 les AC avec communication unidirectionnelle ont la même puissance que ceux avec communication bidirectionnelle, sont aussi bien loin d'être résolues.
Pour ne pas s'arrêter sur un constat d'échec, le mieux est de simplifier le problème en examinant des variantes restreintes d'AC. On dispose alors d'outils combinatoires qui permettent d'établir des limites à la capacité de reconnaissance de ces AC.

3.1 Arguments combinatoires

Avec des variantes restreintes d'AC, les contraintes structurelles relatives au déplacement des données empêchent de disposer en temps utile de l'information nécessaire pour réaliser les calculs. Pour dériver des limites sur les capacités de reconnaissance de ces variantes, la stratégie est alors directe et consiste, en jouant sur ces contraintes, à exhiber des langages *ad hoc*. La mise en œuvre de cette stratégie qui dépend du graphe de communication spécifique, est redéfinie pour chaque variante particulière.
Illustrons la démarche usuelle par un exemple simple. Considérons un AC de dimension 2 avec entrée parallèle et voisinage de Moore unidirectionnel $\mathcal{V} = \{(0,0),(-1,0),(-1,-1),(0,-1)\}$ et qui opère sur des langages bidimensionnels. La cellule de sortie est la cellule $(0,0)$ et pour des entrées carrées de taille (n,n), le temps réel est $n-1$. Conséquence des contraintes sur le flux des données, il n'existe aucune interaction entre la première et la dernière ligne avant ce temps $n-1$. On peut alors identifier un goulot d'étranglement lorsqu'il ne reste plus que de l'ordre de $o(n)$ étapes pour terminer le calcul. Observons à cet effet la situation au temps $n-2$ en supposant que le calcul se poursuit encore pendant k étapes (voir Figure 2). Au temps $n-2$, seul le carré constitué des cellules (i,j) avec $0 \leq i,j \leq k$ peut influencer la cellule de sortie $(0,0)$ au cours des k étapes suivantes. Ce carré se partage en deux selon l'impact respectif de la première et de la dernière ligne de l'entrée : les $k+1$ cellules du bord nord dépendent de la première mais pas de la dernière ligne et inversement les autres cellules dépendent de la dernière mais pas de la première ligne. Aussi les n données élémentaires qui composent la première ligne de l'entrée, se trouvent compressées sur $k+1$ cellules. En d'autres termes, une partie significative de l'information est perdue avant que la première et dernière ligne interagissent si k est de l'ordre de $o(n)$.

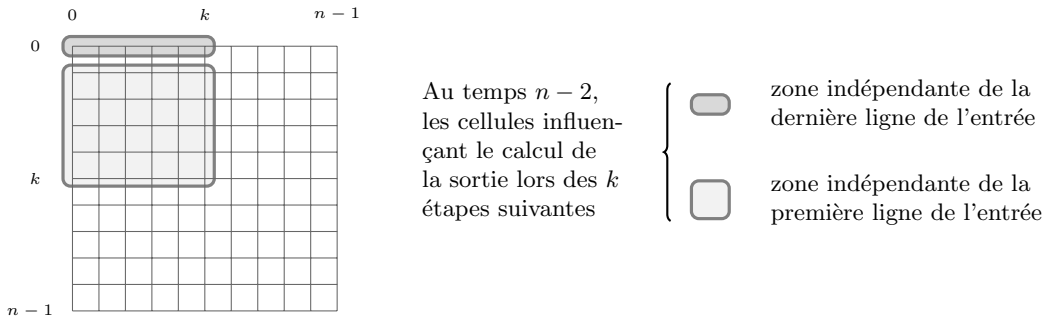


Figure 2: État des lieux au temps $n-2$

Il est ensuite facile d'expliciter des langages non reconnus par AC opérant en temps réel avec voisinage de Moore unidirectionnel. Pour reconnaître la famille des images qui ont même première et même dernière ligne, le calcul nécessite au moins $n+O(n)$ étapes sur des entrées de taille (n,n). Ou bien, pour le langage majorité (i.e. la famille des images binaires qui ont plus de 1 que de 0), le calcul nécessite au moins $n+O(\log n)$ étapes sur des entrées de taille (n,n).
Reste alors à mettre en musique la preuve formelle. On utilise alors soit des techniques algébriques (qui prennent en compte toute une famille d'entrées) soit, de façon plus élégante, la complexité de Kolmogorov (qui se focalise sur une entrée spécifique de nature "complexe") [6].

Cole a été le premier à appliquer ces arguments combinatoires aux AC [9]. Il a explicité des limites sur l'interaction des données dans le cas où le mode d'entrée est séquentiel et a présenté des langages qui ne sont reconnus par aucun SCA en temps réel. Vu que ces langages sont reconnus en temps réel par PCA, la capacité de reconnaissance des RSCA est moindre que celle des RPCA. De même, divers résultats négatifs sont obtenus portant, entre autres, sur les propriétés de clôture par concaténation ou par miroir. Une autre proposition importante soulignée par Cole stipule que la capacité de reconnaissance des RSCA augmente avec la dimension du réseau.

Dans le cas des voisinages restreints, il est facile d'observer que les RPOCA sur des langages dont l'alphabet est unaire, sont équivalents aux automates finis et par suite moins puissants que les RPCA [10]. Lors de mes travaux, j'ai examiné plus en détail les limites de cette classe de complexité.
Dans une première approche, j'ai tiré partie d'une propriété des RPOCA mise en évidence par Čulik [11] à savoir que le calcul de n'importe quel mot englobe aussi tous les calculs des facteurs de ce mot. Cette caractéristique impose certaines contraintes sur la structure des langages reconnus par les RPOCA. Jouant sur ces contraintes, j'ai exhibé un langage qui n'est reconnu par aucun RPOCA [45]. Il s'agit du langage suivant : $L = L_1L_1$ où $L_1 = \{w \in \{0,1\}^* : w = 1^u0^u$ ou $w = 1^u0y10^u$ avec $y \in \{0,1\}^*$ et $u > 0\}$. Du fait que

ce langage est à la fois algébrique et carré d'un langage reconnu par RPOCA, j'ai par la même occasion répondu par la négative à deux questions ouvertes : la classe temps réel des POCA n'est pas close par concaténation et ne contient pas tous les langages algébriques.
Dans une seconde approche, j'ai mis en évidence une autre contrainte relative à la propagation de l'information : l'interaction entre le début et la fin des entrées est très restreinte sur les RPOCA. L'ensemble des mots de la forme uvu fournit ainsi un exemple-type de langage non reconnu par RPOCA [46]. Par la suite, cette deuxième approche a été réutilisée par Klein *et al* [27] pour montrer l'existence d'une hiérarchie infinie (stricte) entre les classes temps réel et temps linéaire des POCA.

Grâce encore à ces techniques algébriques, j'ai explicité des bornes inférieures sur les AC de dimension 2 opérant sur des langages bidimensionnels et ceci pour différents types de voisinage.
À commencer par le voisinage de Moore [50]. J'ai exhibé un langage d'images qui n'est reconnu en temps réel par aucun AC avec voisinage de Moore. En revanche, ce langage est reconnu en temps réel avec le voisinage de von Neumann et son image par rotation à 180^o l'est également en temps minimal avec le voisinage de Moore. Conséquence, la classe temps réel des AC avec voisinage de Moore n'est pas close par rotation à 180^o. De surcroît, elle ne contient pas la classe temps réel des AC avec voisinage de von Neumann.
À toutes fins utiles, il faut rappeler que si le voisinage de Moore est plus étendu que le voisinage de von Neumann et permet à priori une rapidité de traitement accrue, le temps minimal nécessaire à la cellule de sortie pour connaître l'entrée complète, diminue également. De fait, le temps réel avec voisinage de Moore qui est $\max(m, n)$ sur une entrée de taille (m, n), est inférieur au temps réel avec voisinage de von Neumann qui est $m + n$. Il resterait maintenant à déterminer si tout ce qui est reconnu en temps réel avec Moore l'est aussi en temps réel avec von Neumann.
Pour les voisinages unidirectionnels $\mathcal{V}$ tels que $a+b \geq 0$ pour tout (a, b) de $\mathcal{V}$, j'ai présenté des langages bidimensionnels qui sont reconnus en temps réel aussi bien avec voisinage de von Neumann que voisinage de Moore mais qui ne le sont même pas en temps linéaire avec ces voisinages unidirectionnelles $\mathcal{V}$. Enfin j'ai démontré que les classes temps réel et temps linéaire pour les variantes unidirectionnelles de Moore et von Neumann ne sont pas closes par rotation [55, 56].

En collaboration avec J. Mazoyer puis J.C. Dubacq, j'ai également obtenu des résultats de lacune sur la constructibilité des signaux par AC en dimension 1 et en dimension 2 [49, 51]. Ces résultats, basés aussi sur des techniques combinatoires, ont leur pendant en terme de reconnaissance de langages. Ainsi le fait qu'un signal constructible par un AC en dimension 1 soit se déplace à vitesse maximale, soit prend un retard au moins logarithmique, se traduit par une lacune entre le temps n et le temps $n + \log n$ pour la reconnaissance des langages unaires sur POCA.

3.2 Relation avec des propriétés de clôture

Une hypothèse souvent avancée est que le temps réel et le temps linéaire définissent des classes distinctes. Cette conjecture se montre difficile à prouver aussi il est intéressant de trouver des parallèles avec d'autres interrogations. Dans le cas de la dimension 1, Ibarra et Jiang ont ainsi établi un lien entre la capacité de reconnaissance de la classe temps réel et ses propriétés de clôture : ils ont montré l'équivalence entre la clôture par miroir de la classe temps réel et l'égalité des classes temps réel et temps linéaire [23]. Traduit en terme d'AC avec voisinage unidirectionnel, ce critère de clôture conforte l'hypothèse que les AC en temps linéaire sont plus puissants que les AC en temps réel (voir Annexe A Section 6.2). Leur résultat s'appuie sur un algorithme très astucieux mais inclut un détour inutile par des machines séquentielles. Mieux vaut lire la description directe en terme d'AC de M. Delorme et J. Mazoyer [12].

J'ai étendu cet algorithme au cas de la dimension 2 pour des AC avec voisinage de von Neumann en prenant comme propriété de clôture la rotation à 180^o [53]. Ainsi, avec le voisinage de von Neumann, les classes temps réel et temps linéaire sont confondues si et seulement si la classe temps réel est close par rotation à 180^o. Par manque d'outils adaptés, la preuve se limite à des techniques de géométrie élémentaire plutôt fastidieuses.

Dans une nouvelle tentative pour comprendre comment la façon dont l'entrée est fournie influe sur les capacités de calcul, et avec comme base toujours l'algorithme de Ibarra et Jiang, j'ai explicité d'autres correspondances en dimension 1 [56]. Premièrement, on a l'équivalence entre clôture par cycle de la classe temps réel et égalité des classes temps réel et temps linéaire. Deuxièmement, pour les langages sur un alphabet unaire, on a le parallèle avec la clôture par concaténation. Par suite, les propriétés de clôture par miroir et par cycle coïncident pour la classe temps réel. Mais, tandis qu'il est facile de vérifier que la classe temps linéaire est close par miroir, rien n'indique qu'elle le soit par cycle.

4 Jouer avec les paramètres

Une approche naturelle pour mieux comprendre les ressorts de ce modèle est de jouer sur ses paramètres, notamment la dimension du réseau qui définit la répartition des cellules dans l'espace et le voisinage qui spécifie les interactions au niveau local des cellules. Or augmenter la dimension du réseau ouvre de nouvelles modalités pour entrer les données et autorise à augmenter également la dimension des données. Aussi, les façons de spécifier les AC multidimensionnels sont diverses.

Première possibilité, on augmente la dimension de l'espace sans augmenter la dimension des données. Les modèles ainsi définis restent des reconnaisseurs de langages unidimensionnels. De multiples variantes sont alors envisageables selon le mode d'entrée choisi. Concernant les premières généralisations proposées dans la littérature, augmenter la dimension revient implicitement à augmenter l'espace. Ces variantes utilisent sur les entrées de longueur n une grille multidimensionnelle de taille n en chacune des dimensions. L'espace borné d'un AC de dimension d est ainsi défini comme un réseau de n^d cellules où le mot d'entrée est fourni soit en séquentiel sur une cellule, soit en parallèle sur une des dimensions. À noter que si l'espace augmente avec la dimension, les complexités temps réel et temps linéaire restent en revanche linéaires en la taille de l'entrée quelque soit la dimension.

Avec le mode d'entrée parallèle, une autre perspective est de placer l'entrée de façon compacte. Ainsi pour un AC de dimension d, les mots de longueur n sont arrangés dans un cube de dimension d et de côté $\lceil \sqrt[d]{n} \rceil$. Ce cube délimite l'espace de l'AC, autrement dit, l'espace utilisé équivaut à la taille de l'entrée quelque soit la dimension. Par contre, les complexités temps réel et temps linéaire, qui sont de l'ordre de $\sqrt[d]{n}$, diminuent avec la dimension. Maintenant, il existe de multiples façons d'arranger de façon compacte un mot dans un cube de dimension d, d'où il résulte de multiples classes de complexité spécifiques. L'ennui est qu'aucune n'est plus légitime que les autres. De plus, l'ordre de placement est en général trop dépendant de la longueur de l'entrée.

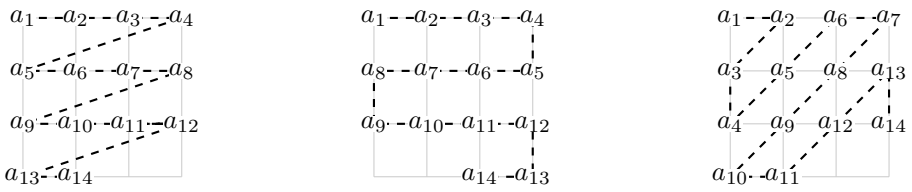


Figure 3: Divers arrangements d'un mot dans un carré

Pour éviter entre autre ce dernier défaut, une piste proposée par M. Delorme et J. Mazoyer consiste à utiliser des courbes qui remplissent l'espace (fil d'Archimède, fil de Hilbert ...). L'automate est alors spécifié avec un paramètre supplémentaire : le fil qui est mémorisé localement par chaque cellule. Et au temps initial, le mot d'entrée est

disposé le long de ce fil.

Deuxième possibilité, on augmente simultanément la dimension de l'espace et la dimension des données. C'est dans le cadre de la dimension 2, où les AC opèrent comme reconnaisseurs de langages d'images, que cette approche est la plus répandue. De fait, cet intérêt particulier pour la dimension 2 est à la fois encouragé, sur le versant applicatif, par l'aspect traitement d'images et est également stimulé par les nombreux travaux qui généralisent la notion de langage rationnel à la dimension 2. Le calcul sur des données bidimensionnelles engendre des interactions plus riches et le choix du voisinage apparaît comme un des paramètres significatifs qui influent sur les capacités de reconnaissance de tels AC.

4.1 Reconnaissance de langages unidimensionnels par des AC de dimension quelconque

Il s'agit ici des variantes d'AC à d dimensions qui sur les entrées de longueur n opèrent sur un espace de n^d cellules. Pour les AC avec mode d'entrée séquentielle et voisinage bidirectionnel, Cole a montré que la capacité de reconnaissance en temps réel de ces automates augmente strictement avec la dimension du réseau [9]. La situation n'est pas si tranchée lorsqu'on examine les autres variantes (AC avec entrée séquentielle et communication unidirectionnelle ou AC avec entrée parallèle et communication uni ou bidirectionnelle). Dans [57], j'ai présenté une étude systématique des variantes avec voisinage de von Neumann unidirectionnel et entrée séquentielle (SOCA) ou entrée parallèle (POCA) et de leurs classes de petite complexité. Chose amusante, la hiérarchie en dimension de ces classes temps réel et temps linéaire s'inscrit entre les classes temps minimal et temps quelconque (bornés linéairement en espace) de la hiérarchie en temps des AC de dimension 1. Voir Figure 4.
J'ai ainsi généralisé les résultats d'accélération linéaire, d'inclusion ou d'égalité entre classes connus dans le cas de la dimension 1 aux dimensions supérieures. Essentiellement, c'est la géométrie induite par la régularité des graphes de dépendances qui entre en jeu. En particulier, les simulations sont basées sur des transformations affines qui préservent les contraintes relatives à la fois au voisinage (les dépendances sont respectées), à la charge de calcul (l'application linéaire sous-jacente est injective) et aux entrées sorties (la correspondance est garantie).
Un phénomène particulier se présente pour les propriétés de clôture des RPOCA. À l'inverse de la dimension 1, les RPOCA de dimension supérieure sont clos par concaténation et par étoile de Kleene.

Sans surprise, la question de savoir si, pour ces modèles, augmenter la dimension augmente strictement les capacités de calcul est ouverte. Et les relations entre la hiérarchie en temps des AC de dimension 1 et cette hiérarchie en dimension sont pour le moins énigmatiques. Les graphes de dépendances d'un AC de dimension 1 travaillant en temps n^k et d'un AC de dimension k travaillant en temps linéaire ont tous les deux de l'ordre de n^{k+1} sites, mais n'ont pas du tout la même structure. Bref, toutes les questions relatives

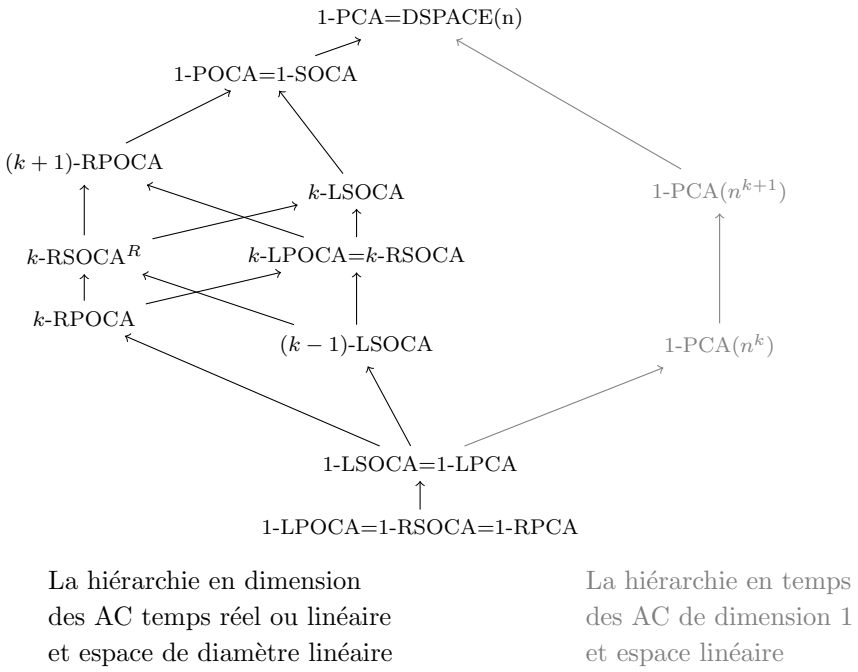


Figure 4: Relations entre diverses classes de langages unidimensionnels. Aucune inclusion stricte n'est connue entre 1-RPCA et 1-PCA.

aux relations temps et espace restent en suspens.

4.2 Le choix du voisinage

Une des caractéristiques des AC est que la transmission de l'information s'effectue de manière locale. C'est le voisinage choisi qui définit le schéma de communication et deux voisinages distincts n'offrent pas systématiquement les mêmes capacités algorithmiques. Ainsi en dimension 1, le voisinage de premiers voisins $\{-1, 0, 1\}$ et le voisinage unidirectionnel $\{-1, 0\}$ ne déterminent pas les mêmes classes de complexité. En réalité, tout voisinage (non pathologique) est équivalent à l'un de ces deux voisinages (de premiers voisins ou unidirectionnel) comme le montre V. Poupet par son résultat de dichotomie [32].
En dimension supérieure, la structure spatio-temporelle définie par chaque voisinage devient plus intriquée et les interactions sont plus complexes. Dès la dimension 2, l'existence de langages d'images reconnus en temps réel avec voisinage de von Neumann sans l'être avec voisinage de Moore indique que l'état de choses est plus varié qu'en dimension 1. On a là deux voisinages bidirectionnels dont les classes temps réel sont distinctes. Dans [54, 55], j'ai examiné plus avant les questions de voisinage en dimension 2 et leur impact sur les capacités de reconnaissance de langages d'images. Une des caractéristiques mise en évidence est l'enveloppe convexe qui délimite le voisinage. Pour

deux voisinages distincts qui partagent la même enveloppe convexe, tout calcul réalisé avec l'un peut être également réalisé avec l'autre et ceci sans perte de temps hormis une phase préliminaire de coût constant. J'ai présenté aussi certains exemples où l'on peut réduire la taille du voisinage tout en gardant les mêmes capacités de reconnaissance. J'ai également obtenu des résultats d'accélération linéaire pour une famille large (mais incomplète) de voisinages.

5 Comparer les AC avec d'autres modèles

Une autre approche pour déterminer les avantages apportés par le modèle des AC est de comparer ses performances avec celles d'autres modèles de calcul.

5.1 Automates alternants

Le calcul alternant qui combine branchement existentiel et branchement universel est de nature parallèle. On trouve ainsi diverses caractérisations des AC en terme d'automates finis alternants. Une première correspondance a été mise en évidence par Ito *et al* [25]. Ils ont montré qu'une variante d'AC (les DOTA) reconnaît la même famille de langages bidimensionnels que les automates finis alternants de dimension 2 muni d'une tête de lecture qui se déplace dans une seule direction sur chacune des deux dimensions.

De manière similaire, j'ai établi que les AC de dimension k avec entrée séquentielle et voisinage ou bidirectionnel ou unidirectionnel, ont leur équivalent en terme d'automates alternants. J'ai obtenu les deux résultats suivants. D'une part, les automates itératifs de dimension k travaillant en temps réel (les k-RSCA) sont équivalents (modulo l'opération miroir sur les entrées) aux automates finis alternants avec une tête unidirectionnelle, k compteurs et travaillant en temps réel [52]. D'autre part, les AC de dimension k avec entrée séquentielle, voisinage unidirectionnel et travaillant en temps réel (les k-RSOCA) sont équivalents (modulo l'opération miroir) aux automates finis alternants à $k+1$ têtes unidirectionnelles [57]. De la sorte, on dispose d'une tête pour chacune des k dimensions et d'une tête pour le temps. En d'autres termes, lorsque les communications sont unidirectionnelles, le temps qui par essence est unidirectionnel équivaut à une dimension de l'espace.

Les simulations présentées pour prouver ces deux résultats sont analogues. Elles consistent, par duplication, à transformer le graphe de dépendances de l'AC en arbre de dépendances. Reste alors à constater que les structures de l'arbre de dépendances de l'AC et de l'arbre des calculs de l'automate fini alternant sont en fait identiques.

Ces simulations impliquent des AC avec entrée séquentielle. Concernant le mode d'entrée parallèle, une de ses particularités est comme le note Jacques Mazoyer, qu'il induit une synchronisation implicite au temps initial. Autrement dit, une description des AC avec entrée parallèle serait envisageable au moyen d'automates synchronisant (cf. [22]). À ce propos, il est bien de rappeler le résultat d'Okhotin qui fait intervenir des grammaires alternantes et propose un lien intéressant avec la hiérarchie de Chomsky. C'est la caractérisation des RPOCA par des grammaire linéaires conjonctives (i.e. des grammaires linéaires enrichies d'un opérateur de conjonction) [31].

Mais quels enseignements peut on tirer de ces correspondances avec ces variantes d'automates finis alternants ? Avoir différentes définitions d'un même objet est toujours intéressant. Cependant, il faut reconnaître que ces automates finis alternants n'apportent pas vraiment d'éclairage nouveau sur les AC. D'une part, sur le versant algorithmique,

les modèles alternants apparaissent plutôt barbares. Imaginer par exemple l'automate itératif de Fischer [16] (qui reconnaît en temps réel les mots de longueur premier) sous la forme d'un automate fini alternant muni d'une tête unidirectionnelle et d'un compteur. D'autre part, pour l'étude des limites des AC, ces modèles alternants ne fournissent pas, pour l'instant, de meilleure compréhension des AC. On sait grâce à Cole [9] que les capacités de reconnaissance des automates itératifs augmentent avec la dimension du réseau, autrement dit que la hiérarchie des k-RSCA est stricte selon la dimension k. La redécouverte dans [24] que cette hiérarchie est stricte pour les automates alternants à k compteurs, ne nous apprend rien de plus. Concernant la hiérarchie des k-RSOCA, savoir si elle est stricte est une question ouverte (cf. Section 4.1) et se traduit en question tout aussi ouverte sur la hiérarchie des automates finis alternants avec têtes unidirectionnelles.

5.2 Autres modèles de calcul massivement parallèles

D'autres modèles de calcul massivement parallèles que les automates cellulaires existent! Les plus classiques : circuits booléens, PRAM et machines de Turing alternantes, suscitent une littérature abondante mais difficile d'y trouver référence aux AC. Pourtant des relations directes apparaissent notamment avec les circuits booléens. En premier, les AC peuvent être considérés comme un type particulier de circuits booléens. Inversement, le premier AC universel (en dimension 2) a été construit via une simulation des circuits booléens [2].

Mais l'aspect complexité est plutôt négligé. Plusieurs relations concernant les classes de complexité ont bien été établis entre les machines de Turing alternantes et une catégorie particulière d'AC travaillant sur des réseaux arborescents. Mais ces AC arborescents, contrairement aux AC classique (sur les réseaux $\mathbb{Z}^d$), ne sont pas des modèles réalistes. Ils reconnaissent en temps minimal des problèmes NP-complets.

En réalité, les AC classiques et les autres modèles de calcul massivement parallèles sont, par leur modalité de communication, de nature différente. Pour les AC, les réseaux sont à croissance bornée : étant donnée une cellule, le nombre de cellules accessibles en $2t$ étapes à partir de cette cellule est linéairement proportionnel au nombre de cellules accessibles en t étapes. En d'autres termes, le taux d'expansion est constant. À l'inverse, et sans même parler de la majorité des PRAM qui se fichent des questions de communication, il n'existe pas de telles restrictions sur la propagation de l'information pour les circuits booléens ou les machines de Turing alternantes (ou les AC arborescents). Le taux d'expansion peut même être exponentiel.
Avec cette divergence, mimer de manière efficace le comportement de machines à croissance non bornée par des AC n'est pas simple. À ma connaissance, il n'existe qu'un seul algorithme non trivial à ranger dans cette catégorie. C'est celui de Chang *et al* [7] et qui a une conséquence intéressante : toute machine de Turing alternante travaillant en temps linéaire peut être simulée par un POCA.

L'opération inverse qui consiste à simuler efficacement le comportement des AC avec d'autres modèles parallèles, est aussi peu explorée. Malgré tout, j'ai construit pour les AC avec communication unidirectionnelle, une simulation par des circuits booléens [59].

À dire vrai, ma préoccupation initiale n'avait pas trait aux circuits booléens mais touchait aux questions de communication unidirectionnelle versus communication bidirectionnelle. Pour les POCA, l'interaction entre les cellules est limitée : l'ensemble du calcul sur une cellule dépend bien du calcul réalisé sur sa cellule voisine mais à l'inverse n'a aucun effet sur lui. Concrètement, les états successifs d'une cellule sont l'image par une transduction fonctionnelle des états successifs de sa voisine. Autrement dit, chaque cellule d'un POCA agit comme un transducteur et un POCA qui travaille en espace n opère comme une succession de n transducteurs. En cherchant à exploiter cette caractéristique, j'ai déniché un résultat de Ladner et Fischer présentant une simulation efficace des transducteurs par des circuits booléens : si le transducteur opère sur un mot de longueur s, la profondeur du circuit qui le mime est en $\log s$ [33]. En empilant n copies du circuit de Ladner et Fischer, on simule alors un POCA qui travaille en espace n et temps $t(n)$ par un circuit de profondeur $n \log(t(n))$.
Une astuce permet ensuite de compresser localement le circuit initial et apporte une amélioration qui n'est significative que pour les petites complexités, avec une profondeur en $n(1 + \log(t(n)/n))$. Même si le gain obtenu par cette compression n'est pas terrible, on ne connaît pas, pour le moment, de simulation plus efficace.

5.3 Modèles séquentiels

Les relations entre calcul parallèle et calcul séquentiel sont pour sûr un problème majeur. Une première question est de savoir si le gain apporté par le parallélisme est toujours significatif. Si le nombre de cellules impliquées dans un calcul parallèle induit une borne sur l'accélération maximale espérée d'un calcul séquentiel, il ne garantit en revanche aucune accélération minimale. Plus avant, l'enjeu pratique est de déterminer des transformations efficaces qui permettent de traduire automatiquement un calcul séquentiel en un calcul parallèle. Mais cet objectif est difficile à réaliser. Le contraste est grand entre d'une part, tous les exemples d'AC construits pour résoudre des problèmes spécifiques et qui illustrent le gain apporté par le parallélisme, et d'autre part, les simulations connues de modèles séquentiels par AC et qui sont absolument inefficientes en regard de l'accélération obtenue. Ainsi pour la simulation des machines de Turing (modèle de calcul local comme les AC), on ne connaît pas actuellement de meilleur résultat que celui de Smith [36] où une étape de calcul de la machine de Turing est mimée par une étape de calcul sur l'AC. Aucune accélération n'a non plus été établie pour des variantes restreintes de machines de Turing.

À ce titre, le résultat de Kosaraju portant sur les automates finis de dimension 2 est remarquable [28]. Bref rappel, un automate fini de dimension 2 est un automate qui opère sur des images et où la tête de lecture se déplace selon les 4 directions N, E, S ou W. Un tel automate sur une image de taille (n, m) peut (sans boucler) effectuer de l'ordre de $(n + m)^2$ pas. Or la simulation de ces automates finis par AC, proposée par Kosaraju, permet de réaliser le même traitement en $O(n + m)$ étapes sur AC, ce qui est une accélération significative. Proprement dit, tout langage d'images reconnu par un automate fini (déterministe ou non) de dimension 2 est reconnu en temps linéaire par

AC de dimension 2. Kosaraju ne fournit cependant dans son article qu'une ébauche de preuve. Par la suite, Fanny Bouin a établi une version détaillée de cet algorithme [5], lors d'un stage de recherche que j'encadrais.

M. Delorme et J. Mazoyer ont démontré un résultat voisin concernant les AC de dimension 2 équipés d'un fil (cette variante d'AC est décrite dans la Section 4) : tout langage rationnel est reconnu en temps réel par un AC de dimension 2 avec voisinage de Moore qu'il soit équipé du fil d'Archimède ou du fil de Hilbert [14]. Et qui dit temps réel dit accélération optimale. Suivant cette piste et avec l'hypothèse additionnelle que chaque cellule connaisse initialement sa position relative à la cellule de sortie, j'ai prouvé que tout langage rationnel est reconnu en temps réel par un AC de dimension 2 avec voisinage de von Neumann et ceci quelque soit le fil choisi [58]. La construction use d'un algorithme d'amincissement classique de traitement d'images appliqué au fil.

6 Perspectives

Un objectif transverse à quelques unes des pistes de recherche présentées ci-dessous est de développer des outils algorithmiques pour les AC en dimension 2. De fait, le passage de la dimension 1 à la dimension 2 implique un saut de complexité substantiel dans la dynamique des AC. Ainsi, savoir si la fonction globale d'un AC est injective (resp. surjective), est une question décidable dans le cas de la dimension 1 mais indécidable dans le cas de la dimension 2 [26]. Sur le plan algorithmique, des exemples comme la construction de cercles discrets en temps réel [13] ou la fermeture transitive des graphes [18] montrent que l'organisation des calculs devient plus compliquée en dimension 2. Les interactions sont plus riches et les différentes notions développés dans le cadre de la dimension 1, comme la notion de signal [16, 9, 49, 15], le calcul avec des grilles [30] ou les concepts de particules et collisions [35, 34], ne sont pas suffisantes pour capturer toutes les possibilités de transmettre et combiner l'information. Bref on aimerait identifier les régularités propres à la géométrie des calculs sur des données bidimensionnelles.

6.1 Accélération linéaire en dimension 2

Par bien des aspects, la situation devient plus délicate lorsqu'on augmente à la fois la dimension du réseau et la dimension des données. Avec des interactions locales plus complexes, les calculs sont plus imbriqués. Et la question du voisinage apparaît alors comme essentielle. En particulier, on ne sait pas si les résultats d'accélération linéaire s'appliquent en dimension 2 quelque soit le voisinage. L'algorithme usuel d'accélération linéaire défini par Beyer [3], utilise une opération préalable de groupage suivie d'une transformation élémentaire.

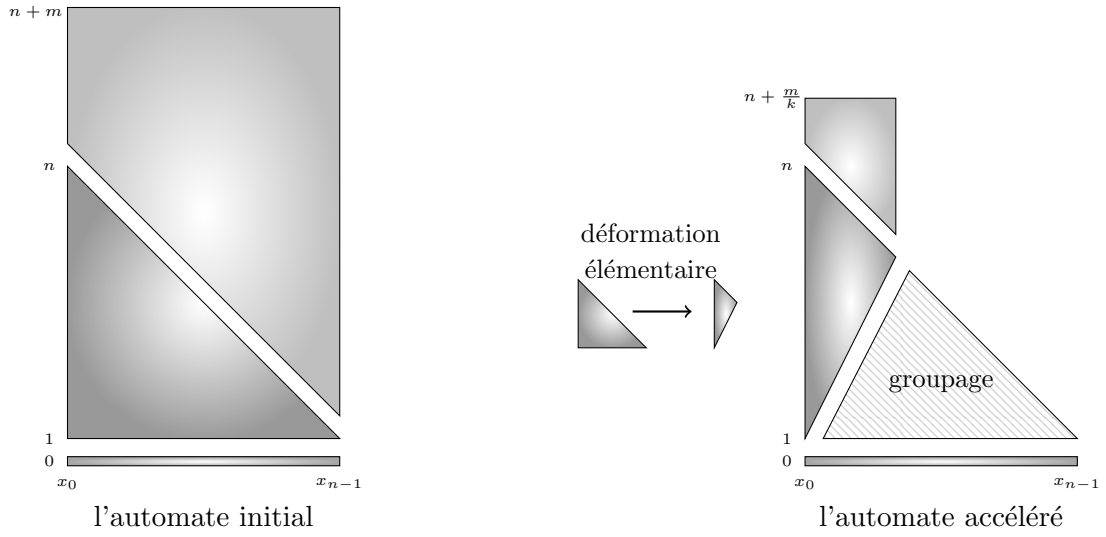


Figure 5: L'accélération de Beyer

Cette approche a un travers : l'opération de groupage nécessite que chaque cellule connaisse initialement la direction de la cellule de sortie et, dès lors, ne s'applique en dimension 2 qu'à certaines classes de voisinages. Or il existe, dans le cas de la dimension 1, un autre algorithme d'accélération qui évite cette opération initiale de groupage. Cet algorithme est la combinaison de deux déformations élémentaires et symétriques du diagramme espace-temps. Je souhaiterais étudier sa généralisation en dimension 2. Mais

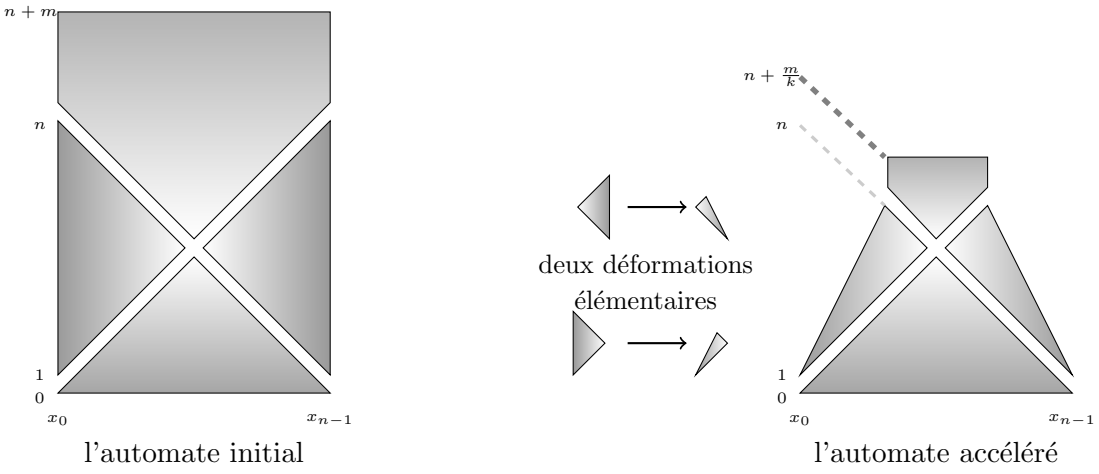


Figure 6: Une autre méthode d'accélération linéaire

quoique l'algorithme en dimension 1 est simple, le passage à la dimension 2 n'est pas immédiat. Une étape préliminaire serait d'expliciter les déformations valides applicables aux graphes de dépendances des AC 2D.

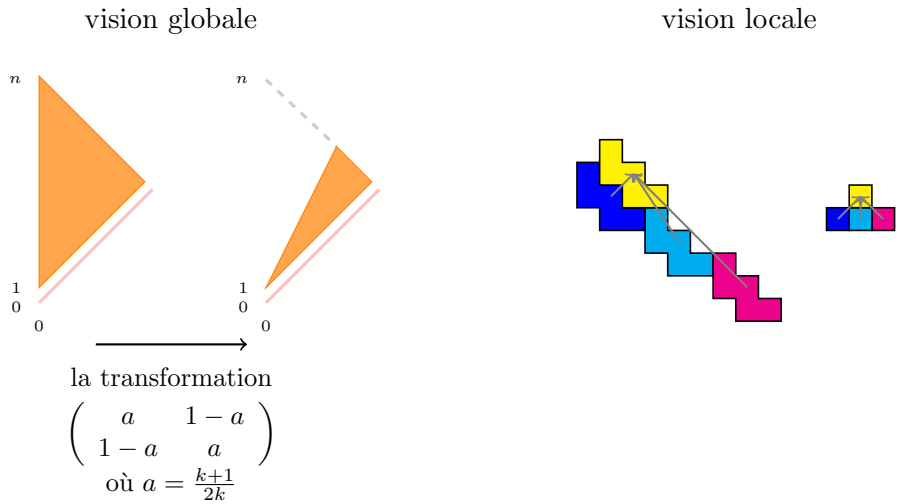


Figure 7: La déformation élémentaire gauche

6.2 Voisinage anarchique

Je souhaiterais aussi comprendre en quoi la connaissance locale que chaque cellule a de sa position et des positions relatives de ses voisines, influe sur les capacités de reconnaissance des AC. En dimension 1, un résultat dû à Szwerinski dit qu'avec le voisinage de premiers voisins $\{-1, 0, 1\}$, les cellules d'un AC peuvent confondre leur droite et leur gauche sans perte de temps et souligne ainsi que l'orientation n'a pas d'incidence [38]. Je présume que les cellules peuvent tout mélanger y compris leur centre. Autrement dit, n'importe quel AC de dimension 1 se simulerait sans perte de temps par un AC dont la fonction de transition est indépendante de l'ordre des états du voisinage. Chaque état de cet AC regrouperait un ensemble (fini!) d'états plausibles. Et l'état final correct serait calculé en distinguant explicitement les bords gauche et droit des mots d'entrée.

Il serait intéressant de déterminer si le résultat de Szwerinski s'étend ou non à la dimension supérieure. Est il possible de simuler des AC de dimension 2 par des AC dont chaque cellule reçoit les informations de ses voisines sans connaître leur provenance respective ? Mais il n'est pas certain que le nombre d'agencements plausibles reste fini lorsqu'on cherche à reconstruire la structure bidimensionnelle de l'entrée à partir des données reçues en vrac.

6.3 Langages d'images

Il existe d'autres travaux qui visent également à caractériser les classes de langages d'images. C'est en premier Blum et Hewitt qui, à la fin des années soixante, ont exploré la question de la reconnaissance de familles d'images par différents types d'automates se déplaçant dans le plan [4]. Plus récemment, avec comme objectif de généraliser à la dimension 2 la notion de langage rationnel, Restivo et Giammarresi ont introduit différents modèles formels qui caractérisent des classes de langages d'images [17]. La plupart de ces approches, basées sur des systèmes de tuiles, des expressions régulières, des formules logiques et divers automates, capturent dans leur version non-déterministes la même classe de langages d'images, les langages dits *reconnaissables*. Par contre, les variantes déterministes ne bénéficient pas des mêmes propriétés de robustesse et définissent des classes de langages distinctes.

Des relations entre ces modèles et les AC sont connues. Ainsi, E. Grandjean, F. Olive et G. Richard ont obtenu une caractérisation logique des AC non déterministes opérant en temps linéaire dont une variante définit également la classe des langages d'images reconnaissables [19]. Par ailleurs, j'ai établi des liens entre les AC et une variante déterministe d'automates, les DOTA: les DOTA se simulent en temps réel sur AC aussi bien avec le voisinage de Moore que celui de von Neumann, en revanche, la rotation à 180° du calcul d'un DOTA se simule en temps réel sur AC avec le voisinage de von Neumann mais pas avec celui de Moore [53]. Expliciter plus avant les liens et les différences entre les AC et les divers modèles qui généralisent la notion de langage rationnel au cadre bidimensionnel, aiderait à mieux comprendre les caractéristiques du calcul sur AC en dimension 2.

À l'instar de la dimension 1 où la reconnaissance de langages unaires est une bonne

entrée en matière pour explorer les possibilités algorithmiques des AC linéaires, le cas spécifique des langages d'images unaires où seule la taille des mots est significative, mérite d'être regardé. Quelques propriétés ont été abordées dans le mémoire de recherche de Maxime Leloup [29]. Et je souhaiterais poursuivre l'étude de ces propriétés et examiner la complexité de reconnaissance sur AC des classes de langages d'images unaires caractérisées par des systèmes de tuiles.

6.4 Accélérer les calculs séquentiels

Il s'agit ici d'un projet commun avec Alex Borello et Gaétan Richard [61]. L'objectif est de déterminer des simulations efficaces de modèles séquentiels par des AC. Bien entendu, il n'est pas question d'aborder le problème dans toute sa généralité mais de considérer juste une version restreinte d'automates finis à plusieurs têtes qualifiés d'"oblivious". La particularité de ces automates "oblivious" réside dans le déplacement des têtes qui ne dépend pas des données mais uniquement de la longueur de l'entrée. Quoique ce modèle soit élémentaire, Holzer a montré que sa capacité de calcul correspond à la classe de complexité parallèle NC^1 [21].

La simulation envisagée repose sur les observations suivantes. La trajectoire des têtes est identique sur tous les mots de même longueur, en particulier sur le mot ne contenant que des a (pour a une lettre de l'alphabet). Les seuls changements de rythme interviennent donc lorsqu'une des têtes atteint l'un des marqueurs gauche ou droit qui bordent le mot. Ainsi, hormis lorsqu'une tête est à proximité d'un bord, la trajectoire des têtes admet des comportements périodiques. Et l'idée est que l'AC peut accélérer le calcul d'un automate "oblivious" en simulant simultanément l'ensemble de ses comportements périodiques. En bref, l'AC effectue des pré-calculs qui simulent toutes les fragments périodiques possibles. Ensuite, le point délicat est de recoller les morceaux du calcul, i.e. les fragments périodiques et non périodiques.

Bibliographie

[1] A. J. Atrubin. A one-dimensional real-time iterative multiplier. *IEEE Transactions on Electronic Computers*, EC-14(1):394–399, 1965.

[2] E. R. Banks. Information processing and transmission in cellular automata. Technical Report AITR-233, MIT Artificial Intelligence Laboratory, January 1 1971.

[3] W. T. Beyer. Recognition of topological invariants by iterative arrays. Technical Report AITR-229, MIT Artificial Intelligence Laboratory, October 1 1969.

[4] M. Blum and C. Hewitt. Automata on two-dimensional tape. In *Proc. IEEE 8th Annual Symposium on Switching and Automata Theory*, IEEE Symposium on Switching and Automata Theory, pages 155–160, 1967.

[5] F. Bouin. Simulation en temps linéaire sur automates cellulaires des automates finis à 2 dimensions. Mémoire de DEA, 2003.

[6] J. Cervelle. *Complexité dynamique et algorithmique des automates cellulaires*. Habilitation à diriger des recherches, Université Paris Est, Marne-la-Vallée, Decembre 2007.

[7] J. H. Chang, O. H. Ibarra, and A. Vergis. On the power of one-way communication. *Journal of the ACM*, 35(3):697–726, July 1988.

[8] J. H. Chang, O. H. Ibarra, and M. A. Palis. Efficient simulations of simple models of parallel computation by time-bounded ATMs and space-bounded TMs. *Theoretical Computer Science*, 68(1):19–36, October 1989.

[9] S. N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machine. *IEEE Transactions on Computing*, 18:349–365, 1969.

[10] K. Čulik II, J. Gruska, and A. Salomaa. Systolic trellis automata. I. *International Journal Computer Mathematics*, 15(3-4):195–212, 1984.

[11] K. Čulik II. Variations of the firing squad problem and applications. *Information Processing Letters*, 30(3):153–157, February 1989.

[12] M. Delorme and J. Mazoyer. Reconnaisance de langages sur automates cellulaires. Research Report 94-46, LIP, ENS Lyon, France, Dec. 1994.

[13] M. Delorme, J. Mazoyer, and L. Tougne. Discrete parabolas and circles on 2D cellular automata. *Theoretical Computer Science*, 218(2):347–417, May 1999.

[14] M. Delorme and J. Mazoyer. Reconnaissance parallèle des langages rationnels sur automates cellulaires plans. *Theoretical Computer Science*, 281(1–2):251–289, May 2002.

[15] M. Delorme and J. Mazoyer. Algorithmic Tools on Cellular Automata. In *Handbook of Natural Computing: Theory, Experiments, and Applications*, edited by G. Rozenberg, T. Baeck and J. Kok (Springer Verlag (Heidelberg)), à paraître: décembre 2010.

[16] P. C. Fischer. Generation of primes by one-dimensional real-time iterative array. *Journal of the ACM*, 12:388–394, 1965.

[17] D. Giammarresi and A. Restivo. Two-dimensional languages. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, pages 215–267. Springer-Verlag, Berlin, 1997.

[18] L. J. Guibas, H. T. Kung, and C. D. Thompson. Direct VLSI implementation of combinatorial algorithms. In *Proceedings of the First Caltech Conference on VLSI*, pages 509–525, Pasadena, CA, January 1979. California Institute of Technology.

[19] E. Grandjean, F. Olive and G. Richard. Linear time complexity on cellular automata. *Rédaction en cours.*

[20] G. A. Hedlund. Endomorphisms and automorphisms of the shift dynamical systems. *Mathematical Systems Theory*, 3(4):320–375, 1969.

[21] M. Holzer. Multi-head finite automata: data-independent versus data-dependent computations. *Theoretical Computer Science*, 286(1):97–116, August 2002.

[22] J. Hromkovič and K. Inoue. A note on realtime one-way synchronized alternating one-counter automata. *Theoretical Computer Science*, 108(2):393–400, February 1993.

[23] O. H. Ibarra and T. Jiang. Relating the power of cellular arrays to their closure properties. *Theoretical Computer Science*, 57(2-3):225–238, May 1988.

[24] K. Inoue, A. Ito and I. Takanami. A note on real time one-way alternating multicounter machines. *Theoretical Computer Science* 88 (1991) 287-296.

[25] A. Ito, K. Inoue and I. Takanami. Deterministic two-dimensional on-line tessellation acceptors are equivalent to two-way two-dimensional alternating finite automata through 180°-degree rotation. *Theoretical Computer Science*, 66(3):273–287, 26 August 1989.

[26] J. Kari. Reversibility and surjectivity problems of cellular automata. *Journal of Computer and System Sciences*, 48(1):149–182, February 1994.

[27] A. Klein and M. Kutrib. Fast one-way cellular automata. *Theoretical Computer Science*, 295(1–3):233–250, February 2003.

[28] S. R. Kosaraju. Fast parallel processing array algorithms for some graph problems (preliminary version). In ACM, editor, *Conference record of the eleventh annual ACM Symposium on Theory of Computing: papers presented at the Symposium, Atlanta, Georgia, April 30–May 2, 1979*, pages 231–236, New York, NY, USA, 1979. ACM Press.

[29] M. Leloup. Reconnaissance de langages d'images unaires par automate cellulaire. Mémoire de Master Recherche, 2007.

[30] J. Mazoyer and J. B. Yunès. Computations on Cellular Automata. In *Handbook of Natural Computing: Theory, Experiments, and Applications*, edited by G. Rozenberg, T. Baeck and J. Kok (Springer Verlag (Heidelberg)), à paraître: décembre 2010.

[31] A. Okhotin. Automaton representation of linear conjunctive languages. In *International Conference on Developments in Language Theory (DLT), LNCS*, volume 6, 2002.

[32] V. Poupet. Cellular automata: Real-time equivalence between one-dimensional neighborhoods. In Volker Diekert and Bruno Durand, editors, *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, volume 3404 of *Lecture Notes in Computer Science*, pages 133–144. Springer, 2005.

[33] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, October 1980.

[34] N. Ollinger and G. Richard. Automata on the plane vs particles and collisions. *Theoretical Computer Science*, 410(27-29):2767 – 2773, 2009.

[35] G. Richard. *Systèmes de particules et collisions discrètes dans les automates cellulaires*. Thèse de doctorat, Université Aix-Marseille, December 2008.

[36] A. R. Smith III. Simple computation-universal cellular spaces. *Journal of the ACM*, 18(3):339–353, July 1971.

[37] A. R. Smith III. Real-time language recognition by one-dimensional cellular automata. *Journal of Computer and System Science*, 6:233–253, 1972.

[38] H. Szwerinski. Symmetrical one-dimensional cellular spaces. *Information and Control*, 67(1–3):163–172, October/November/December 1985.

[39] A. Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66–78, February 1966.

Références personnelles

[40] V. Terrier. Décidabilité de la théorie existentielle de N structuré par l'ordre naturel, la divisibilité, les prédicats puissances et les fonctions puissances. *C. R. Acad. Sci.*, Paris, Ser. I 311, No. 12, 749–752 (1990).

[41] V. Terrier. Décidabilité en arithmétiques faibles. Temps réel sur automates cellulaires. Thèse de doctorat, Université Claude Bernard, Lyon 1 (1991)

[42] V. Terrier. Signals in linear cellular automata. Proc. Workshop on Cellular Automata, Centre for Scientific Computing, Espoo Finland, 1991

[43] V. Terrier. Decidability of the existential theory of the set of natural numbers with order, divisibility, power functions, power predicates, and constants. *Proc. Am. Math. Soc.*, 114, No. 3, 809-816 (1992).

[44] V. Terrier. Real time recognition with cellular automata: a meaningful example. *Theoretical Informatics and Applications*, 27(2):97–120, 1993.

[45] V. Terrier. Language recognizable in real time by cellular automata. *Complex Systems*, 8:325–336, 1994.

[46] V. Terrier. On real time one-way cellular array. *Theoretical Computer Science*, 141(1–2):331–335, April 1995.

[47] V. Terrier. Language not recognizable in real time by one-way cellular automata. *Theoretical Computer Science*, 156(1–2):281–287, March 1996.

[48] V. Terrier. A counting equivalence classes method to prove negative results. *"Cellular Automata: a parallel model.", M. Delorme and J. Mazoyer Eds, Mathematics and Its Applications, December 1998, Kluwer.Cellular automata*, 199–210.

[49] J. Mazoyer and V. Terrier. Signals in one-dimensional cellular automata. *Theoretical Computer Science*, 217:53–80, 1999.

[50] V. Terrier. Two-dimensional cellular automata recognizer. *Theoretical Computer Science*, 218(2):325–346, May 1999.

[51] J.C. Dubacq and V. Terrier. Signals for cellular automata in dimension 2 or higher. In *LATIN 2002*, LNCS 2286:451–464, 2002.

[52] V. Terrier. Characterization of real time iterative array by alternating device. *Theoretical Computer Science*, 290(3):2075–2084, 2003.

[53] V. Terrier. Two-dimensional cellular automata and deterministic on-line tessalation automata. *Theoretical Computer Science*, 301(1–3):167–186, May 2003.

[54] V. Terrier. Two-dimensional cellular automata and their neighborhoods. *Theoretical Computer Science*, 312(2–3):203–222, January 2004.

[55] V. Terrier. Cellular automata recognizer with restricted communication. *Eugen Czeizler and Jarkko Kari (Eds.), Proceedings of DMCS'04 Workshop on Discrete Models for Complex Systems, Turku, Finland 2004*, Pages 84-88.

[56] V. Terrier. Closure properties of cellular automata. *Theoretical Computer Science*, 352(1-3):97–107, 2006.

[57] V. Terrier. Low complexity classes of multidimensional cellular automata. *Theoretical Computer Science*, 369(1-3):142–156, 2006.

[58] V. Terrier. Two-dimensional cellular automata recognizer equipped with a path. In *Premières Journées Automates Cellulaires, JAC'08*, Avril 2008, Pages 174-181.

[59] V. Terrier. Simulation of one-way cellular automata by boolean circuits. *Theoretical Computer Science*, 411(1): 266–276, (2010).

[60] V. Terrier. Language recognition by Cellular Automata. In *Handbook of Natural Computing: Theory, Experiments, and Applications*, edited by G. Rozenberg, T. Baeck and J. Kok (Springer Verlag (Heidelberg)), 40 pages, à paraître: décembre 2010.

[61] A. Borello, G. Richard and V. Terrier A speed-up of oblivious multi-head finite automata by cellular automata. *STACS 2011, 28nd Annual Symposium on Theoretical Aspects of Computer Science, Dortmund*, March 2011.

Résumé

Les automates cellulaires ont été introduits il y a une soixantaine d'années par von Neumann et Ulam qui cherchaient à définir les caractéristiques d'un système formel apte au calcul universel et à l'auto-reproduction. Leur utilité a été rapidement reconnue dans des domaines variés comme la physique et la biologie, pour modéliser des phénomènes complexes.

En informatique, ils offrent un cadre privilégié pour l'étude du parallélisme massif. Leur description est simple et bien formalisée. Ils ont a la même puissance de calcul que les machines de Turing et de plus la richesse algorithmique propre aux machines parallèles tout en restant un modèle physiquement réaliste.

Dans le cadre unificateur de la reconnaissance de langages, je m'intéresse aux questions de complexité sur les automates cellulaires, avec une attention particulière aux petites classes de complexité : calcul en temps réel (i.e. temps minimal) et en temps linéaire; en effet, c'est pour ces classes que l'apport du parallélisme est remarquable par rapport au mode séquentiel.

Avec pour objectif de préciser les capacités de ce modèle et de mieux comprendre ce qu'est un calcul parallèle, trois tendances majeures se dégagent de mes travaux : l'étude des limites de ce modèle, la comparaison avec d'autres modèles de calcul et la question de l'influence de certains paramètres comme la dimension ou le voisinage sur ses capacités de reconnaissance.

Quelques articles représentatifs

1. **Language recognition by Cellular Automata**
2. **Two-dimensional cellular automata recognizer**
3. **Low complexity classes of multidimensional cellular automata**
4. **Simulation of one-way cellular automata by boolean circuits**

Language recognition by Cellular Automata

Véronique Terrier

GREYC, UMR CNRS 6072, Campus II, Université de Caen, 14032 Caen, France

Summary. Cellular automata are a simple and well-formalized model of massively parallel computation which is known to be capable of universal computation. Due to their parallel behavior, cellular automata have rich abilities of information processing but in return it is not so easy to define their power limitations. A convenient approach to characterize the computation capacity of cellular automata, is to investigate their complexity classes. The aim of this chapter is to present the results and questions about the cellular automata complexity classes and their relationships with other models of computations.

1 Preliminary

Cellular automata provide an ideal framework for studying massively parallel computation. Their description is very simple and homogeneous, while, as emphasized by various examples, they allow us to distribute and synchronize the information in a very efficient way. Their ability to do fast computation has soon stimulated complexity issues. And early works have drawn much attention to cellular automata as language recognizer.

Obviously, cellular automata (CA in short) are of the same computational power as Turing machines. Hence the major motivation for CA complexity study is to get finer knowledge on the way parallelism acts and to render explicit the gain that may be achieved with CA. In particular, a lot of interest is devoted to their low time complexity classes as they may provide significant complexity benefits. But, as for the other models of computation, it is not a simple task to evaluate finely their power and their limitations.

In this chapter, we will review the essential developments on CA in the field of language recognition. The purpose is to give insight into the performance of different types of CA, with a major concern with the low time complexity classes. Of course, it will not be possible to report every outcome in detail and some choices have been made. First, we will only deal with space bounded computation. Precisely, due to our interest in fast computation, we will just

consider space bound defined as the space consumed by low time computation. And, despite their interest for general complexity issues, we also choose not to consider either non deterministic or alternating variants of CA. As a matter of fact, they are beyond the scope of realistic devices, because even their minimal time classes contain *NP*-complete problems. For the same reasons, we shall ignore CA with tree-like array structures.

The rest of this chapter is organized as follow. Section 2 introduces the different variants of CA recognizers and their complexity classes. Section 3 reviews the known inclusions and equalities among these complexity classes. Section 4 recalls the limitations established on the recognition power of CA. Section 5 relates to the comparison with other models of computation. As a conclusion, section 6 discusses some of the old open questions which remain up to now unsolved.

2 Definitions and examples

Basically, a CA is a regular array of cells. These cells range over a finite set of states and evolve synchronously at discrete time step. At each step, the new state of each cell is produced by a local transition rule according to the currents states in its neighborhood. So a CA is completely specified by a tuple $(d, S, \mathcal{N}, \delta)$ where d is the dimension of the array of cells indexed by $\mathbb{Z}^d$, S is the finite set of states, the neighborhood $\mathcal{N}$ is a finite ordered subset of $\mathbb{Z}^d$ and δ is the transition function from $S^{|\mathcal{N}|}$ into S.

A cell is denoted by $\mathbf{c}$, $\mathbf{c} \in \mathbb{Z}^k$; a site $(\mathbf{c}, t)$ denotes the cell $\mathbf{c}$ at time t and $\langle \mathbf{c}, t\rangle$ denotes its state. And at each step $t \geq 0$, the state is updated in this way: $\langle \mathbf{c}, t+1\rangle = \delta(\langle \mathbf{c}+\mathbf{v_1}, t\rangle, \cdots, \langle \mathbf{c}+\mathbf{v_r}, t\rangle : \mathcal{N} = \{\mathbf{v_1}, \cdots, \mathbf{v_r}\})$. Depending on the neighborhood, the flow of information may go in all directions of the cellular array as with the von Neumann neighborhood $\{\mathbf{v} \in \mathbb{Z}^k : \Sigma|v_i| \leq 1\}$ or with the Moore one $\{\mathbf{v} \in \mathbb{Z}^k : \max|v_i| \leq 1\}$ or it may be restricted to one-way in some directions as with the one-way von Neumann neighborhood $\{-\mathbf{v} : \mathbf{v} \in \mathbb{N}^k \text{ and } \Sigma v_i \leq 1\}$ or with the one-way Moore one $\{-\mathbf{v} : \mathbf{v} \in \mathbb{N}^k \text{ and } \max v_i \leq 1\}$. In the sequel, one-way communication will refer to such restrictions whereas two-way communication will refer to the ability to transmit information everywhere in the cellular array.

In order that a CA acts as a language recognizer, we have to specify how the input is given to the cellular array and how the result of the computation is obtained.

The input mode. For the sake of simplicity, we assume that Σ the finite alphabet of input letters is a subset of the states set S of the CA. We also consider CA with a quiescent state λ such that $\delta(\lambda, \cdots, \lambda) = \lambda$. We distinguish two modes to give the input to the array: the parallel one and the sequential one. In the *parallel mode*, the whole input is supplied at initial time to the array.

It implies an implicit synchronization at the beginning of the computation. All input symbols are fed into a distinct cell and are arranged in such a way as to keep the input structure. In the *sequential mode*, a specific cell is chosen to receive the input. All cells are initially quiescent and the input symbols are fed serially to the specific cell. When the whole input has been read by this input cell, an end-marker symbol \$ is infinitely fed to it. Because the input cell takes into account not only its neighborhood but also its received input symbol, it behaves in a particular way.

The output mode. The advantage of recognition problems is that the output is of yes/no type. Hence, on a CA, a specific cell is enough to indicate acceptance of rejection as regards the input. The choice of this output cell is arbitrary in the case of two-way communication. But, in the case of one-way communication, it is subject to constraint: the output cell must be able to get all information of the input.

Language recognition. For the purpose of recognition, two subsets of the states set S are specified: the set S_{acc} of accepting states and the set S_{rej} of rejecting states. A language L is said to be *recognized* by a CA, if on input w the output cell enters at some time t_e an accepting state if $w \in L$ or a rejecting state if $w \notin L$; and for all time $t < t_e$, the output cell is neither in an accepting nor a rejecting state. A time complexity T refers to a function relating the input size to an amount of time steps. A CA recognizer works in time T, if it accepts or rejects each word w of size s within $T(s)$ steps.

Usually, we simply focus on acceptance. In this case, only the set S_{acc} of accepting states is specified. A language L is said to be *accepted* by a CA, if on input w the output cell enters an accepting state if and only if $w \in L$. A CA acceptor works in time T, if it accepts each word $w \in L$ of size s within $T(s)$ steps.

In fact, the notion of acceptance and recognition turns out to be equivalent for currently time complexities. Indeed, a CA, on an input of size s, is able to distinguish the output cell at time $T(s)$ when T is any standard time complexity. Hence, the CA can reject at time $T(s)$ all non accepted words of size s. So, except just below, we will not differentiate between CA recognizer and CA acceptor.

Let us give an example of CA recognizer.

Example 1 *The majority language which consists of the words over the alphabet* $\{a, b\}$ *in which there are strictly more a's than b's, is recognized by a one-dimensional CA with parallel input mode and one-way neighborhood* $\{-1, 0\}$.

The CA is defined in this way.
$\Sigma = \{a, b\}$ is the input alphabet,
$S = \{a, b, A, B, 1, 2, n, x\}$ is the states set,
$S_{acc} = \{A\}$ is the set of accepting states,
$S_{rej} = \{B, x\}$ is the set of rejecting states,
the cell -1 is assumed to remain in a persistent state $\sharp$,
$\delta : S \cup \{\sharp\} \times S \to S$ is the transition function displayed on the right.

δ	a	b	A	B	1	2	n	x
a	a	b	A		a	n	A	
b	a	b		B	n	b	B	
A	a		A		a	n	A	
B		b		B	n	b	B	
1	1	2			1	2		
2	1	2			1	2		
n					1	2	n	
x	A	B	x	x	A	B	x	x
$\sharp$	1	2	x	x	A	B		x

Since the neighborhood is one-way, the output cell is chosen to be the rightmost one. The computations on inputs $w_1 = aaaaabbaaabbabbbbba$ and $w_2 = babbaaabbbba$ are depicted in Fig. 1. As the input mode is parallel, the words are supplied at initial time. On each cell, above the diagonal, the length of the sequence of a's and A's or, b's and B's, records the difference between the numbers of a's and b's of the input. The input word w_1 is accepted since the output cell $n-1$ enters the accepting state A, whereas the input word w_2 is rejected since it enters the rejecting state B. The CA recognizer works within time complexity $T(n) = 2n$. As a matter of fact, the cell 0 knows at time 1 that it is the leftmost one; so by mean of a signal which moves one cell to the right every two steps, it is simple to mark the output cell $n-1$ at time $2n$. Thus the status of the words not accepted may be decided at this time even though the set of rejecting states is not specified. More generally, notice that, on an input of length n, the output cell is able to know the whole input at time $n-1$ whereas it is able to know that the input is completed at time n.

Time complexities. Among the time complexities, two functions are of major interest: the real time and linear time. The *real time* complexity, denoted by rt, means that each word is accepted or rejected "as soon as possible". Here, we encounter the only slight difference between acceptance and recognition. For acceptance, the real time complexity corresponds to the minimal time for the output cell to read the whole input, whereas, for recognition, one additional unit of time may be required in order that the output cell knows that the input is completed. In the sequel, we will only deal with real time acceptance. Practically, the real time complexity is specified by the way the input is supplied, the choices of the output site and the neighborhood. The *linear time* complexity, denoted by lt, is just defined as the real time function multiplied by any constant strictly greater than 1.

Space bounded computation. In this chapter, we will restrict ourselves to space bounded computation. During all the computation, only a fixed number of cells, depending on the size of the input, are active. All other cells remain in a persistent state $\sharp$ from the start to the end. We may imagine many space bounds. However, in practice, the bounded space is *uniquely* defined as the

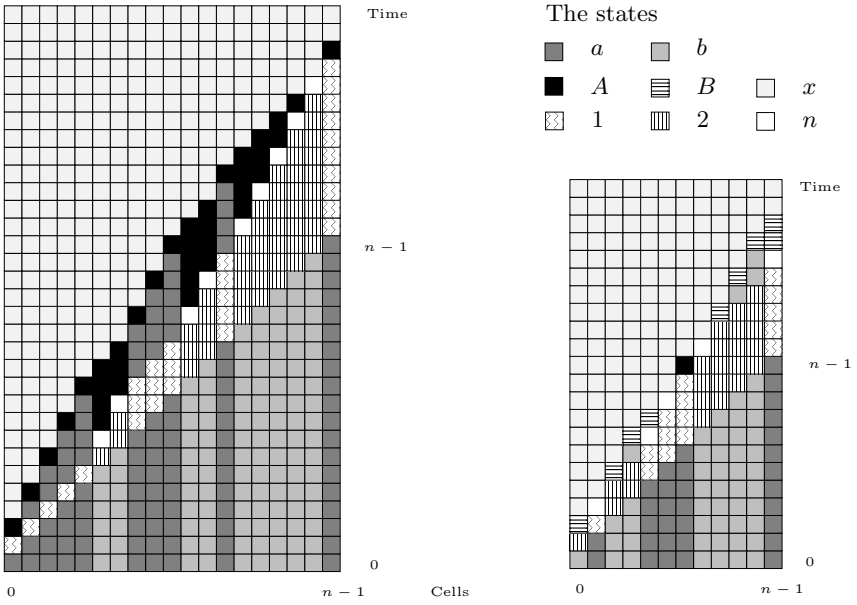


Fig. 1. Recognition of the majority language.

space required to perform real time computation. It means that the active part of the array is identical whatever the effective time complexity may be. Then, in the following, all classes defined in terms of time complexity classes will be actually both time and space bounded. As a matter of fact, space bounded and unbounded computation become the same for real time and linear time complexities. On the other hand, because the space is bounded, the evolution becomes periodical after a number of steps that is exponential in the size of the space. That establishes an upper bound on the relevant time complexities.

The dependency graph. In order to reflect the neighborhood constraints on the sites involved in the computation, we will consider the graph induced by the dependencies between them. Precisely, a dependency graph is defined according to a given type of CA, a fixed time complexity and the input size. It is a directed graph. Its set of vertices consists of the sites which both are influenced by the input and can have an effect on the output site, in other words all relevant sites regarding the computation. Its edges are the couples of sites $((\mathbf{c}+\mathbf{v},t),(\mathbf{c},t+1))$, for all $\mathbf{v}$ belonging to the neighborhood $\mathcal{N}$.

Now the different sorts of CA recognizers will be described in the following subsections. Their features are stated by the array dimension, the input dimension and, in the parallel mode, the way to place the input into the array.

2.1 One-dimensional CA language recognizer.

A one-dimensional CA recognizer is structured as a linear array and operates on words. The space is assumed linearly bounded: the number of active cells equals the length of the input. In practice, the active cells are numbered $0, \cdots, n-1$, for an input of length n. Because the other cells always remain in the persistent state $\sharp$, one-dimensional CA with two-way communication have the same computational power as Turing machines working in $O(n)$ space.
Different variants of one-dimensional CA are currently defined. On the one hand, it depends on the neighborhood which allows either two-way or one-way communication. Actually, from [41], it is known that there are somewhat only two kinds of neighborhoods, the two-way one $\{-1, 0, 1\}$ and the one-way one $\{-1, 0\}$. On the other hand, it depends on the input mode being either parallel or sequential. Hence four variants of one-dimensional CA are characterized according to the neighborhood and input mode choices. They are named PCA, SCA, POCA and SOCA. The first letter "P" or "S" stands for parallel or sequential input mode and the "O" occurrence makes distinctions between one-way or two-way communication.

Automata	Neighborhood	Input mode	Output cell
PCA	$\{-1, 0, 1\}$	parallel	0
SCA	$\{-1, 0, 1\}$	sequential	0
POCA	$\{-1, 0\}$	parallel	$n-1$
SOCA	$\{-1, 0\}$	sequential	$n-1$

Table 1. The four variants of CA in dimension one.

Different denominations have been used in other papers: SCA and SOCA are more often called iterative array (IA) and one-way iterative array (OIA); POCA are also called one-way cellular automata (OCA) or mentioned as trellis automata.

Let us describe the way to carry out the input in the two modes. In the parallel input mode, at initial time 0, the i-th symbol of the input $w = x_0 \cdots x_{n-1}$ of size n, is fed to the cell i: $\langle i, 0\rangle = x_i$. In the sequential input mode, all active cells are initially quiescent (i.e., set to λ). The cell indexed by 0 is chosen to read the input: it gets the i-th symbol x_i of the input $w = x_0 \cdots x_{n-1}$ at time i; then it gets an end-marker \$ at all time $t \geq n$. Note that the input cell 0 evolves according to a particular transition function $\delta_{\text{init}} : (\Sigma \cup \{\$\}) \times S^{|\mathcal{N}|} \to S$, so $\langle 0, t\rangle = \delta_{\text{init}}(x_t, \langle \mathbf{v}_1, t-1\rangle, \cdots, \langle \mathbf{v}_r, t-1\rangle)$ where $\mathcal{N} = \{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$ refers to the neighborhood and $x_t = \$$ when $t \geq n$.

For the output cell that yields the result, we choose the initial cell 0 in case of PCA and SCA which use two-way communication. The unique possibility for POCA and SOCA is the rightmost cell indexed $n-1$ since their neighborhood $\{-1, 0\}$ is one-way.

The Fig. 2 below shows the customary representation of the four models.

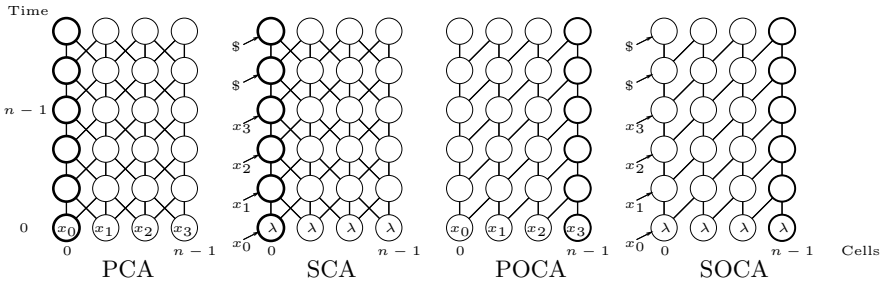


Fig. 2. The space-time diagram of the four one-dimensional CA variants.

Some notation is useful in dealing with the various complexity classes of one-dimensional CA recognizers. For a time complexity function T from $\mathbb{N}$ into $\mathbb{N}$, PCA(T) (resp. SCA(T), POCA(T) and SOCA(T)) will denote the class of languages recognizable in time T by PCA (resp. SCA, POCA and SOCA). With some liberty, PCA will refer to the class of languages recognized in unbounded time by PCA; and in the same way, SCA, POCA and SOCA will indicate the device type as well their corresponding unbounded time complexity classes.

Particular attention will be devoted to the low time complexity classes, namely the real time and the linear time. The real time complexity $rt(n)$ is defined as the earliest time for the output cell to read the whole input of length n. More precisely, it corresponds to $n-1$ in case of PCA, SCA and POCA and to $2n-2$ in case of SOCA. And $lt(n) = \tau\, rt(n)$, where τ is any constant strictly greater than 1, gives rise to linear time complexity. In the following, the classes of language recognized in real time by PCA, SCA, POCA and SOCA will be denoted RPCA, RSCA, RPOCA and RSOCA. For linear time complexity, the corresponding classes will be designated by LPCA, LSCA, LPOCA and LSOCA.

Let us give some examples to illustrate the computation ability of the real time complexity classes.

Example 2 (Cole [10]) *The palindrome language* $\{w \in \Sigma^* \colon w = w^R\}$ *is a real time* SCA *language (*w^R *denotes* w *read backward).*

We describe the algorithm in a geometrical way. Its discretization to obtain the corresponding RSCA is straightforward. First of all, the input word received sequentially on the input cell 0 is sent at maximal speed to the right. Precisely the symbol x_i which is fed on cell $c = 0$ at time $t = i$ follows the line A_i of equation $t = c + i$ with $c \geq 0$. Simultaneously, a signal F of speed $1/3$ starts from the input cell 0 at initial time and draws the line $t = 3c$. So the signals A_i and F intersect on the point $(i/2, 3i/2)$. From this intersection, the symbol x_i carried by A_i goes further along the vertical line $B_i \colon c = i/2$ with $t \geq 3i/2$. In this way, the symbols x_j and x_i meet on the point $(i/2, j + i/2)$

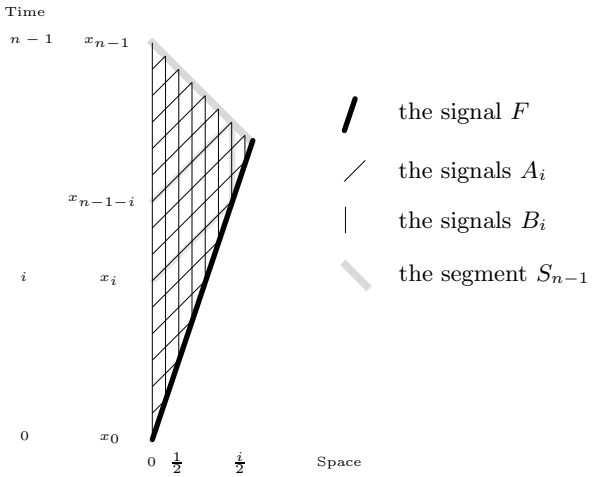


Fig. 3. Real time recognition of the palindrome language by a SCA.

where the signals A_j and B_i intersect. In particular, for any integer k, on the segment $S_k\colon c+t=k$ with $0 \leq c \leq k/4$, which runs at maximal speed to the left from the signal F to the output cell 0, we may compare all pairs of symbols $\{x_i, x_{k-i}\}$ with i such that $0 \leq i \leq k/2$. Now, on an input $x_0 \cdots x_{n-1}$ of length n, the sequence of comparisons between x_i and x_{n-1-i} with $0 \leq i \leq (n-1)/2$ determines whether the input is a palindrome or not. This sequence of comparisons is exactly the one which occurs on the segment $S_{n-1}\colon c+t=n-1$. Finally, observe that the segment S_{n-1} reaches the cell $c=0$ at time $t=n-1$. In other words the result is obtained in real time.

Example 3 (Cole [10]) *The square language* $\{ww\colon w \in \Sigma^*\}$ *is a real time* SCA *language.*

On the one hand, each input symbol x_i is carried along the line A_i of equation $t = c + i$ with $c \geq 0$. On the other hand, first of all, each input symbol x_i is sent at maximal speed to the left, following the signal B_i of equation $t = -c + i$ with $c \geq 0$. Simultaneously, initialized by the input cell c at time 0, a signal G of equation $t = -3c - 1$ starts from the point $(-1/2, 1/2)$ and moves with speed $1/3$ to the left. So the signals B_i and G intersect on the point $(-(i+1)/2, (3i+1)/2)$. From this intersection, the symbol x_i carried by B_i goes further along the signal $C_i\colon t = c + 1 + 2i$ with $t \geq (3i+1)/2$. Then the signal C_i intersects the initial cell $c = 0$ on the point $(0, 1+2i)$. From this intersection, the symbol x_i follows the signal $D_i\colon t = 3c + 1 + 2i$. In this way, for any i, j with $i < j$, the symbols x_i and x_j meet at the point $(j-1)/2-i, (3j-1)/2-i)$ where the signals D_i and A_j intersect. Now, on an input $x_0 \cdots x_{2n-1}$ of even length $2n$, the sequence of comparisons between x_i and x_{n+i} with $0 \leq i < n$, determines whether the input is a square word or not. These comparisons are performed on the points $((n-1-i)/2, (3n-1+i)/2)$.

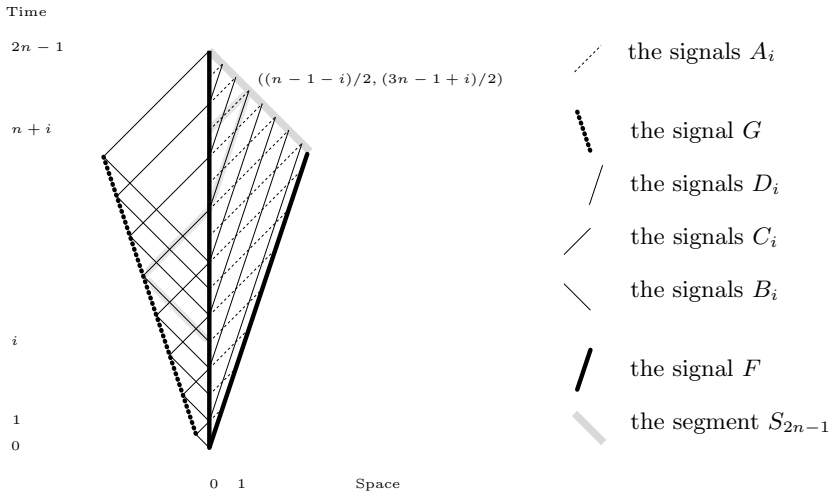


Fig. 4. Real time recognition of the square language by a SCA.

Hence they occur on the segment S_{2n-1}: $t+c = 2n-1$ with $0 \leq c \leq (2n-1)/4$ which starts from the signal F: $t = 3c$, moves at maximal speed to the left and reaches the cell 0 at time $2n-1$. Therefore the result is obtained in minimal time. With regard to space, note that negative cells are not necessarily active, since the negative side can be folded on the positive side, i.e., all activity on the negative cells can be as well performed on the positive side.

Example 4 (Smith [45]) *The Dyck language is a real time* POCA *language.*

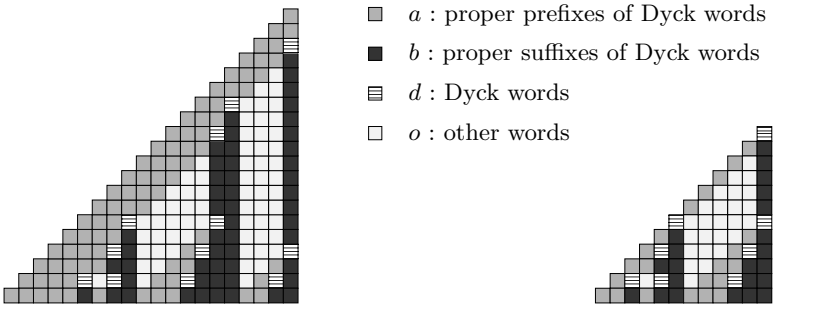


Fig. 5. Recognition of the Dyck language by a RPOCA.

A RPOCA which recognizes the Dyck language over the alphabet $\{a, b\}$ can be defined in this way.

$\Sigma = \{a, b\}$ is the input alphabet,
$S = \{a, b, d, o\}$ is the set of states,
$S_{acc} = \{d\}$ is the set of accepting states
and $\delta : S^2 \to S$ is the transition function displayed on the right.

δ	a	b	d	o
a	a	d	a	a
b	o	b		o
d		b		a
o	o	b	b	o

A significant feature of POCA is that the real time computation on an input w contains the real time computations of all its factors. An illustration is depicted in Fig. 5.

Example 5 (Culik [11]) *The language* $L = \{a^i\, b^{i+j}\, a^j : i, j \in \mathbb{N}\}$ *is a real time* POCA *language.*

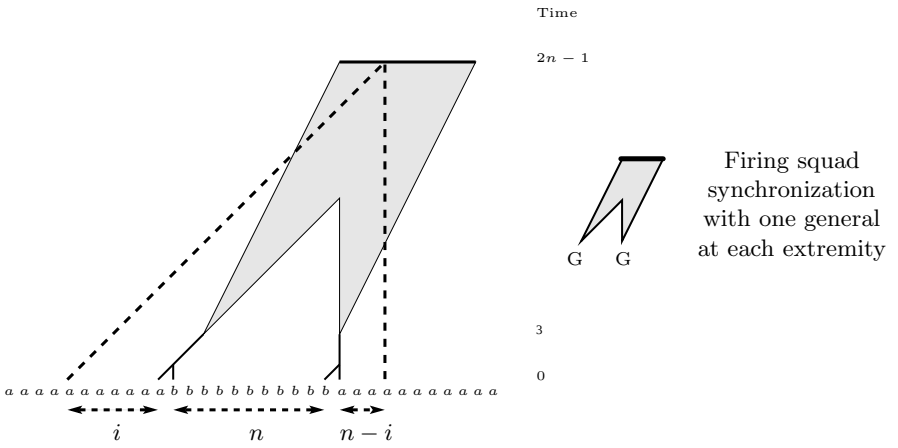


Fig. 6. Real time recognition of the Culik language by POCA.

First, the RPOCA will reject all the words outside the regular language $\{a^i\, b^j\, a^k : i, j, k \in \mathbb{N}\}$. Second, notice that the factors of a word of shape $a^*b^na^*$ which belong to L are on the one hand the words a^ib^i and b^ia^i which will be simply accepted by the RPOCA and on the other hand the words $a^ib^na^{n-i}$. Observe that to accept, in real time, all the factors $a^ib^na^{n-i}$ with $i = 1, \cdots, n-1$ of an input of shape $a^*b^na^*$, means that $n-1$ consecutive cells simultaneously enter at time $2n-1$ and for the first time in an accepting state. In other words, a synchronization process is required to recognize this language. As sketched in Fig. 6, the synchronization can be set up using a firing squad synchronization process with two generals located respectively according to the boundary between the a's and b's and the boundary between the b's and a's.

2.2 Multi-dimensional CA language recognizer.

Natural extensions to higher dimensional arrays have been early investigated in [10, 6, 30]. In this framework, the space increases with the dimension, while the minimal time complexity remains linearly bounded by the length of the input word. Precisely, on an input of length n, the space of a d-dimensional CA recognizer is defined as a d-dimensional array of size n in each dimension. Hence, a d-CA with two-way neighborhoods, has the same computation ability as Turing machines working in $O(n^d)$ space.

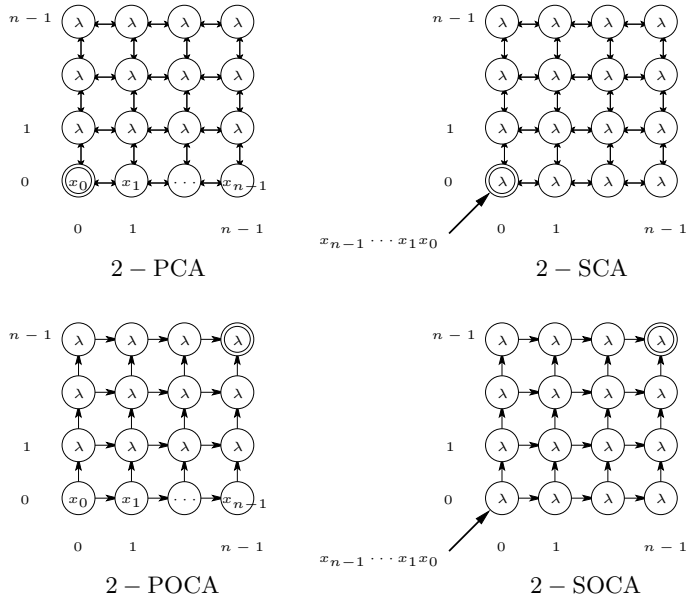


Fig. 7. The space array of the four 2-CA variants.

The diversity of neighborhoods also rises with the dimension. In the case of two-way neighborhoods and sequential input mode, we know from [10] that the computation ability is preserved even though the neighborhood is restricted to the von Neumann one: $\{\mathbf{v} \in \mathbb{Z}^d : \Sigma\, |v_i| \leq 1\}$. Whether the same is true or not for the other variants, is unclear. However, the impact of the neighborhood choice appears less crucial for language recognizers where almost all cells are initially quiescent than for picture language recognizers which will be defined below. Hence, as neighborhood issues are currently studied in this last context, we shall only regard the von Neumann neighborhood and its one-way counterpart $\{-\mathbf{v} : \mathbf{v} \in \mathbb{N}^d \text{ and } \Sigma\, v_i \leq 1\}$ in the case of a multi-dimensional CA language recognizer.

For an input of length n, the space area consists of the cells $\{\mathbf{c} : 0 \leq c_1, \cdots, c_d < n\}$. The two-dimensional case is depicted in Fig. 7. The cells

outside this area remain in a persistent state $\sharp$ during all the computation and each cell inside the area is assumed to remain quiescent until the step when it may be affected by the input. In the parallel mode, the input is supplied on the first dimension of the array: the i-th input symbol of the input $w = x_0 \cdots x_{n-1}$ is fed to the cell $(i, 0, \cdots, 0)$ at time 0. In the sequential mode, the specific cell which gets the input serially from time 0, is chosen to be the cell indexed by $\mathbf{0} = (0, \cdots, 0)$. Of course, this input cell evolves according to a particular transition function δ_{init} which takes into account its neighborhood and the received input symbol. The choice of the output cell depends on the neighborhood. One chooses the cell $\mathbf{0} = (0, \cdots, 0)$ with the von Neumann neighborhood and the opposite corner $\mathbf{n-1} = (n-1, \cdots, n-1)$ with the one-way von Neumann neighborhood.

Analogously to one-dimensional CA recognizer, $d-$PCA, $d-$SCA, $d-$POCA and $d-$SOCA denote the four d-dimensional variants according to whether the input mode is parallel or sequential and whether the neighborhood is the two-way von Neumann one or its one-way counterpart. They are denominated variously in different papers: $\mathrm{k}-$SCA is named k-dimensional iterative array in [10]; $2-$SOCA is named one-way two-dimensional iterative array in [6] and OIA in [30]; $\mathrm{k}-$POCA is named k-dimensional one-way mesh connected array in [7].

The real time function $rt(n)$ which is defined as the minimal time for the output cell to read the whole input of size n, corresponds to $n-1$ in the case of $d-$PCA and $d-$SCA, $d(n-1)$ in the case of $d-$POCA and $(d+1)(n-1)$ in the case of $d-$SOCA.

2.3 Two-dimensional CA language recognizer equipped with a thread.

With the parallel mode in order to save time, we may supply the input word in a more compact way than the linear one. The difficulty is that there are many ways to set up the one-dimensional input words into multi-dimensional arrays. All ways are arbitrary and lead to distinct devices with their own complexity classes. A proper approach has been proposed by Delorme and

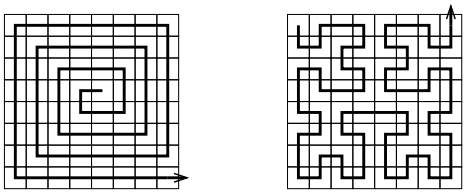


Fig. 8. Archimedean and Hilbert threads.

Mazoyer in [16]. To set up the inputs in a uniform way, they equip the CA

with a thread along which the inputs are written. Thus the thread is a given parameter of the CA which is independent of the inputs and their length. It is defined as an infinite sequence of adjacent positions in the two-dimensional array without repetition. Fig. 8 depicts the Archimedean and Hilbert threads. In practice, the device array is provided with an additional layer which codes the thread. In this layer, each cell records how the thread enters and exits this cell. At initial time, the input symbols are placed consecutively along the thread. Then the evolution of each cell depends on both the states and the thread components of its neighborhood. The output cell is chosen to be the first cell of the thread. Now, the real time complexity depends on both the neighborhood and the thread (precisely, its significant part according to the input length). We shall give no further details on these device types but shall refer to [16, 17] for complete definitions and examples.

2.4 Two-dimensional CA picture recognizer.

In higher dimension, another point of view is to process multidimensional data instead of simple strings. Motivated by image processing issues, a lot of interest has been devoted to picture language recognized by two-dimensional CA. Another stimulation comes from the developments of picture language theory [21]. Hence, in the following, we limit our attention to this context, although investigations of arbitrary dimensional CA with arbitrary dimensional inputs would be fairly instructive.

Let us recall some definitions related to picture languages. A picture p over an alphabet Σ is defined as a rectangular $m \times n$ array of symbols of Σ. The couple (m, n) refers to the size of the picture and $p(\mathbf{c})$ denotes the symbol at position $\mathbf{c}$. The set Σ^{**} denotes the set of all pictures over Σ. A picture language over Σ is any subset of Σ^{**}.

A two-dimensional CA picture recognizer (in short a PictCA) designates a CA recognizer which operates on pictures. On PictCA, the input mode is parallel: at initial time 0 the symbol $p(\mathbf{c})$ of the input picture p is fed to the cell $\mathbf{c}$. For an input of size (m, n), the bounded space consists of the $m \times n$ cells $\mathbf{c} = (x, y)$ with $0 \leq x < m$ and $0 \leq y < n$. Outside, the cells remain in a persistent state $\sharp$ during all the computation.

A priori, the output cell is the cell indexed by $(0, 0)$. The time complexities T are functions defined from $\mathbb{N}^2$ to $\mathbb{N}$. And a PictCA is said to accept a picture language L in time T if it accepts the pictures $p \in L$ of size (m, n) in at most $T(m, n)$ steps. As usual, real time means "as soon as possible" and is conditional on the neighborhood. Precisely, the real time function $rt_{\mathcal{N}}(m, n)$ is, for a PictCA with neighborhood $\mathcal{N}$, the minimal time needed by the output cell $(0, 0)$ to receive any particular part of a picture input of size(m, n). That means $rt_{\mathcal{N}}(m, n) = m + n - 2$ when $\mathcal{N}$ is the von Neumann neighborhood and $rt_{\mathcal{N}}(m, n) = \max(m, n) - 1$ when $\mathcal{N}$ is the Moore neighborhood. The linear time complexities for PictCA with the neighborhood $\mathcal{N}$ are functions $lt_{\mathcal{N}}$ where $lt_{\mathcal{N}}(m, n) = \tau\, rt_{\mathcal{N}}(m, n)$ and τ is any constant strictly greater than

1. In the sequel, the class of all pictures languages recognized by a PictCA with the neighborhood $\mathcal{N}$ in real time (or linear time) will just be named as the real time (or linear time) PictCA with the neighborhood $\mathcal{N}$.

Various algorithms for pictures have been proposed in the general context of mesh-connected arrays of processors. But, their processing elements are not necessarily finite-state contrary to CA. And specific examples which illustrate the possibilities of processing the data on PictCA, are scarce. Anyway, Beyer [1] and Levialdi [36] have independently exhibited two real time PictCA with the Moore neighborhood which recognize the set of connected pictures. The majority language which consists of the pictures over the alphabet $\{0,1\}$ in which there are more 1's than 0's has also been examined. Savage [44] has shown that it is recognized in linear time by PictCA with one-way neighborhoods.

3 Positive results and simulation

In this section, we shall examine the main known equalities and inclusions among CA complexity classes. These positive results are essentially based on the geometrical characteristics inherited from the regularity of the network structure. The proofs are established by simulations which widely use the tools presented in the chapter of Delorme and Mazoyer [18] and exploit the malleability of the dependency graphs.

3.1 Basic equivalences among the low complexity classes

As an introduction, the figure below gives a general overview of the main relationships among the complexity classes in dimension one.

$$\begin{array}{ccc} \mathrm{PCA} = \mathrm{SCA} = \mathrm{DSpace}(n) & & \\ \cup\mathrel{|} & & \\ \mathrm{POCA} = \mathrm{SOCA} & & \\ \cup\mathrel{|} & & \\ \mathrm{LPCA} = \mathrm{LSOCA} = \mathrm{LSCA} & & \\ \cup\mathrel{|} & & \\ \mathrm{RPCA} = \mathrm{RSOCA} = \mathrm{LPOCA} & & \\ \subsetneq & & \subsetneq \\ \mathrm{RPOCA} & \neq & \mathrm{RSCA} \end{array}$$

In this section, we only focus on the various equivalences between the low complexity classes. We shall return to the equality of POCA and SOCA in Section 3.4, to the incomparability of RPOCA and RSCA and their proper inclusions in RPCA in Section 4.1. Further discussions will also follow about

the famous questions whether the inclusions RPCA $\subseteq$ PCA and RPCA $\subseteq$ LPCA are strict in Section 6.1 and in Sections 3.5 and 6.2.

The original proofs of the positive relationships can be found in the following papers. The equality LPCA = LSOCA comes from [25] where it was observed that every PCA working in time $T(n)$ can be simulated by SCA in time $n+T(n)$. The equality LSOCA = LSCA has been noticed in [27]. The equality LPOCA = RPCA is from [9] and the equality RSOCA = LPOCA from [24]. Further relationships between SOCA and POCA in higher dimension have been reported in [55], in particular the equality $d-$RSOCA $= d-$LPOCA and the inclusions $d-$RSOCA $\subseteq$ $(d+1)-$RPOCA and $d-$LSOCA $\subseteq$ $(d+1)-$RSOCA. Let us also notice that restricted to unary languages (i.e., languages over a one-letter alphabet), RSCA is as powerful as RPCA.

All these equalities are easily obtained by basic simulations. To construct such simulations between one device $\mathcal{A}$ and another device $\mathcal{B}$, a simple method consists in exhibiting a transformation which maps the dependency graph of the initial device $\mathcal{A}$ into another directed graph and to verify that this mapped graph, modulo slight modifications, fits the dependency graph of the device $\mathcal{B}$.

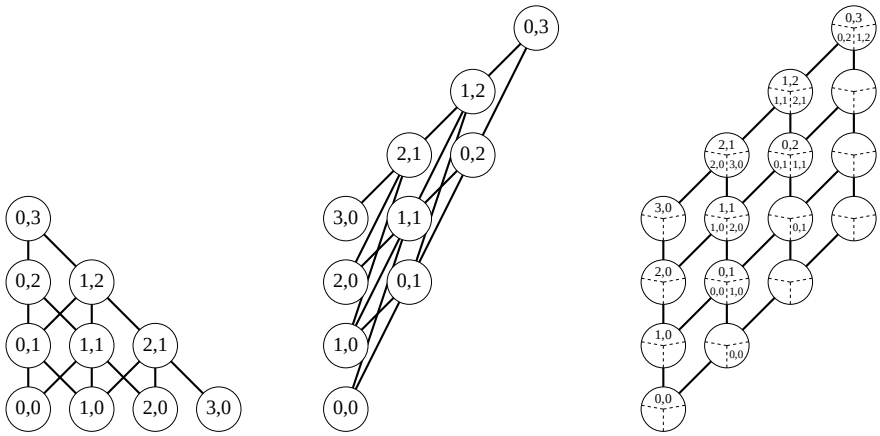


Fig. 9. Simulation of a RPCA $\mathcal{A}$ by a RSOCA $\mathcal{B}$.

To illustrate this, let us consider one example: the inclusion of RPCA into RSOCA. The question is how to simulate the real time computations of a given PCA $\mathcal{A}$ by the real time computations of a SOCA $\mathcal{B}$. The simulation is essentially based on the following transformation $g(c,t) = (t, 2t+c)$. Hence we have to verify that g allows one, with slight modifications, to convert a real time computation of $\mathcal{A}$ into a real time computation of $\mathcal{B}$. For that, we must check that the conditions imposed by the features of the device $\mathcal{B}$

are respected, namely the conditions relating to the structure of the array, the finite memory nature of each cell, the input and output modes and the neighborhood. First, $g(i,0) = (0,i)$ guarantees the conversion from the parallel input mode to the sequential input mode. and $g(0,n-1) = (n-1, 2(n-1))$ ensures the correspondence between the output sites. Second g maps all sites of $\mathcal{A}$ into sites of $\mathcal{B}$, precisely a site of $\mathcal{B}$ is mapped to by at most one site of $\mathcal{A}$. So a finite memory capacity is enough on each cell of $\mathcal{B}$. Finally, on $\mathcal{A}$ governed by the neighborhood $\{-1,0,1\}$, the elementary data movements are $(-1,1),(0,1),(1,1)$. They are converted into the movements $(1,1),(1,2),(1,3)$ which satisfy the dependencies constraints on $\mathcal{B}$. Effectively, the data are transmitted through intermediate sites according to the elementary moves $(1,1)$ and $(0,1)$ and that without exceeding the finite memory capacity of each cell.

An interesting link between one-dimensional devices and two-dimensional devices equipped with a thread, has been observed in [17]. It states that the class of languages recognized in real time by a two-dimensional CA with the Archimedean thread and the Moore neighborhood is strictly contained in the class of languages recognized in real time by one-dimensional PCA. Although the minimal time in $O(\sqrt{n})$ for two-dimensional CA with the Archimedean thread is lower than n the minimal time for PCA, this result is far from being straightforward. Notably, during the simulation, each cell must recover its particular position from the output cell and its neighbors in the initial Archimedean spiral. In fact, it enlightens one on the impact of the way input data are space-distributed, on the recognition power.

At last, notice that for picture languages recognition, PictCA with the von Neumann and Moore neighborhoods can simulate each other with a linear time overhead and so are linear time equivalent. Unfortunately, we ignore finer inclusions concerning PictCA.

3.2 Linear speed up

When investigating complexity classes, an immediate question concerns linear acceleration of the running time. For CA, as the recognition time never goes below real time, a linear speed up corresponds to an acceleration by a constant factor of the running time beyond real time.

Theorem 1. *Let f be a function from $\mathbb{N}$ to $\mathbb{N}$ and rt be the real time function for* PCA *(resp.* SCA*,* POCA *or* SOCA*). For any constant R, a language recognized in time $rt(n) + f(n)$ by a* PCA *(resp.* SCA*,* POCA *or* SOCA*) is also recognized in time $rt(n) + \lceil f(n)/R \rceil$ by another* PCA *(resp.* SCA*,* POCA *or* SOCA*).*

In an early work [1], Beyer demonstrated the speed up result for PCA (included in the case of dimension two). Because [1] was not widely circulated,

the same result can be also found in [29] and in more general settings in [37]. A proof for POCA is in [2] and for SCA and SOCA in [25]. A generalization to SCA in dimension two is given in [30] and to SOCA and POCA in arbitrary dimension in [55].

Let us now recall the usual methods applied to speed up the computation in a linear way. The simplest case is when communication is one-way. Because any two cells are not mutually interdependent, the dependency graph is a directed graph which is acyclic. This characteristic allows one to speed up computation easily. Fig. 10 depicts the scheme for dimension one: once the cell gets the whole input part situated on its left, it can operate R times faster for any integer constant R. This principle can be generalized to higher dimension as long as communication is one-way.

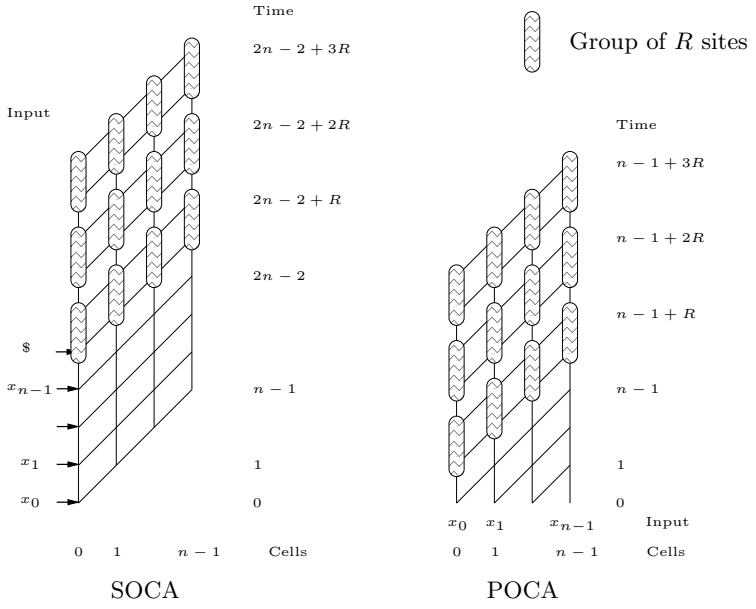


Fig. 10. Linear speed up in case of one-way communication.

When communication is two-way and all cells are interdependent, to reduce the running time requires compacting the space. In grouping cells into fewer ones, the time to exchange information between cells is reduced and so the computation can be achieved at a higher rate. The grouping operation differs according to the input mode.

It is immediate when the input mode is sequential, as all cells have the same initial quiescent state. Initially we have no difficulty in grouping information into fewer cells. Then the accelerated computation can take place immediately

the distinguished cell has obtained the whole input. Fig. 11 illustrates the situation in dimension one. The grouping operation is initially accomplished within each diagonal: for any integer constant R, the sites are grouped together by R. Once the input is read and the end marker is constantly fed, the information initially processed by R diagonals can be processed by a simple one and hence accelerate the computation by a factor R. The fact remains that the time space diagram is distorted. It involves keeping some redundant information in each cell but it preserves the data accessibility requirement while respecting the dependency constraints.

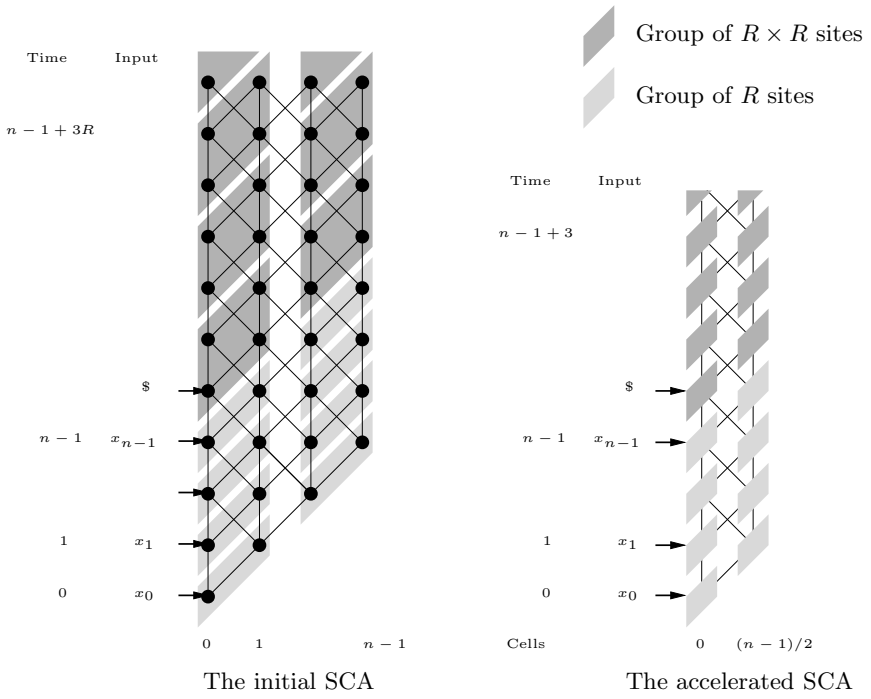


Fig. 11. Linear speed up in case of two-way communication and sequential input mode.

The case of parallel input mode (and two-way communication) is more tricky. The input data are initially fed into the array and some time must be spent grouping them together. In order to avoid losing more time, the accelerated computation must start as soon as possible. Fig. 12 illustrates the method for a PCA $\mathcal{A}$. On the one hand, observe that one can construct a PCA $\mathcal{B}$ which simulates R times faster the PCA $\mathcal{A}$ provided the input given to $\mathcal{A}$ is fed compacted with a factor R to $\mathcal{B}$. On the other hand, there is a grouping PCA which turns the initial ungrouped input into a grouped one. In this way, one can stick the grouping PCA with a twisted variant of the

accelerated PCA $\mathcal{B}$ to obtain the desired PCA. The accelerated computation starts on each cell as soon as its neighbor cells are grouped.

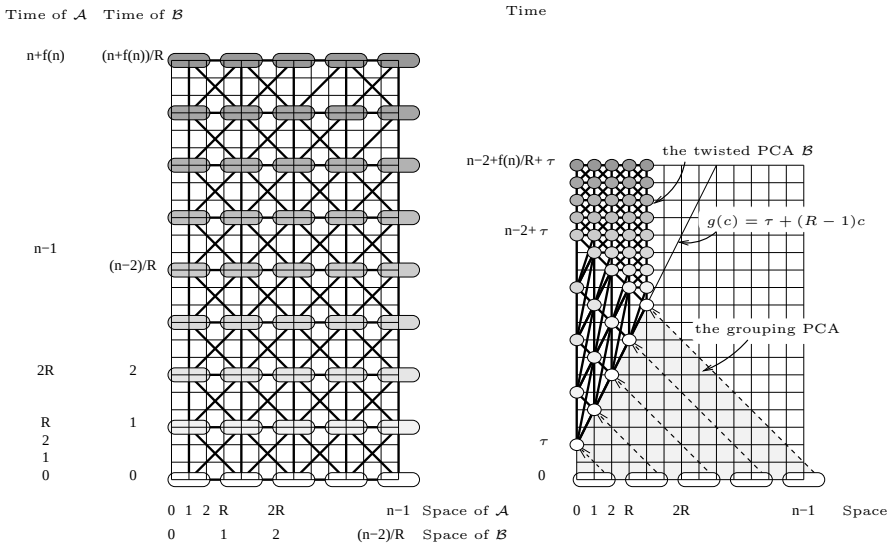


Fig. 12. Linear speed up in case of two-way communication and parallel input mode.

In developing such techniques for picture language recognizer, Beyer in [1] has shown linear speed up result for CA with the von Neumann neighborhood. These techniques work for the Moore neighborhood and similar kinds of neighborhoods as well (See [53]).

Theorem 2. *Let f be a function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$, $\mathcal{N}$ be the von Neumann or Moore neighborhood and $rt_{\mathcal{N}}$ be the corresponding real time function. For any constant R, a picture language recognized in time $rt_{\mathcal{N}}(m,n) + f(m,n)$ by a* PCA *with the neighborhood $\mathcal{N}$ is also recognized in time $rt_{\mathcal{N}}(m,n) + \lceil f(m,n)/R \rceil$ by another* PCA *with the same neighborhood.*

What is disturbing as regards picture language recognition is that some neighborhoods seem not to admit linear speed up. In the common method, each cell implicitly knows in which direction to send its content to achieve the grouping process. This direction corresponds to a shorter path towards the output cell. But for some neighborhoods, this direction differs according to the position of the cell into the array and no alternative efficient grouping process is currently known.

3.3 Constant speed up

In addition to the speed up results of the previous Section 3.2, real time complexity could also be defined modulo a constant. In dimension one, this property was first observed in [9] and a whole complete generalization has be done in [41]. Excluding pathological neighborhoods for which the output cell is not able to read the whole input, we have constant time acceleration for PCA:

Proposition 1. *For any neighborhood $\mathcal{N}$, any function satisfying $f(n) \geq rt_{\mathcal{N}}(n)$ and any constant τ:*

$$\mathrm{PCA}_{\mathcal{N}}(f(n) + \tau) = \mathrm{PCA}_{\mathcal{N}}(f(n))$$

An interesting consequence which emphasizes the robustness of one-dimensional CA model, follows from this speed up (See [41]). The computation ability in dimension one is somewhat independent of the underlying communication graph:

Theorem 3. *In dimension one, all neighborhoods are real-time equivalent to either the one-way neighborhood $\{-1, 0\}$ or the standard one $\{-1, 0, 1\}$.*

The situation turns out to be less satisfactory in higher dimensions. For the two classical von Neumann and Moore neighborhoods, the real time complexity can also be defined modulo a constant. But a uniform approach does not yet exist to deal with various neighborhoods. And in spite of investigations about arbitrary neighborhoods reported in [14], nothing much is known.

3.4 Equivalence between the parallel and sequential input modes

A result that cannot be ignored says that, if we do not bother with the time comparison, the parallel and sequential input modes are equivalent. This result is fairly astonishing in the case of one-way communication. Indeed the parallel input mode combined with one-way communication induces strong limitations on the access to the data. As illustrated in Fig. 13, a cell of a POCA has no access on the input letters applied on its right whereas every cell of a SOCA has access to the whole input. Moreover, for SOCA, the end marker \$ supplied after the whole input letters provides information on the length of the input. On the other hand, such knowledge is impossible for POCA. Despite these outstanding differences, Ibarra and Jiang [24] have shown that SOCA and POCA accept the same class of languages:

Theorem 4. POCA = SOCA

Due to the advantages of SOCA over POCA, the simulation of a POCA by a SOCA is straightforward. In contrast, the reverse simulation is considerably more involved. Let us just present a rough idea of the construction and its difficulties. To simulate a SOCA by a POCA, the basic idea is to systematically

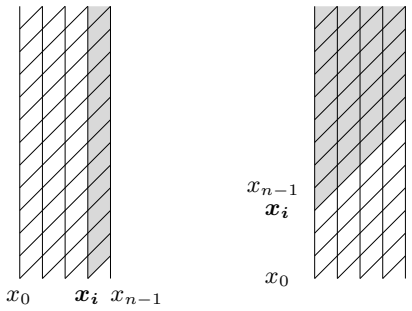


Fig. 13. Influence of the i-th input symbol on a POCA and on a SOCA

generate each possible input, to simulate the SOCA on it and check if it is actually the real input. For that, the working area is divided in two parts. The enumeration process takes place in the left part. First the inputs of length 1, then those of length 2, then those of length 3, etc, are generated in such a way that they can be obtained serially by the leftmost cell of the right part. Simultaneously, in the right part, the SOCA is simulated on the successive generated inputs. Moreover, the successive inputs are compared with the real input.

The delicate point is that the POCA has no hints about the length of the real input and has to deal with inputs of all possible lengths. In particular, the time and the space required to simulate the SOCA on the current input depends on its length. Also this simulation time induces a delay between the generation of two inputs. And the demarcation of the two parts of the working area fluctuates according to the current length of the generated inputs. Then to distribute and to synchronize the different subcomputations into the working area entails many of the techniques described in the chapter [18]. A whole complete description can be found in the original paper [24] and with a direct construction in [15].

The drawback of this construction is its exponential cost. Indeed whatever the time complexity of the initial SOCA, the POCA simulates the behavior of the SOCA on an exponential number of inputs. Additional questions related to the simulation cost of SOCA by POCA will be discussed in the later Section 6.2.

3.5 Closure properties

Closure properties are naturally investigated in order to evaluate the computation ability of the different CA classes and to possibly achieve separation results. Obviously, all deterministic CA classes satisfy the closure under boolean operations. More attention has been paid to the closure properties under other language operations as concatenation, reverse or cycle. As you may recall, the reverse of a language L is $L^R = \{w^R : w \in L\}$ where w^R is the

word w written backwards and its cycle is $L^{Cy} = \{vu : uv \in L\}$. The Table 2 below sums up the known results in dimension one (Y stands for *yes*, N for *no* and ? for *open*).

class	reverse closure	concatenation closure	cycle closure
PCA	Y	Y	Y
POCA	Y	Y	Y
LPCA	Y	?	?
RPCA	?	?	?
RSCA	N	N	N
RPOCA	Y	N	N

Table 2. Closure properties in dimension one.

Let us give further precisions about these results and the open questions. According to the fact that PCA has the same computation power as linear space Turing machine, it satisfies various closure properties: reverse, concatenation, Kleene star, ε-free homomorphism, inverse homomorphism, cycle ... We know from [8] that the one-way counterparts POCA and SOCA also share the above closure properties. The proofs make essential use of similar constructions as the one outlined in the previous Section 3.4 to show that POCA = SOCA. In Section 4.1, we shall account for the negative closure properties of the RSCA class and its higher dimension counterparts and then for the negative closure under concatenation of RPOCA. Curiously enough, this last result is the opposite in higher dimension: for any dimension $d > 1$, $d-$RPOCA satisfies closure under concatenation as well as closure under Kleene star [55]. As observed in [9], the closure under reverse of RPOCA follows from the symmetry of its dependency graph. For the same structural reason, the result extends to the higher dimensions [55]. The closure under reverse of LPCA is an immediate consequence of the linear speed up results [27].

It is not known whether RPCA is closed under reverse or under concatenation. But a striking result due to Ibarra and Jiang [27] relates the property of RPCA to be closed under reverse and its ability to be as powerful as LPCA. In a similar way, the cycle closure property of RPCA can be linked to the equality between RPCA and LPCA [54]. The following theorem gathers these relationships.

Theorem 5. *The following three statements are equivalent:*

- RPCA = LPCA
- RPCA *is closed under reverse*
- RPCA *is closed under cycle*

This theorem is very interesting and deserves some explanations. Let us focus on the equivalence between the ability of RPCA to be as powerful as LPCA and its reverse closure property. The proof relating to the third statement makes essential use of similar arguments. As LPCA is closed under reverse, the equality RPCA = LPCA directly implies the closure under reverse of RPCA. To show the converse implication is more tricky. How may a closure property involve speed up of linear time computation? First the assertion LPCA = RPCA can be restated in the following equivalent statement "Let c be any constant. If $\{w\sharp^{c2^{\lceil \log |w| \rceil}} : w \in L\}$ is a RPCA language then L is a RPCA language". The key argument developed in [27] is that the reverse form of this statement is true.

Lemma 1. *Let c be any constant. If $\{\sharp^{c2^{\lceil \log |w| \rceil}} w : w \in L\}$ is a* RPCA *language then L is a* RPCA *language.*

The proof of this meaningful lemma is fairly involved. A whole description can be found in the original paper [27] and with a direct construction in [15]. Now according to this lemma 1 and providing RPCA is closed under reverse, we get the following chain of implications:

$L \in$ LPCA
$\Rightarrow$ $\widetilde{L} = \{w\sharp^{c2^{\lceil \log |w| \rceil}} : w \in L\} \in$ RPCA (straightforward property)
$\Rightarrow$ $\widetilde{L}^{\mathrm{R}} = \{\sharp^{c2^{\lceil \log |w| \rceil}} w : w \in L^{\mathrm{R}}\} \in$ RPCA (reverse closure)
$\Rightarrow$ $L^{\mathrm{R}} \in$ RPCA (Lemma 1)
$\Rightarrow$ $L \in$ RPCA (reverse closure)

An immediate consequence of Theorem 5 is that if RPCA is closed under reverse or under cycle then it is closed under concatenation [27]. Whether the converse is true, is still open. We just know a weaker implication also based on Lemma 1 [54]:

Proposition 2. *If* RPCA *is closed under concatenation then*

- LPCA *unary languages are* RPCA *unary languages;*
- LPCA *is closed under concatenation.*

Here let us have a look at closure properties of picture language recognizers. Up to now the closure properties under concatenation have not been studied. In the matter of the reverse operation, its counterpart for picture languages is the 180° rotation operation. As in the one-dimensional case, the linear speed up results entail the closure under rotation of linear time PictCA with the von Neumann or Moore neighborhood. In contrast, we shall see in Section 4.1 that real time PictCA with the Moore neighborhood, real time and linear PictCA with the one-way von Neumann or one-way Moore neighborhood do not satisfy closure under rotation. Interestingly, the arguments developed by Ibarra and Jiang to relate, in dimension one, closure properties to computation ability can be extended to PictCA recognizer [52]:

Proposition 3. *Real time* PictCA *with the von Neumann neighborhood is closed under rotation if and only if real time and linear time* PictCA *with the von Neumann neighborhood are equivalent.*

Let us notice at last a nice property pointed out by Szwerinski [47]. In dimension one, a PCA may confuse right and left without time loss. Formally, a local transition function $\delta : S^3 \to S$ is said symmetrical if for all $r, c, l \in S$ holds: $\delta(l, c, r) = \delta(r, c, l)$. So the following proposition, as shown in [47] and in a more general setting in [32], underlines that the orientation does not matter for one-dimensional PCA.

Proposition 4. *If a language L is recognized in time T by some* PCA *then L is recognized in the same time T by another* PCA *whose transition function is symmetrical.*

4 Limitations

Our purpose in this section will be to present the known limitations on the recognition power of CA. It makes use of either algebraic arguments or diagonalization ones.

4.1 Algebraic arguments

Let us start with a basic negative result. As observed in [12], POCA operating in real time on languages over a one-letter alphabet are no more powerful than finite automata:

Proposition 5. *The class of unary languages recognized by* RPOCA *is the class of regular unary languages.*

This result has been strengthened in [4]: it requires at least an amount of $n + \log n$ time to recognize non regular unary languages on POCA.

An interesting consequence of Proposition 5, observed in [9], derives from the existence of non regular unary languages which are recognized by RSCA and therefore by RPCA. For instance, the languages $\{1^{2^n} : n \in \mathbb{N}\}$ and $\{1^p : p \text{ is a prime}\}$ belong to RSCA (See [9, 20] and the chapter of Mazoyer and Yunès [38]). It yields the following relationships.

Corollary 1.
- RPOCA $\subsetneq$ RPCA
- RSCA $\not\subseteq$ RPOCA

Let us now concentrate on elaborate techniques to obtain lower bounds. These techniques have been introduced by Hartmanis and Stearns [23] for real time Turing machines and adapted to real time SCA by Cole [10]. Using counting arguments, they exploit limits on interaction between data.

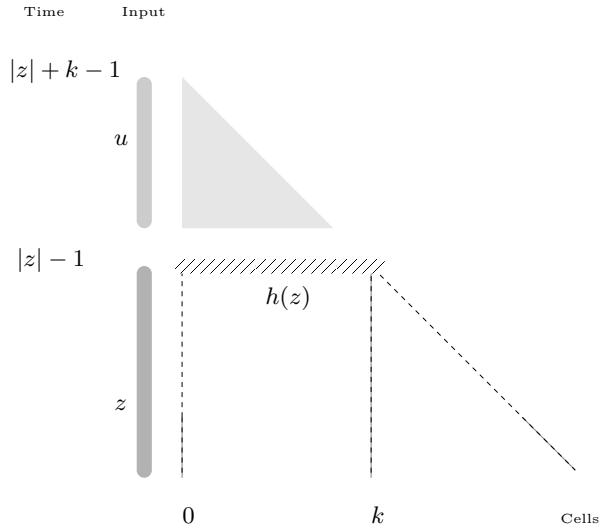


Fig. 14. Real time computation of a SCA on some input $w = zu$.

The method for real time SCA.

Let us recall the method used in [10] to get negative results on RSCA. As illustrated in Fig. 14, a characteristic of the real time SCA computation on a given input w is the following one. The suffix of arbitrary length k of w only has an impact on the computation during the k last steps. In particular, the first part z and the suffix u of length k of the input $w = zu$ interact on a number of cells independent of the length of z. Hence significant information on z may be lost before the k last steps of the computation. Precisely, consider the configuration of the SCA at time $|z| - 1$ when the last symbol of z is fed. At this time, the useful part consists of the $k+1$ first sites which may influence the result of the computation obtained on the output cell, k steps after. Let $h(z)$ denote the sequence of states of length $k+1$ of this useful part. Thus the result of the computation of the RSCA on the input zu is completely specified by $h(z)$ and u. That sets an upper bound on the number of distinct behaviors and thus implies the following condition on the structure of RSCA languages.

Proposition 6. *Let L be a language over some alphabet Σ. Let X be a finite subset of Σ^* and let k be the maximal length of the words in this set X. Consider, for each word $z \in \Sigma^*$, the indicator function p_z from X into $\{0,1\}$ defined as:*

$$p_z(u) = \begin{cases} 1 \text{ if } zu \in L \\ 0 \text{ otherwise} \end{cases}$$

If the language L is recognized in real time by some SCA *then the number of distinct functions $|\{p_z : z \in \Sigma^*\}|$ is of order at most $2^{O(k)}$.*

Proof. Suppose that L is recognized by some RSCA $\mathcal{A}$. For any words $z, z' \in \Sigma^*$, observe that if $h(z) = h(z')$ then for all $u \in X$ either zu and $z'u$ are both accepted or both rejected by $\mathcal{A}$. Indeed the words u in X have length at most k and $h(z)$, as defined above, consists of the information accessible to the k last steps. Hence if $h(z) = h(z')$ then $p_z = p_{z'}$. Now notice that every h(z) is a word of length $k+1$ on the finite set of states of $\mathcal{A}$. Therefore, the number of distinct functions p_z does not exceed the number of distinct sequences $h(z)$ which is of order $2^{O(k)}$. □

We shall now give a couple of examples.

Example 6 *$L = \{x_1\sharp x_2\sharp\cdots x_t\sharp x\colon x_1, x_2, \cdots, x_t, x \in \{0,1\}^*$ and $x = x_j$ for some j with $1 \leq j \leq t\}$ is not a real time* SCA *language.*

Proof. Set $X = \{0,1\}^k$. Associate to each subset A of X, the word $z_A = x_1\sharp x_2\sharp\cdots x_t\sharp$ where $x_1, x_2, \cdots, x_t$ is some enumeration of the words in A. Namely, z_A is defined in such a way that $z_A u \in L$ if and only if $u \in A$. Thus, if A and B are two distinct subsets of X then $p_{z_A} \neq p_{z_B}$. So the number of functions $|\{p_z\colon z \in \Sigma^*\}|$ is at least 2^{2^k}, the number of subsets of X. Therefore, according to Proposition 6, L is not a *RSCA* language. □

The following example differs from the previous one in that its complexity is in regard of the number of functions p_z.

Example 7 *$L = \{01^{a_1}01^{b_1}01^{a_2}01^{b_2}\cdots 01^{a_t}01^{b_t}01^a01^b\colon a_1, b_1, \cdots, a_t, b_t \geq 0$ and $a = a_j, b = b_j$ for some j with $1 \leq j \leq t\}$ is not a real time* SCA *language.*

Proof. Set $X = \{01^a01^b\colon a, b \geq 0$ and $a+b+2 \leq k\}$. Associate to each subset A of X, the word $z_A = x_1\cdots x_t$ where $x_1, \cdots, x_t$ is some enumeration of the words in A. As z_A is defined, $z_A u \in L$ if and only if $u \in A$. Thus, if A and B are two distinct subsets of X then $p_{z_A} \neq p_{z_B}$. So the number of functions $|\{p_z\colon z \in \Sigma^*\}|$ is at least $2^{k(k-1)}$, the number of subsets of X. Hence L is not a *RSCA* language. □

Several other languages are known not to belong to RSCA. Among them the languages which have been shown not real time recognizable by multitape Turing machines in [23, 43], are also not real time recognizable by SCA. Indeed both devices share the same features on data accessibility which are exploited to get limitations. Actually, all these languages are RPCA languages. As a consequence, RSCA is less powerful than RPCA. Furthermore, specific examples exhibited in [23, 43, 10, 19, 35] yield several negative properties. A representative one is the language of words which end with a palindrome of length at least 3: $L = \{w \in \Sigma^*\colon w = uv, v = v^R, |v| > 2\}$. Yet L is linear context free and belongs to RPOCA, furthermore the reverse of L and the palindrome language belong to RSCA. Therefore RSCA contains neither all

linear context free languages nor RPOCA languages and is not closed under reversal and under concatenation. The following corollary summarizes the various results obtained.

Corollary 2. • RSCA *is strictly contained in* RPCA.
- RSCA *and* RPOCA *are incomparable.*
- RSCA *is not closed under reversal, concatenation, Kleene closure, sequential mapping, nor the operations of taking derivatives and quotients.*
- RSCA *does not contain all deterministic linear context free languages.*

One further noteworthy result of Cole in [10] is that the power of real time SCA increases with the dimension of the space.

Proposition 7. *For any dimension* d, $d-\mathrm{RSCA} \subsetneq (d+1)-\mathrm{RSCA}$.

Proof. The number of cells which influence the k last steps of any $d-\mathrm{RSCA}$ computation increases polynomially with the dimension d. Precisely, the counterpart of $h(z)$ in dimension d, is a d-dimensional word of diameter $O(k)$ and volume $O(k^d)$. Hence the number of distinct words $h(z)$ is in $2^{O(k^d)}$. Furthermore, the condition stated in Proposition 6 can be rewritten in this way: "If the language L is recognized in real time by some $d-\mathrm{SCA}$ then the number of distinct functions $|\{p_z : z \in \Sigma^*\}|$ is of order at most $2^{O(k^d)}$." As a consequence the language presented in Example 6 is not a $d-\mathrm{RSCA}$ language whatever the dimension d may be. On the other hand, the language given in Example 7 is not a $1-\mathrm{RSCA}$ language but it may be, and in fact is a $2-\mathrm{RSCA}$ language. Of course, languages with the same kind of structure as the one of Example 7 can be built to separate $d-\mathrm{RSCA}$ and $(d+1)-\mathrm{RSCA}$. □

The method for real time POCA.

Let us now look at the case of RPOCA. A first characteristic noticed in [11] is that the real time computation on an input w contains the real time computations of all its factors. Recall the Example 4 in Section 2: the RPOCA which tests whether an input is a Dyck word, processes together all its factors. So the evolution on an input of size n decides the memberships of $n(n+1)/2$ words. This constraint has been exploited in [48] to get a non RPOCA language.

A second characteristic of RPOCA computation noticed in [49] is the following one. As depicted in Fig. 15, on an input w, its prefixes u and suffixes v only interact during the $|uv|-1$ last steps. More precisely, on input $w = uzv$, consider at time $|z|$ the useful part which consists of the $|uv|$ sites which may have an impact on the result obtained $|uv|-1$ steps after. This part subdivides into the first $|u|$ sites named $h_z(u)$ and the last $|v|$ sites named $h'_z(v)$. Note that the result of the computation on uzv is completely specified by $h_z(u)$ and $h'_z(v)$. Furthermore, according to the first characteristic, the results of all the factors which contained z are determined by $h_z(u)$ and $h'_z(v)$. On the other hand, $h_z(u)$ is not influenced by the suffix v as well as $h'_z(v)$ by the prefix

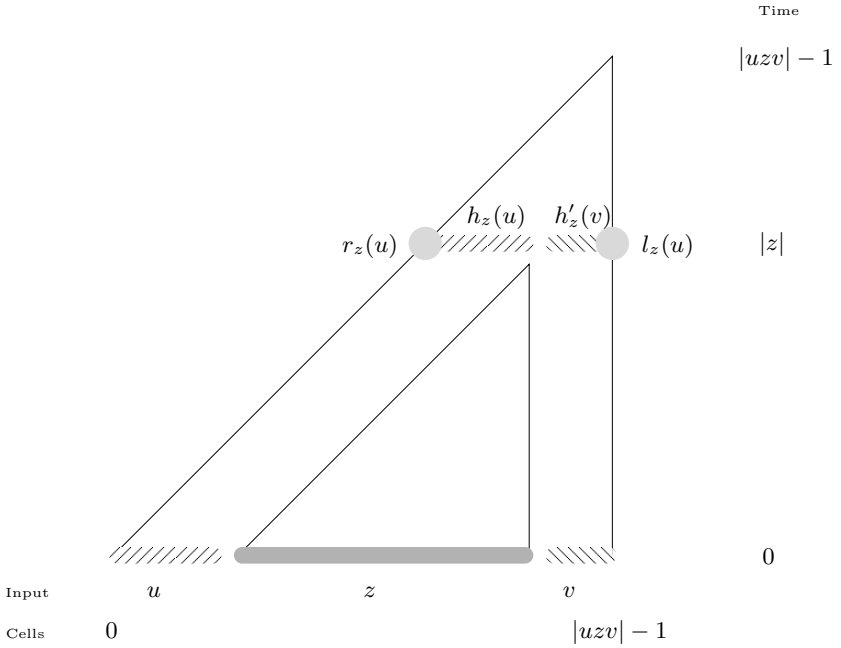


Fig. 15. Real time computation of a POCA on some input uzv.

u. Hence, intuitively the information exchange between u and v takes place when significant information about z may be lost. This situation gives rise to the following condition on the structure of RPOCA languages.

Proposition 8. *Let L be a language over some alphabet Σ. Let X, Y be two sets of words on Σ. Denote the set of all suffixes of words in X by* $\mathrm{Suff}(X)$ *and the set of all prefixes of words in Y by* $\mathrm{Pref}(Y)$*. Consider, for each word $z \in \Sigma^*$, the indicator function p_z defined as:*

$$p_z : \mathrm{Suff}(X) \times \mathrm{Pref}(Y) \to \{0,1\}$$
$$(u,v) \mapsto \begin{cases} 1 \text{ if } uzv \in L \\ 0 \text{ otherwise} \end{cases}$$

If L is recognized in real time by some POCA *then the number of distinct functions $|\{p_z : z \in \Sigma^*\}|$ is of order at most $2^{O(|\mathrm{Suff}(X)|+|\mathrm{Pref}(Y)|)}$.*

Proof. Suppose that L is recognized by some RPOCA $\mathcal{A}$. Denote S the set of states of $\mathcal{A}$. Consider the two functions l_z from $\mathrm{Suff}(X)$ into S and r_z from $\mathrm{Pref}(Y)$ into S defined in this following way. To every u in $\mathrm{Suff}(X)$, l_z associates the state entered at time $|z|$ by the leftmost cell involved in the real time computation of $\mathcal{A}$ on input uz. Symmetrically, to every v in $\mathrm{Pref}(Y)$, r_z associates the state entered at time $|z|$ by the rightmost cell involved in the real time computation of $\mathcal{A}$ on input zv. Observe that $h_z(u)$, as defined

above, is the sequence of $r_z(x)$ where x ranges over the set of all suffixes of u: $h_z(u_1u_2\cdots u_k) = r_z(u_1u_2\cdots u_k)r_z(u_2\cdots u_k)\cdots r_z(u_k)$. In a symmetric way, $h'_z(v)$ is the sequence of $l_z(y)$ where y ranges over the set of all prefixes of v: $h'_z(v_1\cdots v_{k-1}v_k) = l_z(v_1)\cdots l_z(v_1\cdots v_{k-1})l_z(v_1v_2\cdots v_k)$. Therefore, if $r_z = r_{z'}$ and $l_z = l_{z'}$, then for all $u \in \mathrm{Suff}(X)$ and $v \in \mathrm{Pref}(Y)$ either uzv and $uz'v$ are both accepted or both rejected by $\mathcal{A}$. In other words, if $r_z = r_{z'}$ and $l_z = l_{z'}$ then $p_z = p_{z'}$. Therefore, the number of distinct functions p_z does not exceed the product of the number of distinct functions r_z with the number of distinct functions l_z which is in $2^{O(|\mathrm{Suff}(X)|+|\mathrm{Pref}(Y)|)}$. □

Example 8 *$L = \{x\sharp x_1\$y_1\sharp x_2\$y_2\sharp\cdots\sharp x_t\$y_t\sharp y\colon x_1, y_1, x_2, y_2\cdots, x_t, y_t, x, y \in \{0,1\}^*$ and $x = x_j, y = y_j$ for some j with $1 \le j \le t\}$ is not a real time* POCA *language.*

Proof. Set $X = Y = \{0,1\}^k$. Associate to each subset A of $\mathrm{Suff}(X)\times\mathrm{Pref}(Y)$, the word $z_A = \sharp x_1\$y_1\sharp x_2\$y_2\sharp\cdots\sharp x_t\$y_t\sharp$ where $(x_1, y_1), \cdots, (x_t, y_t)$ is some enumeration of the words in A. By construction, $x\, z_A\, y \in L$ if and only if $(x, y) \in A$. Thus, if A, B are two distinct subsets of $\mathrm{Suff}(X) \times \mathrm{Pref}(Y)$ then $p_{z_A} \neq p_{z_B}$. Therefore, the number of distinct functions $|\{p_z\colon z \in \Sigma^*\}|$ is at least $2^{2^{2k}}$ the number of subsets of $\mathrm{Suff}(X) \times \mathrm{Pref}(Y)$. It is of greater order than $2^{O(|\mathrm{Suff}(X)|+|\mathrm{Pref}(Y)|)} = 2^{O(2^k)}$. □

Example 9 *Let consider the linear context free language $L_1 = \{w\colon w = 1^u0^u$ or $w = 1^u0y10^u$ with $y \in \{0,1\}^*$ and $u > 0\}$. The context free language $L = L_1 \cdot L_1$ is not a real time* POCA *language.*

Proof. Fix some integer k. Set $X = \{1^k\}$ and $Y = \{0^k\}$. Associate to each subset A of $\mathrm{Suff}(X) \times \mathrm{Pref}(Y)$, the word $z_A = 0^{i_1}1^{j_1}\cdots 0^{i_t}1^{j_t}$ where $(1^{i_1}, 0^{j_1}), \cdots, (1^{i_t}, 0^{j_t})$ is some enumeration of the words in A. As z_A is defined, $1^u z_A\, 0^v \in L$ if and only if $(1^u, 0^v) \in A$. Thus, if A, B are two distinct subsets of $\mathrm{Suff}(X) \times \mathrm{Pref}(Y)$ then $p_{z_A} \neq p_{z_B}$. Therefore, the number of distinct functions $|\{p_z\colon z \in \Sigma^*\}|$ is at least $2^{(k+1)^2}$ the number of subsets of $\mathrm{Suff}(X) \times \mathrm{Pref}(Y)$. It is of greater order than $2^{O(|\mathrm{Suff}(X)|+|\mathrm{Pref}(Y)|)} = 2^{O(k)}$. We conclude according to Proposition 8 that L is not a RPOCA language. □

Since linear context free languages are RPOCA languages, an immediate consequence of Example 9 is that RPOCA is not closed under concatenation and does not contain all context free languages. With further results obtained in [34], the following corollary gives us the main negative properties of RPOCA.

Corollary 3. • *The class of real time* POCA *languages is not closed under concatenation, Kleene closure, ε-free homomorphisms.*
- *The class of real time* POCA *languages does not contain all context free languages.*

Furthermore, as shown in [3, 34], the previous arguments combined with padding techniques lead to an infinite hierarchy of separated classes between real time SCA and linear time SCA as well as another one between real time POCA and linear time POCA. For the sake of Proposition 9 below, the definition of Fischer's constructibility can be found in the chapter of Delorme and Mazoyer [18]. At least, the next function T can be viewed as a "reasonable" function of asymptotic order at most n.

Proposition 9. • *Let T, T' be two functions from $\mathbb{N}$ to $\mathbb{N}$ such that the inverse of T is Fischer constructible and $T' \in o(T)$.*
We have the strict inclusion $\mathrm{SCA}(n + T'(n)) \subsetneq \mathrm{SCA}(n + T(n))$.
- *Let T, T' be two functions from $\mathbb{N}$ to $\mathbb{N}$ such that the inverse of T is Fischer constructible and $T' \log(T') \in o(T)$.*
We have the strict inclusion $\mathrm{POCA}(n + T'(n)) \subsetneq \mathrm{POCA}(n + T(n))$.

Similarly, such algebraic techniques may be applied to exhibit limits on the computation ability of picture language recognizers. In this way, weakness of real time computation with the Moore neighborhood has been observed in [50]. Precisely, there exists a picture language which is real time recognizable by no PictCA with the Moore neighborhood. Furthermore, this picture language is real time recognizable by a PictCA with the von Neumann neighborhood and its corresponding language obtained by rotation of 180^o is accepted in real time with both the von Neumann and Moore neighborhoods.

In the same vein, restricted communication has been shown to reduce the computational power of low complexity picture classes [54]. Precisely, there exists a language recognized in real time by PictCA with both the von Neumann and Moore neighborhoods but not recognized in linear time with any one-way neighborhoods $\mathcal{N}$ such that $a + b \geq 0$ for every (a, b) in $\mathcal{N}$. Furthermore, the corresponding language obtained by rotation of 180^o is recognized by PictCA with one-way neighborhoods $\mathcal{N}_1 = \{(0,0),(0,1),(1,0)\}$, $\mathcal{N}_2 = \{(0,0),(0,1),(1,0),(1,1)\}$ or $\mathcal{N}_3 = \{(0,0),(0,1),(1,0),(1,1),(-1,1),(1,-1)\}$.

The various consequences are summarized in the following proposition.

Proposition 10. • *Real time* PictCA *with the Moore neighborhood does not contain real time* PictCA *with the von Neumann neighborhood.*
- *Real time* PictCA *with the Moore neighborhood is not closed under rotation.*
- *Real time* PictCA *with the Moore and von Neumann neighborhoods are not contained in linear time* PictCA *with a one-way neighborhood $\mathcal{N}$ where $a + b \geq 0$ for all $(a, b) \in \mathcal{N}$.*
- *Real time and linear time* PictCA *with the one-way neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$ are not closed under rotation.*

As emphasized by Cervelle and Formenti in [5], we can as well make use of Kolmogorov complexity to derive these results. This alternative approach

expresses in a more direct manner the fact that significant information is lost. Here, we shall not recall the formal definition of Kolmogorov complexity, but just give an example to illustrate the method. The reader is referred to [5] for comprehensive definitions and for an example involving POCA device.

A basic example is the language not belonging to RSCA which was presented in Example 6: $L = \{x_1\sharp x_2\sharp\cdots\sharp x_t\sharp x\colon x_1, x_2, \cdots, x_t, x \in \{0,1\}^*$ and $x = x_j$ for some j with $1 \leq j \leq t\}$. Contrary to the algebraic method, we will not take into account all possible evolutions, but just focus on a "complex" one. Let us fix ω some Kolmogorov random number of length 2^k. Consider some bijection val between $X = \{0,1\}^k$ and $\{0, \cdots, 2^k - 1\}$ which codes each word of X by a distinct integer below 2^k, for instance $val(a_0 \cdots a_{k-1}) = \sum a_i 2^i$. Consider the set $A_\omega = \{x \in X\colon$ the symbol of rank $val(x)$ in ω is a $1\}$ and $z_\omega = x_1\sharp\cdots x_t\sharp$ where $x_1, \cdots, x_t$ is some enumeration of the words in A_ω. Now if L is recognized in real time by some SCA $\mathcal{A}$ then we can reconstruct ω from the description of $\mathcal{A}$, the description of the bijection val and the sequence of states $h(z_\omega)$. Indeed, from $\mathcal{A}$ and $h(z_\omega)$, we can decide, for each $x \in X$, whether $z_\omega x$ is in L or not, and so whether the symbol of rank $val(x)$ in ω is a 1 or a 0. The respective lengths of these descriptions are in $O(1)$ for $\mathcal{A}$, in $O(\log k)$ for val and in $O(k)$ for $h(z_\omega)$ and their sum is less than the 2^k bits of the word ω. Hence it leads to a contradiction on the incompressibility of ω.

Undoubtedly, algebraic arguments and also Kolmogorov complexity are powerful tools to establish limits on the computation ability of restricted devices with low complexity. But we have to admit that many questions on the power of these devices are left open. For instance, the same arguments are exploited to prove that some languages do not belong to real time Turing machines nor RSCA. Does it mean that real time SCA are no more powerful than real time Turing machines? Moreover, most of the witness languages which enable us to derive negative properties, are usually built in an *ad hoc* manner. Then we fail to determine the status of more "natural" languages like, for RPOCA, the majority language $\{w \in \{0,1\}^*\colon w$ has more 1's than 0's$\}$ or the square language $\{ww : w \in \{0,1\}^*\}$.

4.2 Diagonalization arguments

In computational complexity, many separation results use diagonalization techniques. Diagonalization also works to separate CA classes but essentially in an indirect way via the Turing machines. It consists in exhibiting efficient simulations of CA by Turing machines which allow us to translate Turing machine results into CA results. In this regard, Goldschlager [22] has shown that whatever its dimension may be, a CA which works in time T can be simulated by a Turing machine in space T.

Fact 1 *For any dimension d and any complexity function T, $d-\mathrm{PCA}(T) \subseteq \mathrm{DSpace}(T)$.*

Indeed, due to the regularity of the dependency graphs, the simulation which consists of a depth first search of these graphs of height T, can be performed in space T by a Turing machine. In addition, this result holds as well as for CA as picture recognizer. As a consequence, for CA in dimension 2 and higher, separation follows from the Turing machine space hierarchy. In particular, for language and picture recognition, in dimension greater than 1, CA working in linear time are strictly less powerful than CA working in unrestricted time and that within the same bounded space.

In the case of restricted communication, a better simulation has been obtained for POCA as language recognizer [7].

Fact 2 *For any dimension d,* $d-\mathrm{POCA} \subseteq \mathrm{DSpace}(n^{2-1/d})$.

It allows us to show that, in dimension 2 and higher, restricted communication reduces the language recognition ability. Precisely, there is a language accepted by a $2-\mathrm{PCA}$ that cannot be accepted by any $d-\mathrm{POCA}$ whatever the dimension d. Yet the question is still open in the case of PictCA.

5 Comparison with other models

In this section, we shall compare CA with other computational models. Such investigations may help to identify significant features of CA.

5.1 Sequential models

In order to determine to what extent the use of parallelism provides significant advantages, one of our major concerns is the relationship between parallel and sequential computation.

First, efficient simulations of CA by sequential devices permits us to render explicit limitations on the CA computation ability. Notably, as seen in the previous section, the simulations which connect CA time complexities with Turing space complexities, entail separation results for CA. But to relate the CA time with the Turing time, there seems to be no better simulation than the trivial one which states that the work of a CA performed within time T and space S can be done by a Turing machine within time $T \times S$ and space S. For the other representative sequential model, the random access machine, a lower overhead has been obtained in [26]. By the way of a precomputation phase, it has been shown that any $d-\mathrm{POCA}$ working in time $T(n)$ can be simulated by a unit cost random access machine in time $n^d T(n)/\log^{1+1/d} T(n)$.

Conversely, the simulation of sequential devices by CA is a great challenge. What gain may be achieved with CA? As regards specific sequential problems, the CA computation power is manifest. For instance, it has been observed in [28] that real time POCA contains a language which is P-complete and

in [8] that the language QBF of true quantified boolean formulas which is PSpace-complete, is in POCA.

But when we try to get general simulations, we come up against the delicate question of whether parallel algorithms are always faster than sequential ones. Indeed, there is no guarantee that efficient parallelization is always possible. Or there might exist a faster CA for each singular sequential solution whereas no general simulation exists. Besides, let us recall Fischer's algorithm to recognize the set $\{1^p : p \text{ is a prime}\}$ [20] and Cole's and Culik's ones seen in section 2. They suggest that clever strategies in parallel are stupid in sequential and *vice versa*. Then when the conception of efficient parallel algorithms makes use of radically different techniques from the sequential ones, automatic parallelization appears highly improbable.

Hence without surprise, the known simulation of Turing machines by CA provides no parallel speed up. As viewed in the chapter of Ollinger [40], the early construction of Smith in [46] mimics one step of computation on a Turing machine (with an arbitrary number of tapes) in one step on a CA (with unbounded space). Furthermore, no effective simulations have been proposed even for restricted variants. For instance, we do not know whether any finite automata with k heads can be simulated on CA in less than $O(n^k)$ steps which is the sequential time complexity. And we wonder whether one-way multihead finite automata whose sequential time complexity is linear, may be simulated in real time on CA.

In contrast to this great ignorance, the result of Kosaraju is noteworthy [33].

Proposition 11. *Any picture language recognized by a* 4*-way finite automata can be recognized in linear time by a* PictCA*.*

Unfortunately, a complete proof has never been published. Let us just outline the basic idea. The key point is to code the behavior of the finite automaton on a block of size $n \times n$ by a directed graph. A vertex of such a graph is a couple (q, n) where q is an automaton state and n a boundary node of the block. Then the directed graph records for each couple of vertices $((q_1, n_1), (q_2, n_2))$, if, when the automaton enters in state q_1 at boundary node n_1, it exits in state q_2 at boundary node n_2. The trick is that the space to record the adjacency matrix of this graph is of the same order as the corresponding block. Furthermore, the adjacency matrix of a block of size $2^i \times 2^i$ can be effectively computed from the four adjacency matrices of the four sub-blocks of size $2^{i-1} \times 2^{i-1}$. To this end, the four adjacency matrices are reorganized in one, the transitive closure is computed and then the new non-boundary nodes are eliminated. Now using this procedure recursively, the adjacency matrix of the whole initial pattern can be computed and that in linear time.

5.2 Alternating automata and alternating grammars

The correspondence between CA and alternating finite automata was first pointed out by Ito *et al.* In [31], they showed the equivalence between a particular variant of two dimensional CA and a restricted type of two dimensional alternating finite automata. In [51, 55], alternating analogs of real time CA with sequential input mode have been given as follows.

Proposition 12. • *Real time* $d-$SCA *are equivalent through reverse to real time one-way alternating finite automata with* d *counters.*
• *Real time* $d-$SOCA *are equivalent through reverse to one-way alternating finite automata with* $d+1$ *heads.*

Similarly, for CA with parallel input mode which implicitly induces a synchronization at initial time, we might search a characterization in terms of one-way synchronized finite automata. Yet, these equivalences are somewhat unsatisfying in the sense that the corresponding types of alternating finite automata provide no further information about the computation power of CA. Moreover writing algorithms is more intuitive for CA than for alternating devices.

Introducing the notion of alternating grammar in [39], Okhotin has exhibited a characterization of RPOCA which gives some insight on the relationship between CA and the Chomsky hierarchy. First let us briefly present the alternating grammars. An alternating grammar is a grammar enhanced with a conjunctive operation denoted by &. Each production is of the form $\alpha \to \alpha_1 \,\&\, \cdots \,\&\, \alpha_k$ where $\alpha, \alpha_1, \cdots, \alpha_k$ are strings over a set of variables and terminals. Such a production denotes that the language generated by α is the intersection of the languages generated by $\alpha_1, \cdots, \alpha_k$. Analogously to linear context free grammar, a linear conjunctive grammar is defined as an alternating grammar with the restrictions that for every production $\alpha \to \alpha_1 \,\&\, \cdots \,\&\, \alpha_k$, α is a symbol and no α_i has more than one instance of variable.

From an algorithm given in [45], it was already known that POCA are able to recognize in real time every linear context free languages. Finally, as shown in [39], to extend linear context free grammar with the conjunctive operation &, leads to a complete characterization of RPOCA.

Proposition 13. *The languages recognized in real time by* POCA *are precisely the languages generated by linear conjunctive grammar.*

5.3 Other massively parallel models

There exist other massively parallel computational models than the CA one. Among them, boolean circuits, parallel random access machines (PRAM) and alternating Turing machines attract great attention. Curiously enough, CA and these parallel models seem not to recognize the existence of each other.

Actually the way in which they modelize parallelism, differs on several essential points. On CA, the network structure is homogeneous and the interactions are uniform and local. Furthermore, the d-dimensional array structures usually considered, have a constant expansion rate: from a given cell, the number of cells accessible in $2t$ steps is linearly related to the number of cells accessible in t steps. In contrast, constraints on the network structures of uniform boolean circuits are rather weak. Besides most of the PRAM variants neglect the communication issue that is the bottleneck in physical machines. Another point of discord is the parallel computation thesis which states that parallel time is polynomially equivalent to sequential space. This relationship satisfied by boolean circuits, PRAM and alternating Turing machines, seems not to apply to CA whose network is structured as a d-dimensional array.

Hence, despite their common concern about massively parallel computation, very little is known about their links. At least, it has been proved in [8] that POCA can simulate linear time bounded alternating Turing machine. As a consequence, remark also that POCA contains NSpace($\sqrt{n}$).

6 Questions

Central questions about CA as language recognizer, have emerged from the very beginning and up to now remain without answer. To end this chapter, we shall go back over some emblematic ones.

6.1 The linear time versus linear space question

The first important issue concerning the recognition power of CA is whether, for one-dimensional space bounded CA, minimal time is less powerful than unrestricted time [45]. According to the intuition that more time gives more power, the equality RPCA = PCA between minimal time and unrestricted time CA seems very unlikely. But we fail to separate these classes. Actually, this flaw in knowledge is not specific to parallel computation when we think on similar questions in the computational complexity theory such that L $\stackrel{?}{=}$ P or P $\stackrel{?}{=}$ PSpace and more generally when we wonder how do time and space relate.

As a matter of fact, the equality RPCA = PCA would imply the equality P = PSpace since RPCA is included in P and some PSpace-complete language belongs to PCA. More precisely, using padding techniques, we know from [42] that if RPCA = PCA (in other words, if PCA(f) = DSpace(f) for $f(n) = n$) then for every space constructible function f: PCA(f) = DTime(f^2) = DSpace(f).

The common difficulty in establishing strict hierarchies lies in the fact that the amount of only one resource (here time) is varying while the amount of

a second resource (here space) remains fixed. In that case, classical diagonalization arguments are of no help. And algebraic techniques only work for low level complexity classes.

6.2 The influence of the input mode

When the communication is two-way, the input mode, either parallel or sequential, does not have a great impact on the recognition time. Indeed, in this case whatever the input mode may be, we are free to re-arrange the input in various ways into the space-time diagram within linear time. So PCA and SCA are time-wise equivalent up to linear time complexity.

The situation is not so simple when the communication is one-way. In this case, there exists a strong restriction for parallel input mode on the access to the input. The key point is that the i-th cell of a POCA can only access to the first i symbols of the input whereas every cell of a SOCA has access to the whole input within linear time. Despite this restriction, POCA and SOCA are equivalent. But taking time into account would make the difference. On the one hand, SOCA simulates POCA without time overhead. On the other hand, the only known algorithm to simulate a SOCA by a POCA is based on a brute-force strategy with an exponential cost even when the SOCA works in linear time (cf. Section 3.4).

Now we may wonder whether there exist less costly simulations by POCA for linear time or polynomial time SOCA. This question echoes another famous one: does RPCA equal LPCA? Indeed RPCA, RSOCA and LPOCA are equivalent and also LPCA and LSOCA are equivalent. Thus the question whether RPCA is as powerful as LPCA is the same one as whether LPOCA can simulate LSOCA or not. Further, recall the result of Ibarra and Jiang which relates this question to the closure properties under reverse and under cycle of RPCA (cf. Section 3.5). This result can be restated in this way: LPOCA is as powerful as LSOCA if and only if LPOCA is closed under reverse. Currently, the only idea to recognize, on a POCA, the reverse or the cycle of a language recognized by a LPOCA, is the same one that for the simulation of a SOCA by a POCA: the exhaustive strategy which consists in systematically generating all possible inputs. In other words, we encounter the same obstacle: an exponential cost which is far from the expected linear cost.

Of course it reinforces the belief that LPCA is strictly more powerful than RPCA. Moreover, a close look at the method used by Ibarra and Jiang to link recognition ability and closure properties suggests that the result could be generalized in the following similar way. For POCA, the amount of time sufficient to simulate an arbitrary SOCA of complexity f equals the amount of time sufficient to recognize the reverse (or the cycle) of an arbitrary language accepted in time f by a POCA. The common difficulty is to make explicit the impact of the manner in which the input is supplied in case of one-way communication.

6.3 The neighborhood influence

Another major issue is the impact of the underlying communication graph on the computation ability. First of all, the difference between one-way communication and two-way communication is not well understood. We do not know whether POCA is as powerful as PCA. In fact a strict inclusion between POCA and PCA would separate linear time and linear space CA. But also, as it was stressed in [24], it would improve Savitch's theorem. Indeed, $\mathrm{NSpace}(\sqrt{n}) \subseteq \mathrm{POCA} \subseteq \mathrm{PCA} = \mathrm{DSpace}(n)$. Hence to distinguish POCA and PCA would distinguish $\mathrm{NSpace}(\sqrt{n})$ and $\mathrm{DSpace}(n)$. On the other hand, we are far from claiming the equality of POCA and PCA since we do not even know whether POCA is able to simulate PCA working in quasi-linear time.

However, in dimension higher than one, one-way communication sets limits on the language recognition ability. There exists a language accepted by some $2-\mathrm{PCA}$ which is accepted by no $d-\mathrm{POCA}$, whatever the dimension d may be (cf. Section 4.2). Curiously enough, we fail to get such a result in the case of picture language recognition. The question whether the inclusion between PictCA with one-way neighborhoods and PictCA with two-way neighborhoods is proper or not, is still open, like in dimension one.

In many aspects, the situation becomes much more complicated for computation on picture languages. Contrary to dimension one where all neighborhoods are real time equivalent to either the one-way neighborhood $\{-1, 0\}$ or the two-way neighborhood $\{-1, 0, 1\}$, the recognition ability of PictCA appears more widely influenced by the choice of the neighborhood. For instance, there exists a picture language recognized in real time with the von Neumann neighborhood which is not recognized in real time with the Moore neighborhood. On the other hand, it is an open question whether all picture languages recognized in real time with the Moore neighborhood are also recognized in real time with the von Neumann neighborhood. More generally, we ignore the precise relationships between the various neighborhoods. And worse, some neighborhoods seem not to admit linear speed up. Actually, the rules which lie behind the communication properties are not simple to grasp. One difficulty is the question of orientation in the two dimensional underlying communication graph. Notably, Szwerinski's property which states that a one dimensional PCA may confuse right and left without time loss and emphasizes that the orientation does not matter in dimension one, seems not to apply to PictCA. And one factor which might discriminate between the various neighborhoods, would be whether each cell of the array somehow knows the direction towards the output cell.

References

1. W. T. Beyer. Recognition of topological invariants by iterative arrays. Technical Report AITR-229, MIT Artificial Intelligence Laboratory, October 1 1969.

2. W. Bucher and K. Čulik II. On real time and linear time cellular automata. *RAIRO Informatique Théorique et Applications/Theoretical Informatics and Applications*, 81:307–325, 1984.
3. T. Buchholz, A. Klein, and M. Kutrib. Iterative arrays with small time bounds. In Mogens Nielsen and Branislav Rovan, editors, *MFCS*, volume 1893 of *Lecture Notes in Computer Science*, pages 243–252. Springer, 2000.
4. T. Buchholz and M. Kutrib. On time computability of functions in one-way cellular automata. *Acta Informatica*, 35(4):329–352, 1998.
5. J. Cervelle and E. Formenti. Algorithmic Complexity and Cellular Automata. In Robert A. Meyers, editor, *Encyclopedia of Complexity and System Science*. Springer-Verlag, 2009.
6. J. H. Chang, O. H. Ibarra, and M. A. Palis. Parallel parsing on a one-way array of finite state machines. *IEEE Transactions on Computers*, C-36(1):64–75, January 1987.
7. J. H. Chang, O. H. Ibarra, and M. A. Palis. Efficient simulations of simple models of parallel computation by time-bounded ATMs and space-bounded TMs. *Theoretical Computer Science*, 68(1):19–36, October 1989.
8. J. H. Chang, O. H. Ibarra, and A. Vergis. On the power of one-way communication. *Journal of the ACM*, 35(3):697–726, July 1988.
9. C. Choffrut and K. Culik II. On real-time cellular automata and trellis automata. *Acta Informatica*, 21(4):393–407, November 1984.
10. S. N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machine. *IEEE Transactions on Computing*, 18:349–365, 1969.
11. K. Culik II. Variations of the firing squad problem and applications. *Information Processing Letters*, 30(3):153–157, February 1989.
12. K. Culik II, J. Gruska, and A. Salomaa. Systolic trellis automata. I. *International Journal Computer Mathematics*, 15(3-4):195–212, 1984.
13. K. Culik II, J. Gruska, and A. Salomaa. Systolic trellis automata. II. *International Journal Computer Mathematics*, 16(1):3–22, 1984.
14. M. Delacourt and V. Poupet. Real time language recognition on 2D cellular automata: Dealing with non-convex neighborhoods. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007*, volume 4708 of *Lecture Notes in Computer Science*, pages 298–309, 2007.
15. M. Delorme and J. Mazoyer. Reconnaisance de langages sur automates cellulaires. Research Report 94-46, LIP, ENS Lyon, France, Dec. 1994.
16. M. Delorme and J. Mazoyer. Reconnaissance parallèle des langages rationnels sur automates cellulaires plans. [Parallel recognition of rational languages on plane cellular automata] Selected papers in honour of Maurice Nivat. *Theoretical Computer Science*, 281(1-2):251–289, June 2002.
17. M. Delorme and J. Mazoyer. Real-time recognition of languages on an two-dimensional Archimedean thread. *Theoretical Computer Science*, 322(2):335–354, August 2004.
18. M. Delorme and J. Mazoyer. Algorithmic tools on Cellular Automata. In *This Book*.
19. C. R. Dyer. One-way bounded cellular automata. *Information and Control*, 44(3):261–281, March 1980.
20. P. C. Fischer. Generation of primes by one-dimensional real-time iterative array. *Journal of the ACM*, 12:388–394, 1965.

21. D. Giammarresi and A. Restivo. Two-Dimensional Languages. in G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, vol. 3, pp. 215-267, Springer-Verlag, 1997.
22. L. M. Goldschlager. A universal interconnection pattern for parallel computers. *Journal of the ACM*, 29(4):1073–1086, 1982.
23. J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. (AMS)*, 117:285–306, 1965.
24. O. H. Ibarra and T. Jiang. On one-way cellular arrays. *SIAM Journal on Computing*, 16(6):1135–1154, December 1987.
25. O. H. Ibarra, M. A. Palis, and S. M. Kim. Some results concerning linear iterative (systolic) arrays. *Journal of Parallel and Distributed Computing*, 2(2):182–218, May 1985.
26. O. H. Ibarra and M. A. Palis. On efficient simulations of systolic arrays of random-access machines. *SIAM Journal on Computing*, 16(2):367–377, April 1987.
27. O. H. Ibarra and T. Jiang. Relating the power of cellular arrays to their closure properties. *Theoretical Computer Science*, 57(2-3):225–238, May 1988.
28. O. H. Ibarra and S. M. Kim. Characterizations and computational complexity of systolic trellis automata. *Theoretical Computer Science*, 29(1-2):123–153, March 1984.
29. O. H. Ibarra, S. M. Kim, and S. Moran. Sequential machine characterizations of trellis and cellular automata and applications. *SIAM Journal on Computing*, 14(2):426–447, 1985.
30. O. H. Ibarra and M. A. Palis. Two-dimensional iterative arrays: characterizations and applications. *Theoretical Computer Science*, 57(1):47–86, April 1988.
31. A. Ito, K. Inoue and I. Takanami. Deterministic two-dimensional on-line tessellation acceptors are equivalent to two-way two-dimensional alternating finite automata through 180°-degree rotation. *Theoretical Computer Science*, 66(3):273–287, 26 August 1989.
32. Y. Kobuchi. A note on symmetrical cellular spaces. *Information Processing Letters*, 25(6):413–415, 26 July 1987.
33. S. R. Kosaraju. Fast parallel processing array algorithms for some graph problems (preliminary version). In ACM, editor, *Conference record of the eleventh annual ACM Symposium on Theory of Computing: papers presented at the Symposium, Atlanta, Georgia, April 30–May 2, 1979*, pages 231–236, New York, NY, USA, 1979. ACM Press.
34. A. Klein and M. Kutrib. Fast one-way cellular automata. *Theoretical Computer Science*, 295(1–3):233–250, February 2003.
35. M. Kutrib. Automata arrays and context-free languages. In *Where mathematics, computer science, linguistics and biology meet*, pages 139–148. Kluwer Acad. Publ., Dordrecht, 2001.
36. S. Levialdi. On shrinking binary picture patterns. *Communications of the ACM*, 15(1):7–10, January 1972.
37. J. Mazoyer and N. Reimen. A linear speed-up theorem for cellular automata. *Theoretical Computer Science*, 101(1):59–98, July 1992.
38. J. Mazoyer and J. B. Yunès. Computations on One Dimensional Cellular Automata. In *This Book*.
39. A. Okhotin. Automaton representation of linear conjunctive languages. In *International Conference on Developments in Language Theory (DLT), LNCS*, volume 6, 2002.

40. N. Ollinger. Computational universality of Cellular Automata In *This Book*.
41. V. Poupet. Cellular automata: Real-time equivalence between one-dimensional neighborhoods. In Volker Diekert and Bruno Durand, editors, *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, volume 3404 of *Lecture Notes in Computer Science*, pages 133–144. Springer, 2005.
42. V. Poupet. A padding technique on cellular automata to transfer inclusions of complexity classes. In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *Second International Symposium on Computer Science in Russia*, volume 4649 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2007.
43. A. L. Rosenberg. Real-time definable languages. *Journal of the ACM*, 14(4):645–662, October 1967.
44. C. Savage. Recognizing majority on a one-way mesh. *Information Processing Letters*, 27(5):221–225, April 1988.
45. A. R. Smith III. Real-time language recognition by one-dimensional cellular automata. *Journal of Computer and System Science*, 6:233–253, 1972.
46. A. R. Smith III. Simple computation-universal cellular spaces. *Journal of the ACM*, 18(3):339–353, July 1971.
47. H. Szwerinski. Symmetrical one-dimensional cellular spaces. *Information and Control*, 67(1–3):163–172, October/November/December 1985.
48. V. Terrier. On real time one-way cellular array. *Theoretical Computer Science*, 141(1–2):331–335, April 1995.
49. V. Terrier. Language not recognizable in real time by one-way cellular automata. *Theoretical Computer Science*, 156(1–2):281–285, March 1996.
50. V. Terrier. Two-dimensional cellular automata recognizer. *Theoretical Computer Science*, 218(2):325–346, May 1999.
51. V. Terrier. Characterization of real time iterative array by alternating device. *Theoretical Computer Science*, 290(3):2075–2084, 2003.
52. V. Terrier. Two-dimensional cellular automata and deterministic on-line tessalation automata. *Theoretical Computer Science*, 301(1–3):167–186, May 2003.
53. V. Terrier. Two-dimensional cellular automata and their neighborhoods. *Theoretical Computer Science*, 312(2–3) 203–222, January 2004.
54. V. Terrier. Closure properties of cellular automata. *Theoretical Computer Science*, 352(1-3):97–107, 2006.
55. V. Terrier. Low complexity classes of multidimensional cellular automata. *Theoretical Computer Science*, 369(1-3):142–156, 2006.

ELSEVIER

Theoretical Computer Science 218 (1999) 325–346

Theoretical Computer Science

Two-dimensional cellular automata recognizer

Véronique Terrier *

GREYC, Université de Caen, Campus 2, 14032 Caen, France

Abstract

We are investigating cellular automata on two-dimensional array as language recognizer. Linear acceleration for Moore and Von Neumann neighborhood is presented. Relationships with one-dimensional CAs and Turing machines are considered. Some limitations of the power capabilities of real-time recognition are shown. © 1999 Published by Elsevier Science B.V. All rights reserved.

1. Introduction

Cellular automata appear to be a relevant model for massively parallel computation. To evaluate their computation capability a lot of interest has been devoted to one-dimensional CAs as language recognizer. In recent years several papers investigate various types of two-dimensional CAs [1, 4, 5, 8]. Here we will investigate some basic properties of two-dimensional CAs as two-dimensional language recognizer with parallel input mode and bounded computation. Indeed in a practical point of view two-dimensional array looks more natural. Clearly on two-dimensional array the behavior of CAs becomes more complex. For instance with regard to the global properties the reversibility and surjectivity problems become indecidable in dimension two [6]. About the language recognition when the input mode is sequential Cole [2] has shown that the power strictly increases with the dimension of the space. But actually to solve problems in higher dimensions often consists in extending methods developed for the dimension one. Here we will generalize techniques used in one-dimensional CAs.

There are several possible definitions of two-dimensional CA recognizers according to the choice of the neighborhood, the different ways to supply the input and to get the result. In this paper we will restrict to Moore and Von Neumann neighborhood, with a parallel input mode where the inputs are supplied in an array rectangular in shape and with the result of the computation is get on a distinguished cell (more often the upper-leftmost cell). The definitions are specified in Section 2. In Section 3 we adapt the

* E-mail: veroniqu@info.unicaen.fr.

0304-3975/99/$ - see front matter
PII: S0304-3975(98)00329-6

linear speed-up methods known for one-dimensional CAs to two-dimensional CAs. In particular for Von Neumann neighborhood this generalization leads to a solution where the grouped cells overlap each other. In Sections 4 and 5 we compare two-dimensional CAs with one-dimensional CAs and Turing machines. The last section focuses on real-time computation and its limitation using counting arguments developed before for one-dimensional CAs. In particular we present a language not real-time recognizable with Moore neighborhood which is real-time recognizable with Von Neumann neighborhood.

2. Definitions

A two-dimensional cellular automaton is a two-dimensional array of identical finite automata (cells) indexed by Z^2 and working synchronously. Each cell evolves according its neighborhood at discrete time step. Formally a two-dimensional CA is a 3-tuple (S,N,δ) where S is the set of states, $N=\{(x_1,y_1),\ldots,(x_k,y_k)\}\subset Z^2$ the neighborhood, $\delta: S^k \to S$ the transition function. Denoting $\langle (u,v),t\rangle$ the state of the cell (u,v) at time t, we have $\langle (u,v),t\rangle = \delta(\langle (u+x_1,v+y_1),t-1\rangle,\ldots,\langle (u+x_k,v+x_k),t-1\rangle)$.

The two classic neighborhoods are considered in this paper: the Moore neighborhood where $N=\{(x,y):\ \|(x,y)\|_\infty \leqslant 1\}$, the Von Neumann neighborhood where $N=\{(x,y):\ \|(x,y)\|_1 = 1\}$.

Here we are interested in CAs as language recognizer. For that purpose we distinguish two subsets of S : Σ and S_{accept}; Σ corresponds to the alphabet of the language and S_{accept} is the subset of accepting states. We specify also a distinguished cell, more often the cell (1,1), which communicates with outside.

In a natural way the two-dimensional languages will be investigated but also to compare two-dimensional CAs with other one-dimensional models the one-dimensional languages will be examined. For two-dimensional languages we need the classic definitions developed for two-dimensional objects. A picture over an alphabet Σ is a $m*n$ array of elements of Σ. The size of the picture is denoted by (m,n). $p(i,j)$, where $1\leqslant i\leqslant m$ and $1\leqslant j\leqslant n$, denotes the element on line i and column j of the picture p. Σ^{**} denotes the set of all pictures over Σ and Σ^{m*n} the set of all pictures of size (m,n).

We will consider only parallel input mode: at initial time 0, for $1\leqslant i\leqslant n$ and $1\leqslant j\leqslant m$, each element $p(i,j)$ of the input picture p is fed to the cell (i,j). We restrict to bounded computation, so the other cells remain in a special state # during all the computation. We say that a CA accepts the picture p in t steps if the distinguished cell enters an accepting state at time t. Let T be a function from N^2 to N. We say that a CA recognizes the language L in time T if it accepts the pictures p of size (m,n) in time $T(m,n)$. As in one-dimensional case we define real time as the minimal time needed by the distinguished cell to read any particular part of the input. Thus when the distinguished cell is the cell (1,1), the real time corresponds to the function $T(m,n)=\max(m,n)-1$ for Moore neighborhood and the function $T(m,n)=m+n-2$ for Von Neumann neighborhood. And for a constant $c>1$, $T(m,n)=c\max(m,n)$ (resp. $c(m+n)$) gives rise to linear time. $\text{CA}_{\text{Moore}}(T(m,n))$ (resp. $\text{CA}_{\text{VN}}(T(m,n))$) will denote the class of

2-dimensional languages recognized in time $T(m,n)$ by a CA with Moore (resp. Von Neumann) neighborhood.

For one-dimensional language recognition by two-dimensional CAs the first ambiguity is in the way the input words are fed into two-dimensional array. For our practical purposes we will consider the words are written in a square in a snakelike order manner. So a word of length n will correspond to a picture of size $(\lceil\sqrt{n}\rceil, \lceil\sqrt{n}\rceil)$ with the letters successively written from left to right and from right to left, and where the last line is completed with $\lceil\sqrt{n}\rceil^2 - n$ blank symbols β.

For example $a_1 \ldots a_{14}$ corresponds to the picture

$$\begin{array}{cccc} a_1 & a_2 & a_3 & a_4 \\ a_8 & a_7 & a_6 & a_5 \\ a_9 & a_{10} & a_{11} & a_{12} \\ \beta & \beta & a_{14} & a_{13} \end{array}$$

Let f be a function from N to N. We say that a CA recognizes the language L in time f if it accepts the words w of length n in time $f(n)$. Thus when the distinguished cell is the cell (1,1), the real time corresponds to the function $f(n) = \lceil\sqrt{n}\rceil - 1$ for Moore neighborhood and the function $f(n) = 2\lceil\sqrt{n}\rceil - 2$ for Von Neumann neighborhood. $\mathrm{CA}_{\mathrm{Moore}}(f(n))$ (resp. $\mathrm{CA}_{\mathrm{VN}}(f(n))$) will denote the class of 1-dimensional languages recognized in time $f(n)$ by a CA with Moore (resp. Von Neumann) neighborhood.

3. Linear acceleration

The linear acceleration known for one-dimensional CAs (cf. [3, 7]) which establishes that a language recognized in time $f(n)$ is also recognized in time $n + \lceil (f(n) - n)/k \rceil$ for some k could be generalized to two-dimensional CAs. An initial phase (of $\max(m,n)$ steps for Moore neighborhood and $m + n - 1$ steps for Von Neumann neighborhood) will consist in compacting the cells by group of $k \times k$ cells. After this phase the CA could work k times faster for Moore neighborhood and $(k+1)/2$ times faster for Von Neumann neighborhood. So a language recognized in time $T(m,n)$ will be also recognized in time $\max(m,n) + \lceil (T(m,n) - \max(m,n))/k \rceil$ with Moore neighborhood and in time $m + n - 1 + 2\lceil (T(m,n) - m - n + 1)/(k+1) \rceil$ with Von Neumann neighborhood.

3.1. The Moore neighborhood case

Let A be a CA which recognizes the language L in time $T(m,n)$. We will build a CA B which recognizes the language L in time $\max(m,n) + \lceil (T(m,n) - \max(m,n))/k \rceil$ for some k. The features of the CA B are as described in [7] for the one-dimensional case: while computing at the same speed as the initial CA A, the CA B performs grouping and synchronization to stop it and launch an automaton which runs k times faster than the initial CA A. The grouping process is performed both on the lines and the columns. The lines grouping process propagates leftwards and is such that at time

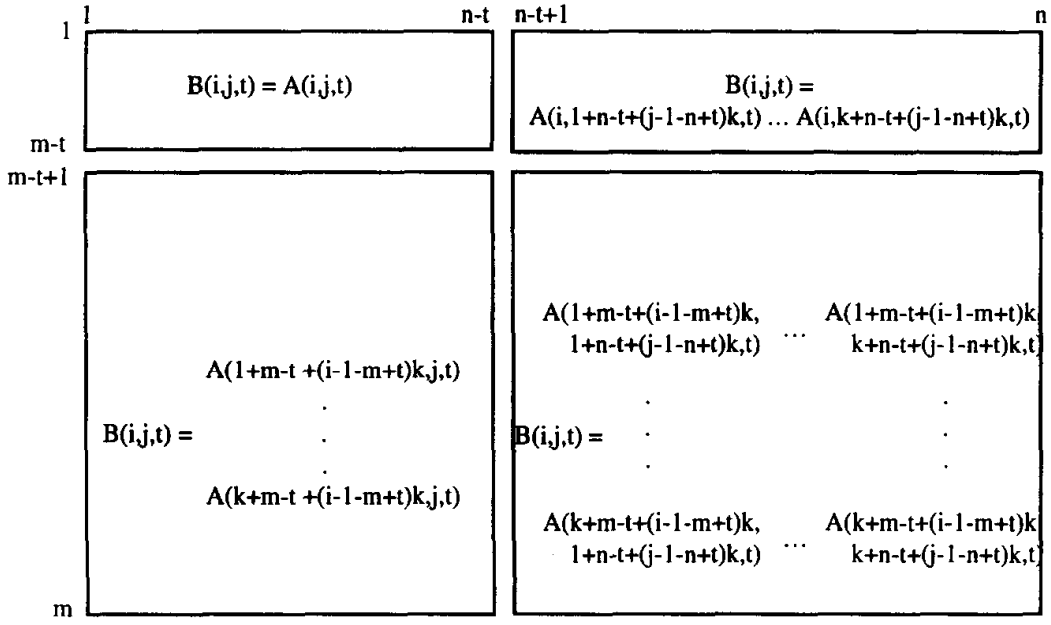


Fig. 1. The grouping process at time t.

t the lines $m-t+1, m-t+2, \ldots, m$ are grouped by k; so to stop it a synchronisation is required at time m. Symmetrically the columns grouping process propagates upwards and must be stopped at time n.

We first describe the synchronization processes. Recall that it exists a one- dimensional CA which synchronizes a segment of length n in $n-1$ steps provided the two extremities be set initially in special states G_{left} and G_{right} (see [7]). So on the automaton B we synchronize each column in order to synchronize all cells at time m and each line in order to synchronize all cells at time n. For that purpose on each cell (i,j) one state is devoted to the synchronization of the line i and a second one to the synchronization of the column j. At time 1 for each cell these two states are set in the special states G_{left} or G_{right} or in a quiescent state according their west, east, north and south neighbors are in the border state #.

We now describe the grouping process. See Fig. 1. We need the following notations. $A(i,j,t)$ will denote the state of the cell (i,j) at time t on the initial CA A (in particular $A(i,j,t)$ is the border state # if $i>m$ or $j>n$); $B(i,j,t)$ will denote the state of the cell (i,j) at time t on the CA B.

$$L_{\text{left}}(i,t) \text{ will represent } \begin{cases} i & \text{if } i \leqslant m-t \text{ and } t \leqslant m, \\ 1+m-t+(i-(1+m-t))k & \text{if } i > m-t \text{ and } t \leqslant m, \\ 1+(i-1)k & \text{if } t > m, \end{cases}$$

$$C_{\text{top}}(j,t) \begin{cases} j & \text{if } j \leqslant n-t \text{ and } t \leqslant n, \\ 1+n-t+(j-(1+n-t))k & \text{if } j > n-t \text{ and } t \leqslant n, \\ 1+(j-1)k & \text{if } t > n, \end{cases}$$

$$L_{\text{right}}(i,t) \begin{cases} L_{\text{left}}(i,t) & \text{if } i \leqslant m-t, \\ L_{\text{left}}(i,t)+k-1 & \text{if } i > m-t, \end{cases}$$

and

$$C_{\text{bottom}}(j,t)\begin{cases} C_{\text{top}}(j,t) & \text{if } j\leqslant n-t,\\ C_{\text{top}}(j,t)+k-1 & \text{if } j>n-t.\end{cases}$$

Proposition 1. *For any CA A there exists a CA B such that if at time* 0 *the two CA A and B are in the same configuration then at time t with* $t\leqslant\max(m,n)$, $B(i,j,t)$ *contains the following array of states*:

$$\begin{array}{ccc} A(L_{\text{left}}(i,t),C_{\text{top}}(j,t),t) & \cdots & A(L_{\text{right}}(i,t),C_{\text{top}}(j,t),t)\\ \vdots & \ddots & \vdots\\ A(L_{\text{left}}(i,t),C_{\text{bottom}}(j,t),t) & \cdots & A(L_{\text{right}}(i,t),C_{\text{bottom}}(j,t),t)\end{array};$$

and from time $t>\max(m,n)$ *the CA B can perform in one step k steps of the CA A.*

Proof. At time 1, $L_{\text{left}}(i,1)$ is i, $L_{\text{right}}(i,1)$ is i for $i<m$ and $i+k-1$ for $i=m$, $C_{\text{top}}(j,1)$ is j, $C_{\text{bottom}}(j,1)$ is j for $j<n$ and $j+k-1$ for $j=m$. Hence we could define a transition function such that the site (m,n) whose east and south neighbors are in state # enters in state $\begin{array}{ccc} A(m,n,1) & \# & \cdots\ \# \\ \# & & \# \\ \vdots & & \vdots \\ \# & \cdots & \# \end{array}$; the rightmost sites (i,n), with $i<m$, whose east neighbors are in state # enters in state $A(i,n,1)\#\ldots\#$; the lowest sites (m,j), with $j<n$, whose north neighbors are in state # enters in state $\begin{array}{c} A(m,j,1) \\ \# \\ \vdots \\ \# \end{array}$; the other sites evolve as on the CA A.

At times $t\leqslant\min(m,n)$, observe that as $L_{\text{left}}(i,t)-1\geqslant L_{\text{left}}(i-1,t-1)$, $L_{\text{right}}(i,t)+1\leqslant L_{\text{right}}(i+1,t-1)$, $C_{\text{top}}(j,t)-1\geqslant C_{\text{top}}(j-1,t-1)$, $C_{\text{bottom}}(j,t)+1\leqslant C_{\text{bottom}}(j+1,t-1)$, we have the required information to compute all the states $A(u,v,t)$ with $L_{\text{right}}(i,t)\leqslant u\leqslant L_{\text{left}}(i,t)$ and $C_{\text{bottom}}(j,t)\leqslant v\leqslant C_{\text{top}}(j,t)$ on the site (i,j) at time t. And as $L_{\text{right}}(i,t)=L_{\text{right}}(i+1,t-1)-1$, $L_{\text{left}}(i,t)=L_{\text{left}}(i+1,t-1)-1$, $C_{\text{bottom}}(j,t)=C_{\text{bottom}}(j+1,t-1)-1$, $C_{\text{top}}(j,t)=C_{\text{top}}(j+1,t-1)-1$, we are able to localize the relevant states $A(u,v,t)$ with $L_{\text{left}}(i,t)\leqslant u\leqslant L_{\text{right}}(i,t)$ and $C_{\text{top}}(j,t)\leqslant v\leqslant C_{\text{bottom}}(j,t)$ among all the states $A(u,v,t)$ with $L_{\text{left}}(i-1,t-1)+1\leqslant u\leqslant L_{\text{right}}(i+1,t-1)-1$ and $C_{\text{top}}(j-1,t-1)+1\leqslant v\leqslant C_{\text{bottom}}(j+1,t-1)-1$ we may compute.

At times t with $\min(m,n)<t\leqslant\max(m,n)$, for symmetry reasons we can suppose without loss of generality that $m\leqslant n$. Observe that as $L_{\text{left}}(i,t)-1\geqslant L_{\text{left}}(i-1,t-1)$, $L_{\text{right}}(i,t)+1\leqslant L_{\text{right}}(i+1,t-1)$, $C_{\text{top}}(j,t)-1\geqslant C_{\text{top}}(j-1,t-1)$, $C_{\text{bottom}}(j,t)+1\leqslant C_{\text{bottom}}(j+1,t-1)$, we have the required information to compute all the states $A(u,v,t)$ with $L_{\text{left}}(i,t)\leqslant u\leqslant L_{\text{right}}(i,t)$ and $C_{\text{top}}(j,t)\leqslant v\leqslant C_{\text{bottom}}(j,t)$ on the site (i,j) at time t. And as $L_{\text{right}}(i,t)=L_{\text{right}}(i,t-1)-1$, $L_{\text{left}}(i,t)=L_{\text{left}}(i,t-1)-1$, $C_{\text{bottom}}(j,t)=$

$C_{\text{bottom}}(j+1,t-1)-1$, $C_{\text{top}}(j,t)=C_{\text{top}}(j+1,t-1)-1$, we are able to determine, provided the synchronization at time m, the new localization of the relevant states.

From the time $t>\max(m,n)$, $L_{\text{left}}(i,t)-k=L_{\text{left}}(i-1,t-1)$, $L_{\text{right}}(i,t)+k=L_{\text{right}}(i+1,t-1)$, $C_{\text{top}}(j,t)-k=C_{\text{top}}(j-1,t-1)$, $C_{\text{bottom}}(j,t)+k=C_{\text{bottom}}(j+1,t-1)$. Thus we have the required information to compute k steps of the CA A in one step on the CA B. And the synchronization at time $\max(m,n)$ allows B to enter a new behavior.

3.2. *The Von Neumann neighborhood case*

See Figs. 2–5. For Von Neumann neighborhood the grouping process is performed by compacting not the lines and the columns but the diagonals. Precisely three grouping processes will interact: from the cell $(m,1)$ toward the diagonal $i=j$, the diagonals $i-j=m-t$ for $t=1,\ldots,m$ will be grouped by k in m steps; from the cell $(1,n)$ toward the diagonal $i=j$, the diagonals $j-i=n-t$ for $t=1,\ldots,n$ will be grouped by k in n steps; from the cell (m,n) toward the diagonal $i+j=2$, the diagonals $i+j=m+n-t+1$ for $t=1,\ldots,m+n-1$ will be grouped by k in $m+n-1$ steps. In order to compute at the same speed than the initial CA A, the grouping CA B will pack together only cells of same parity (cells (i,j) with $i+j$ of same parity). Actually the grouped cells will overlap. We choose also k odd.

To stop these grouping processes, synchronizations will be required at times m,n and $m+n-1$. For that purpose we synchronize each column in order to synchronize all cells at time m, each line in order to synchronize all cells at time n and in addition from time m we synchronize each line in order to synchronize all cells at time $m+n-1$.

We will now focus on the grouping process of the diagonals $i-j=m-t$ for $t=1,\ldots,m$ without dealing with the other grouping processes. This grouping process is initiated from the cell $(m,1)$, propagates to the Northeast reaching at time t the diagonal $i-j=m-t$ and stops at time m on the diagonal $i=j$. At time t the grouping process works only on the diagonals $i-j=\alpha$ with α of same parity than $m-t$ in shifting $k-1$ diagonals of each k-grouped diagonals to the next one of same parity. To describe more precisely the process we need the following notations. We denote by $A(i,j,t)$ the state of the cell (i,j) at time t on the initial CA A; we denote by $B(i,j,t)$ the state of the cell (i,j) at time t on the grouping CA B. $G(i,j,t)$ denotes $A(i,j,t)$ if $i-j<m-t$ else the sequence of the k sites $A(u,v,t)$ such that $u+v=i+j$ and $i-j+2\lfloor\frac{1}{2}[i-j-(m-t)]\rfloor(k-1)\leqslant u-v\leqslant i-j+2\lfloor\frac{1}{2}[i-j-(m-t)]\rfloor(k-1)+2(k-1)$; $H(i,j,t)$ denotes the sequence of the k sites $A(u,v,t)$ such that $u+v=i+j$ and $i-j+2\lfloor\frac{1}{2}[i-j-(m-(t+1))]\rfloor(k-1)\leqslant u-v\leqslant i-j+2\lfloor\frac{1}{2}[i-j-(m-(t+1))]\rfloor(k-1)+2(k-1)$.

Proposition 2. *For any CA A there exists a CA B such that if at time 0 the two CAs A and B are in the same configuration then at times $t<m$, $B(i,j,t)$ contains $A(i,j,t)$ if $i-j<m-t$, $G(i,j,t)$ if $i-j\geqslant m-t$ and $i+j+m+t$ is odd, $G(i,j,t)$, $G(i+1,j,t-1)$, $H(i,j,t-1)$ if $i-j\geqslant m-t$ and $i+j+m+t$ is even and from time $t\geqslant m$, $B(i,j,t)$ contains the sites $A(u,v,t)$ with $u+v=i+j$, $\max(0,(i-j)k-(k-1))\leqslant u-v\leqslant(i-j)k+k-1$.*

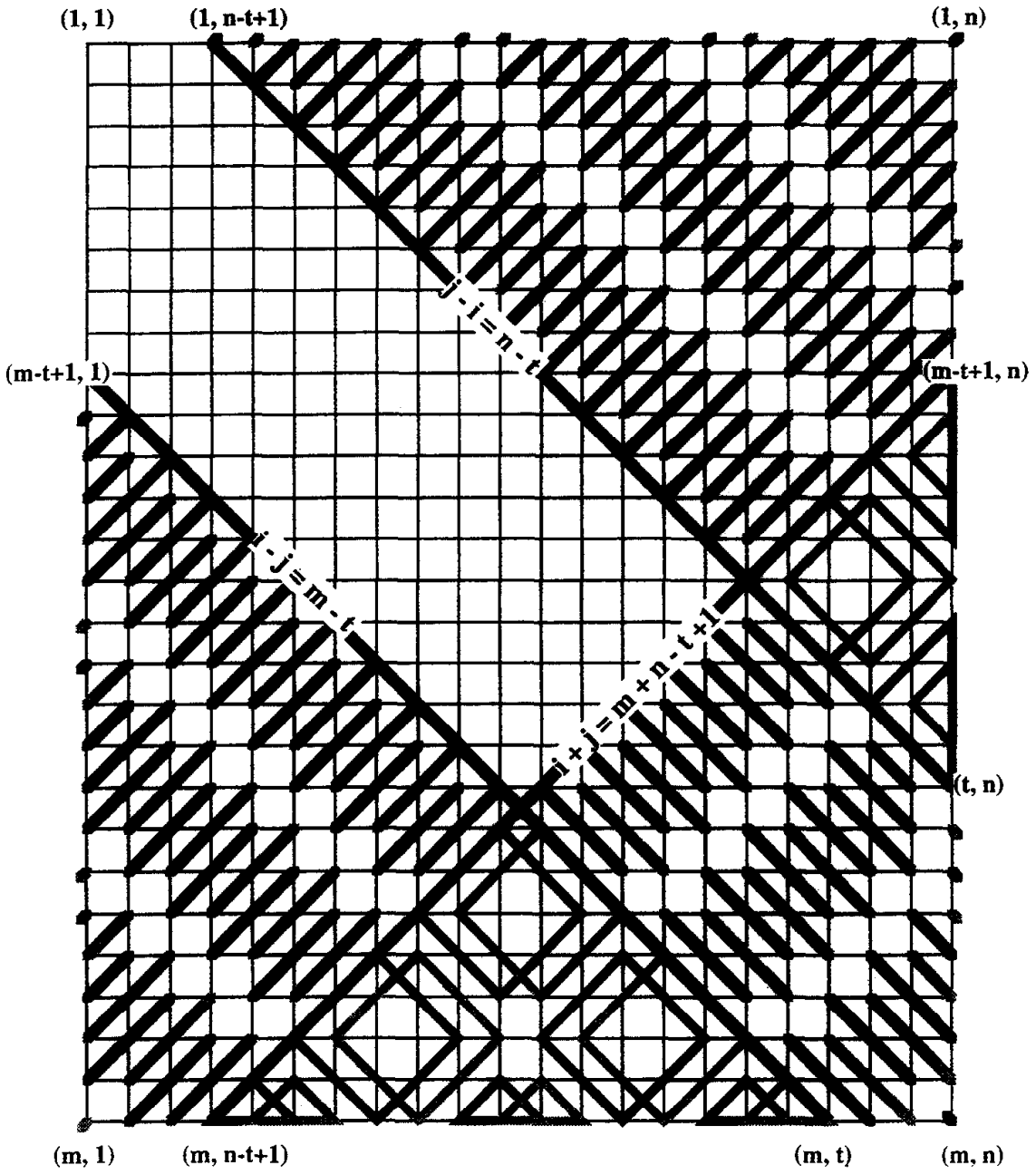


Fig. 2. Time $t < \min(n, m)$.

Proof. At time 1 on the site $(m, 1)$ whose both west and south neighbors are in border state # a wave is initiated which spreads at maximal speed to the north and the east. So this wave proceeds at time t on the cells (i, j) such that $i - j \geqslant m - t$ and is able to discriminate the parity of $i + j + m + t$. Then in the case of $i - j < m - t$ the behavior of the CA B is the same one of the CA A. In the case of $i - j \geqslant m - t + 1$ and $i + j + m + t + 1$ odd, note that $G(i, j, t + 1)$ deals with the same cells as $G(i, j, t)$. And clearly $G(u, v, t)$ with $(u - i, v - j) \leqslant 1$ contain all the required states to compute $G(i, j, t + 1)$. In the case of $i - j \geqslant m - t + 1$ and $i + j + m + t + 1$ even, from $H(i, j - 1, t - 1), G(i + 1, j - 1, t - 1)$ and $H(i + 1, j, t - 1)$ we can compute on the cell (i, j) at time $t + 1$ the sites $A(u, v, t)$ with $u + v = i + j$ and $i - j + 2 + 2\lfloor \frac{1}{2}(i - j - (m - t)) \rfloor (k - 1) \leqslant u - v \leqslant i - j + 2 +$

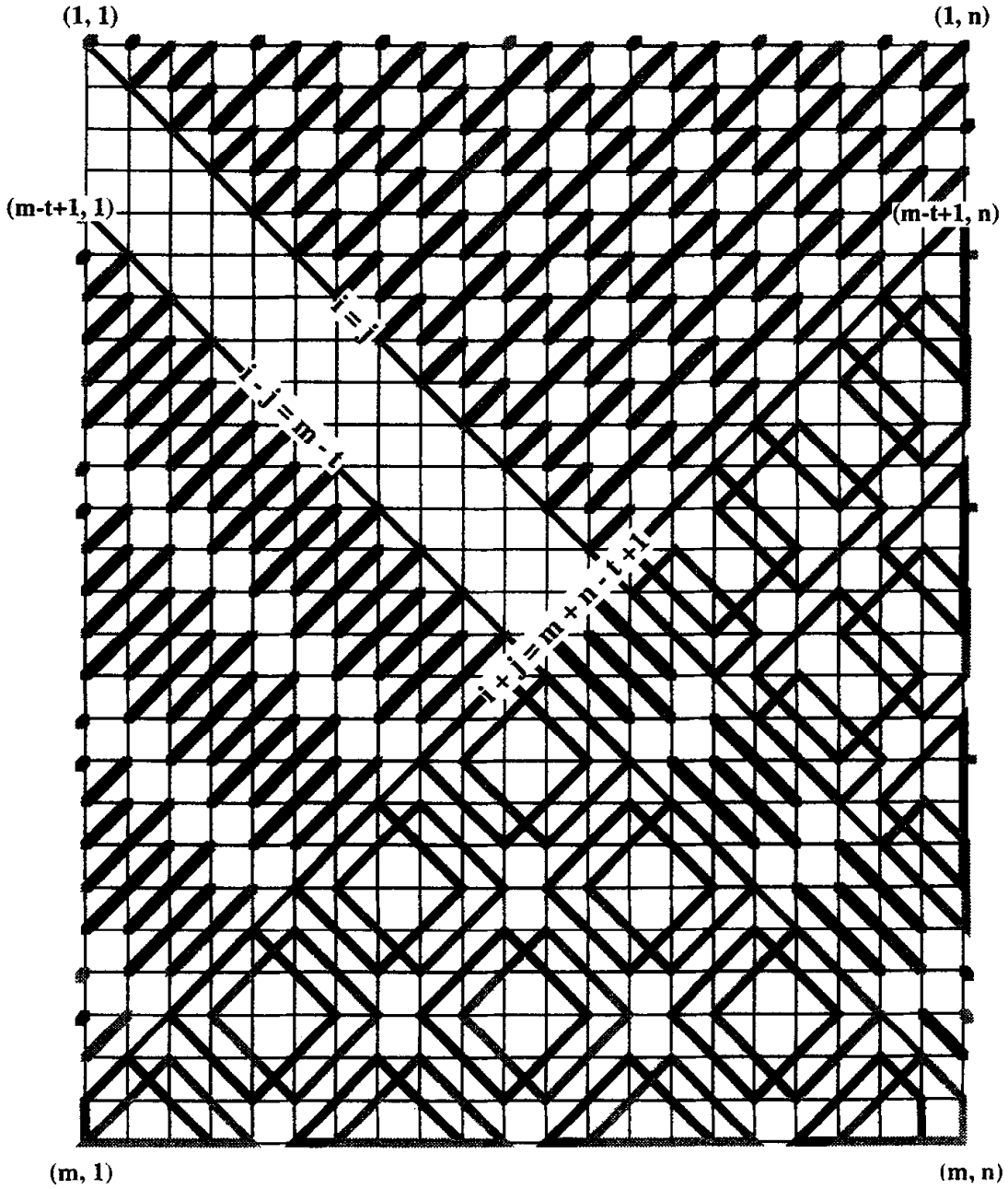


Fig. 3. $\min(n,m) \leqslant \text{time } t < \max(m,n)$.

$2\lfloor\frac{1}{2}(i-j-(m-t))\rfloor(k-1)+2(k-1)$; in addition $G(i,j,t)$ contains the state of the site $A(u,v,t)$ with $u+v=i+j$ and $u-v=i-j+2\lfloor\frac{1}{2}(i-j-(m-t))\rfloor(k-1)$. So we get $H(i,j,t)$. Moreover from $H(i,j,t)$ and $G(u,v,t)$ with $(u-i,v-j)\leqslant 1$ we have all the required states to compute $G(i,j,t+1)$. Besides $G(i+1,j,t-1)$ comes from the south neighbor.

At time m when the synchonization occurs, the process works as before on the cell (i,j) of odd parity. But on the cells (i,j) of even parity which contain k sites of the CA A, only $(k-1)/2$ (instead $k-1$ as before) sites of the CA A are shifted to the even cells $(i-1,j+1)$. Therefore $B(i,j,m)$ contains the sites $A(u,v,m)$ with

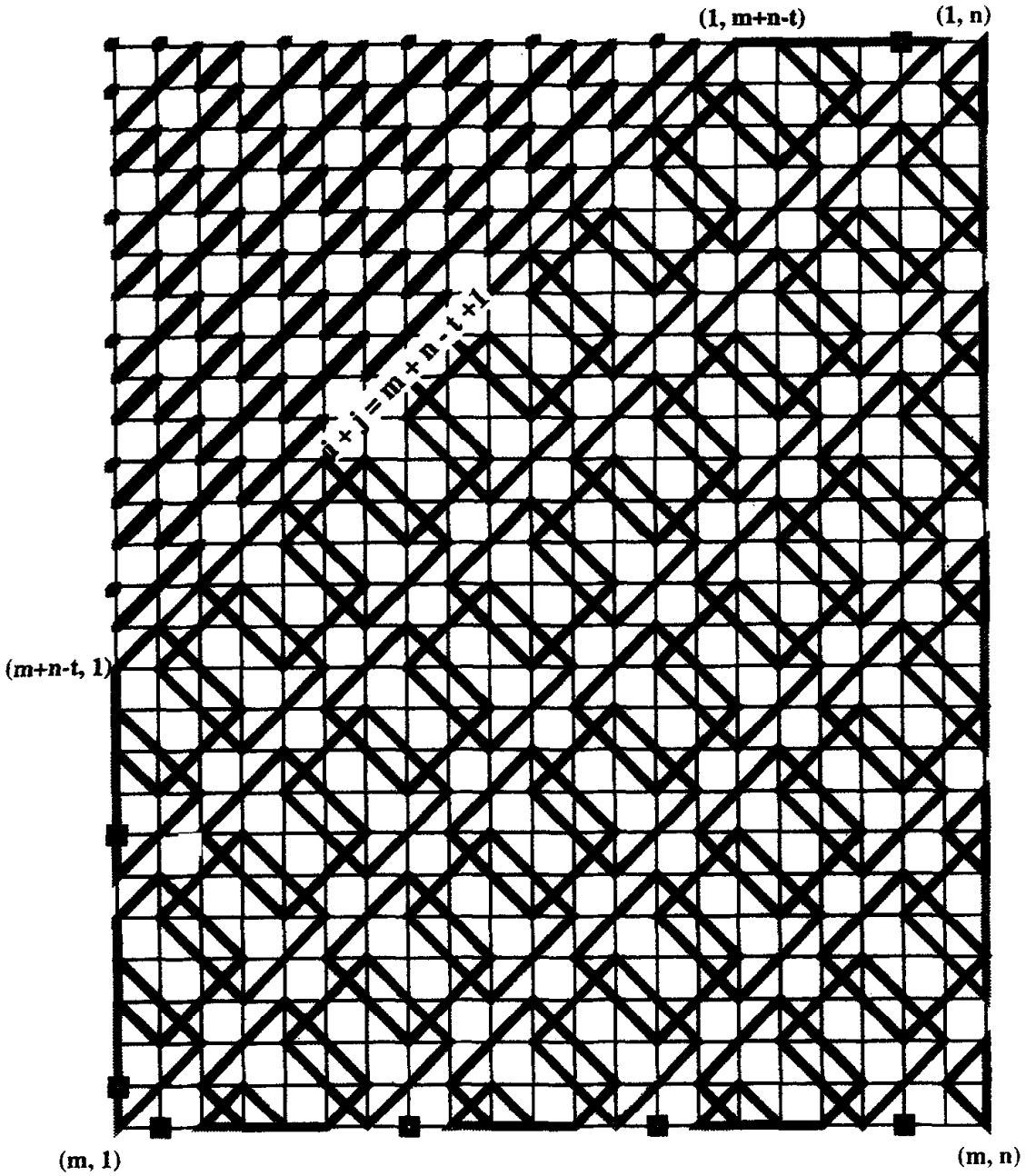


Fig. 4. $\max(m,n) \leqslant \text{time}\, t < m+n-1$.

$u+v=i+j$, $(i-j)k-(k-1) \leqslant u-v \leqslant (i-j)k+k-1$ if $u-v>0$ or $0 \leqslant u-v \leqslant k-1$ if $u=v$. And after the step m the grouping process stops.

Actually the same process is achieved on the diagonals $j-i=n-t$ for $t=1,\ldots,n$ from the cell $(1,n)$ toward the diagonal $i=j$ in n steps and on the diagonals $i+j=m+n-t+1$ for $t=1,\ldots,m+n-1$ from the cell (m,n) toward the diagonal $i+j=2$ (cf. Figs. 2–5). Moreover, a composition (which we do not detail) of the grouping processes of the diagonals $i-j=m-t$ $(t=1,\ldots,m)$ and $i+j=m+n-t+1$ $(t=1,\ldots,m+n-1)$ occurs on the sites $B(i,j,t)$ with $i-j \geqslant m-t$ and $i+j \geqslant m+n-t+1$ at times $t \leqslant m$, and another composition of the grouping processes of the diagonals $j-i=n-t$ $(t=1,\ldots,n)$

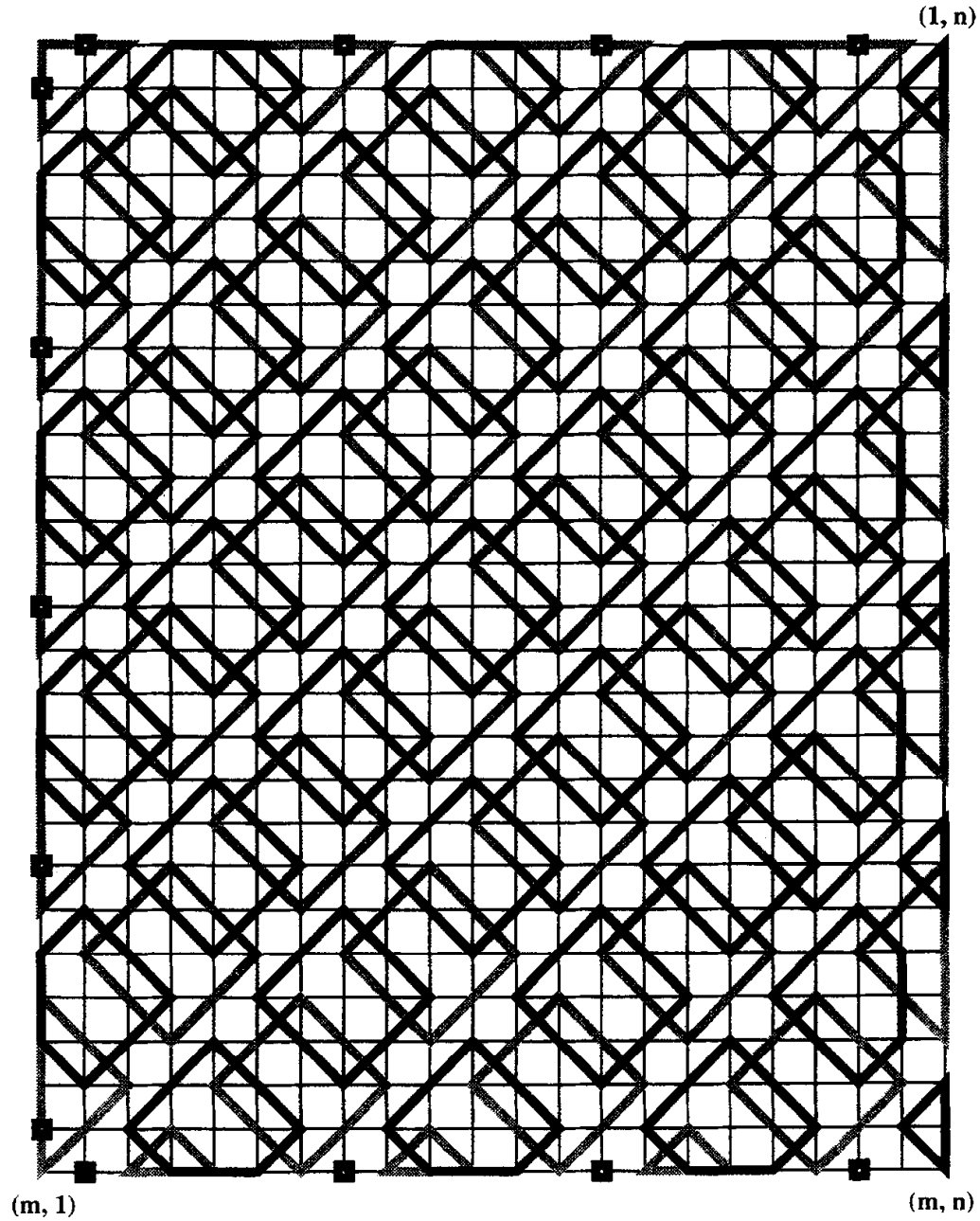


Fig. 5. time $t \geqslant m+n-1$.

and $i+j=m+n-t+1$ $(t=1,\ldots,m+n-1)$ occurs on the sites $B(i,j,t)$ with $j-i \geqslant n-t$ and $i+j \geqslant m+n-t+1$ at times $t \leqslant n$.

Consequently, at time $m+n-1$ $B(i,j,t)$ contains the sites $A(u,v,t)$ such that $u+v$ and $i+j$ have the same parity, $(i-j)k-(k-1) \leqslant u-v \leqslant (i-j)k+k-1$ and $(i+j-2)k-(k-1) \leqslant u+v-2 \leqslant (i+j-2)k+k-1$. In other words $B(i,j,t)$ contains the sequence of k^2 sites $A(1+(i-1)k+\alpha, 1+(j-1)k+\beta, t)$ with $\|(\alpha,\beta)\|_1 \leqslant k-1$ and $\alpha+\beta$ even. Therefore the site $B(i+\alpha, j+\beta, t)$ with $\|(\alpha,\beta)\|_1 \leqslant 2$ contains the sites $A(1+(i-1)k+\alpha, 1+(j-1)k+\beta, t)$ with ($\|(\alpha,\beta)\|_1 \leqslant 3k-1$ and $\alpha+\beta$ even) or ($\|(\alpha,\beta)\|_1 \leqslant 2k-1$ and $\alpha+\beta$ odd) in particular the sites $A(1+(i-1)k+\alpha, 1+(j-1)k+\beta, t)$ with $\|(\alpha,\beta)\|_1 \leqslant 2k$. Hence from this time, $k+1$ steps of the CA A can be

simulated in 2 steps on the CA B. Actually for any integer $s, sk+1$ steps of the CA A can be simulated in $s+1$ steps on the CA B.

Corollary 1. *If* $T(m,n) \geqslant c(m+n)$ *for any* $c>1$, *then* $\mathrm{CA}_{\mathrm{VN}}(T(m,n)) = \mathrm{CA}_{\mathrm{Moore}}(T(m,n))$.

Proof. $\mathrm{CA}_{\mathrm{VN}}(T(m,n) \subset \mathrm{CA}_{\mathrm{Moore}}(T(m,n))$ indeed the Von Neumann neighborhood is contained in the Moore neighborhood.

$\mathrm{CA}_{\mathrm{Moore}}(T(m,n)) \subset \mathrm{CA}_{\mathrm{VN}}(2T(m,n))$ since one step on a Moore CA can be simulated in two steps on a Von Neumann CA. Let k be an odd integer such that $k \geqslant 4c/(c-1)$, it exists since $c>1$.

We have

$$\begin{aligned}\mathrm{CA}_{\mathrm{VN}}(2T(m,n)) &\subset CA_{\mathrm{VN}}(m+n+2)\lceil(2T(m,n)-m-n)/(k+1)\rceil) \\ &\quad \text{according to Section 3.2} \\ &\subset \mathrm{CA}_{\mathrm{VN}}(T(m,n)/c + 4T(m,n)/k) \quad \text{as } T(m,n) \geqslant c(m+n) \\ &\subset \mathrm{CA}_{\mathrm{VN}}(T(m,n)) \quad \text{by choice of } k.\end{aligned}$$

4. Comparison with one-dimensional CAs

Note that the way the input words are supplied into two-dimensional array will be determining in the comparison of one- and two-dimensional CAs. Recall that here we consider the words are written in a square in a snakelike order manner. The following propositions depend closely on this choice.

4.1. Simulation of a one-dimensional CA by a two-dimensional CA

Proposition 3. *If the language L is recognized by a one-dimensional CA in time* $f(n)$ *then L is recognized by a two-dimensional CA with Von Neumann neighborhood in time* $\lceil\sqrt{n}\rceil + f(n)$.

Proof. As the input words are written in a snakelike order, to identify the relevant neighborhood of each site of the square is simple. For that purpose each site will be marked with a state indicating its respective left and right neighbors (NW for the sites with the left neighbor on the north and the right neighbor on the west, and in the same way EW, ES, WS, WE, NE) (Fig. 6).

This marking process propagates downwards one line by step in the following way.

At time 1, the site $(1,1)$ whose north and east neighbors are in state #, enters in state NW; the site $(1,\lceil\sqrt{n}\rceil)$ whose north and west neighbors are in state #, enters in state ES; the sites $(1,i)$ with $1<i<\lceil\sqrt{n}\rceil$ whose north neighbor but not their east or their west neighbors are in state #, enter in state EW.

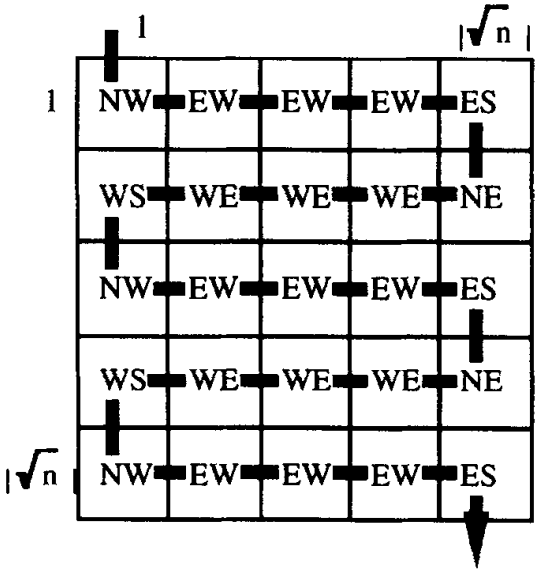


Fig. 6.

At time $i>1$, the sites not marked enter in state WS, WE, NE, NW, EW, ES, respectively if their north neighbor is in state NW, EW, ES, WS, WE, NE, respectively.

Therefore at time $\lceil\sqrt{n}\rceil$ all sites are marked.

Parallely at time 1 a synchronization process starts from both ends of each line. So all cells are synchronized at time $\lceil\sqrt{n}\rceil$. Then from this time the two-dimensional CA operates as in one-dimensional case.

Corollary 2. *If the language L is recognized by a one-dimensional CA in time $f(n)$ then L is recognized by a two-dimensional CA with Von Neumann neighborhood in time $f(n)/k$.*

Proof. Indeed in one-dimensional case the minimal time is n, therefore of greater order than $\lceil\sqrt{n}\rceil$. And according to Section 3.2, we have the desired linear acceleration.

Corollary 3. *If the language L is recognized by a one-dimensional CA in time $f(n)$ then L is recognized by a two-dimensional CA with Moore neighborhood in time $f(n)/k$.*

4.2. Simulation of a two-dimensional CA by a one-dimensional CA

The following proposition is a special case of a result in [1].

Proposition 4. *If the language L is recognized by a two-dimensional CA A (with Moore or Von Neumann neighborhood) in time $f(n)$ then L is recognized by a one-dimensional CA B in time $O(\sqrt{n}f(n))$.*

Proof. On the two-dimensional CA A an input word of length n is supplied in a snakelike order as a picture of size $(\lceil\sqrt{n}\rceil,\lceil\sqrt{n}\rceil)$. So the initial phase of the simulation consists in dividing the working area of the one-dimensional CA B in $\lceil\sqrt{n}\rceil$ segments

of length $\lceil\sqrt{n}\rceil$ so that the ith segment of B corresponds to the ith line of A if i is odd, the reverse of the ith line of A if i is even. For that purpose we distinguish the leftmost cells of each segments i.e. the cells $1+i\lceil\sqrt{n}\rceil$ with $i=0,\ldots,\lceil\sqrt{n}\rceil-1$. Precisely we construct a CA which distinguishes the sites $(1+i\lceil\sqrt{n}\rceil,(i-1)^2+i\lceil\sqrt{n}\rceil)$. The construction is displayed by the Figs. 7 and 8. In parallel a signal end of line is sent from the leftmost cell n at time 0, it meets the signal P on the section corresponding to the diagonal $t=c-1+(\lceil\sqrt{n}\rceil-1)^2$. Therefore among the distinguished sites $(1+i\lceil\sqrt{n}\rceil,(i-1)^2+i\lceil\sqrt{n}\rceil)$ we can select the sites on the diagonal $t=c-1+(\lceil\sqrt{n}\rceil-1)^2$ i.e. the sites $(1+i\lceil\sqrt{n}\rceil,(\lceil\sqrt{n}\rceil-1)^2+i\lceil\sqrt{n}\rceil)$. From these sites vertical signals are initiated, they mark the cells $1+i\lceil\sqrt{n}\rceil$ with $i=0,\ldots,\lceil\sqrt{n}\rceil-1$. During this initial phase each input bit remains on its cell and a synchronization process is initiated on the cell 1 at time 0, in this way the cells are synchronized at time $2n-1$. From this time the simulation of the step k of the CA A will be done at time $2n-1+k(\lceil\sqrt{n}\rceil+1)$ on the CA B in this following way (cf. Fig. 8):

(1) at time $2n-1+k(\lceil\sqrt{n}\rceil+1)$ providing the step k of the CA A is computed the new states are propagated at maximal speed to the left and to the right;

(2) at the next step $2n+k(\lceil\sqrt{n}\rceil+1)$ in order to reverse each segment the digits are sent at maximal speed to the right, bounce on the rightmost cell of the segment and return at maximal speed to the left, in parallel a synchronization process is initiated from both ends of each segment;

(3) in this way at time $2n+k(\lceil\sqrt{n}\rceil+1)+\lceil\sqrt{n}\rceil-1$ the synchronization occurs. Moreover, the ith cell of the jth segment receives the ith states of the $(j-1)$th and $(j+1)$th segments and the $(\lceil\sqrt{n}\rceil-i)$th state of the jth segment which correspond to the states of the cells $(i,j-1),(i,j),(i,j+1)$ (resp. $(\lceil\sqrt{n}\rceil-i,j-1),(\lceil\sqrt{n}\rceil-i,j),(\lceil\sqrt{n}\rceil-i,j+1)$) of the CA A if $j+k$ is even (resp. odd).

Thus at the next step $2n-1+(k+1)(\lceil\sqrt{n}\rceil+1)$ we know all the required neighborhood to compute the new states corresponding to the step $k+1$ of the CA A.

In fact the computation of the last $n-(\lceil\sqrt{n}\rceil-1)^2$ cells are performed on the segment $\lceil\sqrt{n}\rceil-1$.

5. Comparison with Turing machine

Here as two-dimensional CAs are restricted to bounded computation, two-dimensional CAs will have the same power than Turing machines bounded in space n. Now we describe direct simulations.

5.1. Simulation of a Turing machine working in space n by a two-dimensional CA

Proposition 5. *If L is recognized by a Turing machine which works in time $f(n)$ and space n then L is recognized in time $\lceil\sqrt{n}\rceil+f(n)$ by a two-dimensional CA.*

Proof. The construction of the two-dimensional CA is based on the usual techniques [9] where the simulated heads cells remain fixed on the site $(1,1)$ and the simulated tapes are shifted. In order that the simulation is performed in real time, the simulated

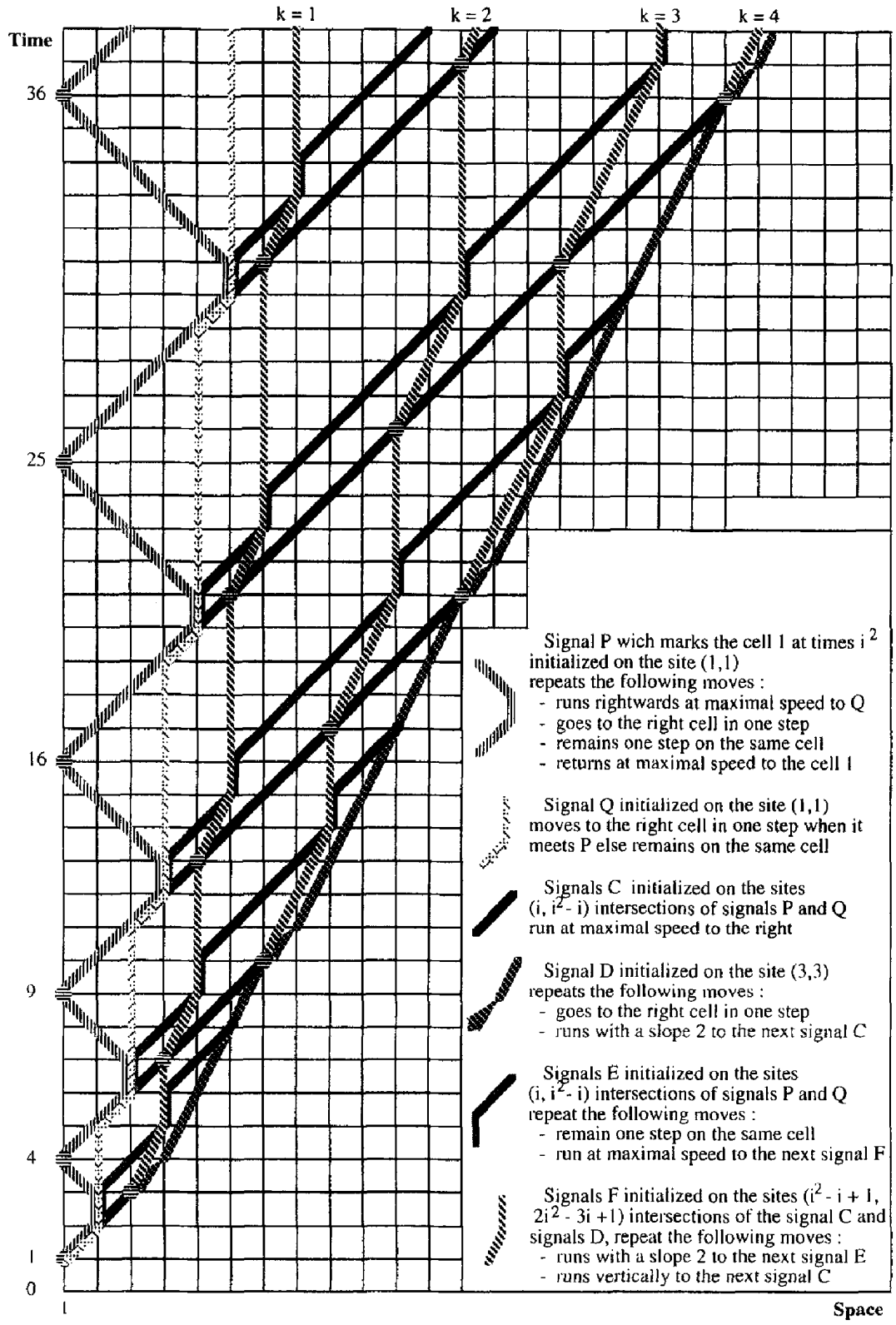


Fig. 7. Construction of the sites $(1 + ik, (i-1)^2 + ik)$ intersections of the signals C and F.

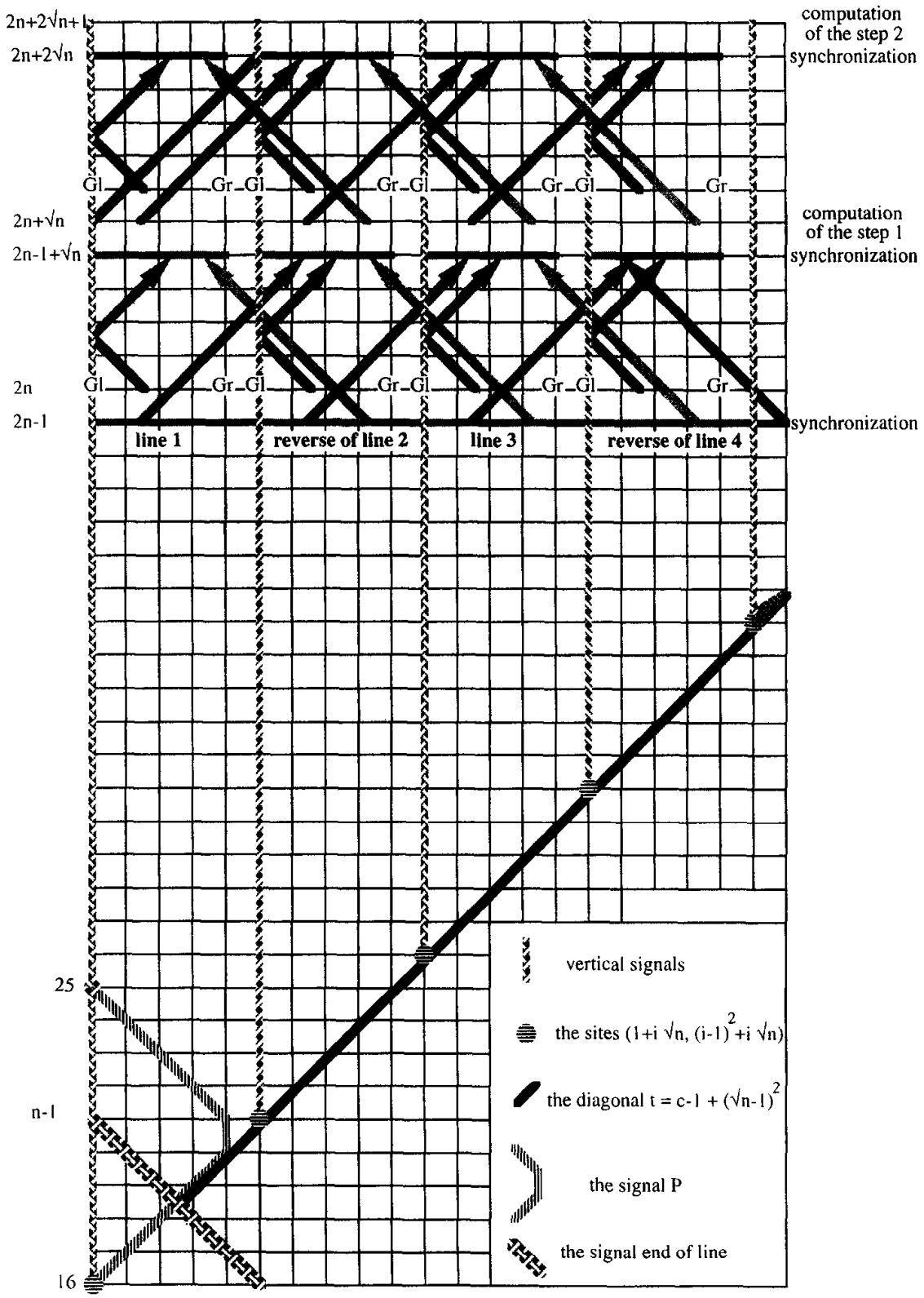

2n+2√n+1
2n+2√n
computation of the step 2
synchronization
Gl
Gr_Gl
Gr_Gl
Gr_Gl
Gr
2n+√n
2n-1+√n
computation of the step 1
synchronization
2n
2n-1
Gl
Gr_Gl
Gr_Gl
Gr_Gl
Gr
synchronization
line 1
reverse of line 2
line 3
reverse of line 4
25
n-1
16
vertical signals
the sites (1+i √n, (i-1)^2+i √n)
the diagonal t = c-1 + (√n-1)^2
the signal P
the signal end of line

Fig. 8.

squares must be grouped. So an initial phase of $\lceil\sqrt{n}\rceil$ steps is required to group the columns by two to the left and also to exhibit, as in Section 4.1, the trace of the simulated tapes. As the Turing machine works in space n, this trace is contained in the bounded area.

Proposition 5 can also be obtained as a consequence of Proposition 3.

5.2. Simulation of a two-dimensional CA by a Turing machine

Proposition 6. *Let L be a one-dimensional language recognized in time $f(n)$ by a two-dimensional CA then L is recognized in time $O(nf(n))$ and space n by a Turing machine with two tapes.*

Proof. We construct a Turing machine with two tapes, the second tape with two tracks. On the CA an input word of length n is supplied in a snakelike order as a picture of size $(\lceil\sqrt{n}\rceil, \lceil\sqrt{n}\rceil)$. So the initial phase of the simulation consists in dividing the input word of length n into $\lceil\sqrt{n}\rceil$ blocks of length $\lceil\sqrt{n}\rceil$ so that the ith block corresponds to the ith line if i is odd, the reverse of the ith line if i is even. A first scan of the input permits to compute $\lceil\sqrt{n}\rceil$ on the second tape. A second scan of the input is required to mark each $i.\ \lceil\sqrt{n}\rceil$th letters (for $i=1,\ldots,\lceil\sqrt{n}\rceil$) of the input by $*$.
So the initial phase takes $O(n)$ steps.

Then the simulation of one step of the CA goes as follows. See Fig. 9. On the first tape the head scans the input one square by step. On the second tape the head scans $\lceil\sqrt{n}\rceil$ squares successively from left to right and from right to left. During the left-to-right sweeps it records on the first track the odd blocks (red on the first tape) and during the right-to-left sweeps it records on the second track the even blocks (red on the first tape).

By this way at the end of the scan of the ith block on the first tape, we have the ith line on the upper (resp. lower) track, the $(i-1)$th line on the lower (resp. upper) track and the head points to the rightmost (resp. leftmost) square if i is even (resp. odd). So simultaneously the heads will point to the $(j+1)$th elements of the $(i-1)$th, ith, $(i+1)$th lines; and if in the state the contents of the squares scanned during the two last steps (the $(j-1)$th and the jth elements of the $(i-1)$th, ith, $(i+1)$th lines) are recorded we know all the relevant information to compute the jth element of the ith line at the next step of the CA.

Thus simulating one step of the CA requires $n+\lceil\sqrt{n}\rceil+1$ steps on the TM and the lines are shifted $\lceil\sqrt{n}\rceil+1$ squares rightward. Now it is the odd line which are written in reverse. So the simulation of the next step is made in the same way but in reverse.

6. Real-time recognition

In this section we will exploit methods, developed before in one-dimensional case [2, 10] which used equivalence classes and counting arguments to show that some languages are not real time recognizable.

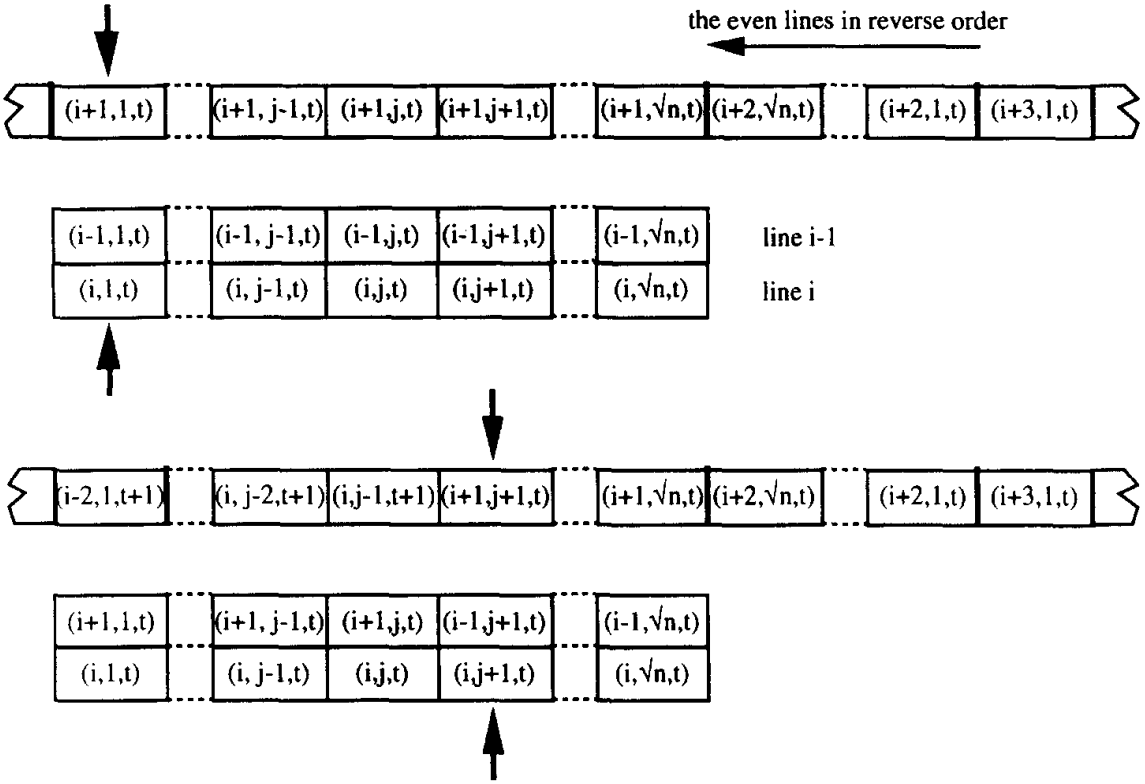


Fig. 9. Simulation of the line i.

6.1. *The Moore neighborhood case*

A characteristic of 2-CAs in real time with Moore neighborhood is that on picture of size (n,n) the input element of the cell (n,n) has only one way (the diagonal) to reach the distinguished cell(1,1) and then this element interacts with only $O(n)$ elements among the n^2 elements of the input array. This feature allow us to express a relation of equivalence. Then we will present a language which will be shown by counting arguments not to be a real time language for CAs with Moore neighborhood. Finally we will define a CA with Von Neumann neighborhood able to recognize it in real time.

Notation. See Fig. 10. Let M be a 2-CA with Moore neighborhood with (1,1) as distinguished cell. We consider the behavior of the CA working in real time on input pictures of size (m,n). We will use the following notations. For any set $U \subset Z^2$ we denote by $\mathrm{Neig}(U)$ the Moore neighborhood of U and $\mathrm{Neig}^t(U)$ its tth iterate. Let A be the set $\{(x,y): 1 \leqslant x \leqslant m, 1 \leqslant y \leqslant n\}$ and $E_t = \mathrm{Neig}^{\max(m,n)-1-t}(\{(1,1)\}) \cap A$ denote the set of the cells at step t which can have an effect on the result of the computation, i.e. which can affect the cell (1,1) at time $\max(m,n)-1$.

Let $R_t(U) = \mathrm{Neig}^t(U) \cap E_t$ represent the set of cells which contain at time t the relevant information regarding the input bits supplied at initial time in U.

A portion of picture p on a set U consists in the elements $p(i,j)$ such that $(i,j) \in U$.

We denote by $p \oplus c$ the picture resulting of the concatenation of a portion of picture p on A/U and a portion of picture c on U.

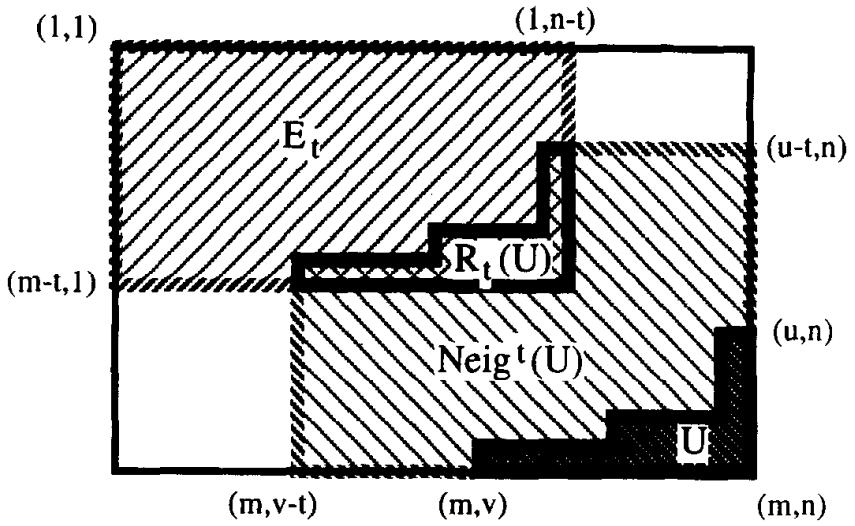


Fig. 10.

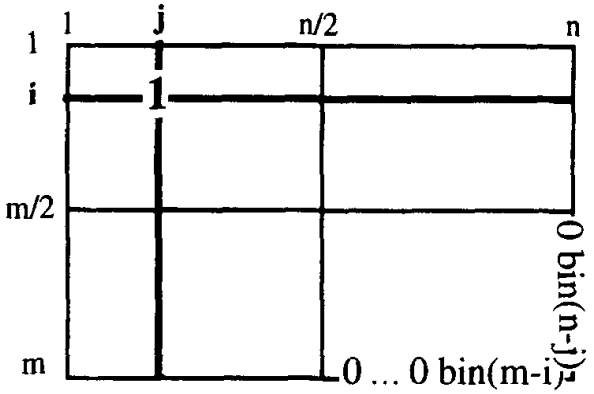


Fig. 11. The language L.

The relation of equivalence. Note that by definition of $R_t(U)$ the states of $\mathrm{Neig}(R_{t+1}(U))/R_t(U)$ at time $t=0,\ldots,\max(m,n)-1$ are independent of the input bits supplied in U at time 0. So we could define the following relation of equivalence: two portions of picture p, p' on A/U are *U-equivalent* if the two states sequences of the cells $\mathrm{Neig}(R_{t+1}(U))/R_t(U)$ at time $t=0,\ldots,\max(m,n)-2$ during the evolution of the CA on the inputs p and p' are equal.

Fact 1. *If p and p' are U-equivalent then for all portions of picture c on U, $p \oplus c$ is accepted by the CA if and only if $p' \oplus c$ is accepted by the CA.*

Proof. As the states of R_t at time t are function of the states of $\mathrm{Neig}(R_t(U))/R_{t-1}(U)$ and the states of $R_{t-1}(U)$ at time $t-1$, if p and p' are U-equivalent then for all portions of picture c on U the sequences of states of $R_t(U)$ at time $t=1,\ldots,\max(m,n)-1$ generated by the CA on the inputs $p \oplus c$ and $p' \oplus c$ are equal. In particular, as $R_{\max(m,n)-1}(U)=\{(1,1)\}$ if p and p' are U-equivalent then for all portions of picture c on U we have $p \oplus c$ is accepted by the CA if and only if $p' \oplus c$ is accepted by the CA.

Let $L=\{p\in\{0,1\}^{**}$: p is of size (m,n) with $m\geqslant 2$ $(\log(n)+2)$ and $n\geqslant 2$ $(\log(m)+2)$ and it exists i,j such that $1\leqslant i\leqslant m/2, 1\leqslant j\leqslant n/2, p(i,j)=1$, $p(m,\lfloor\frac{n}{2}\rfloor+1)=\cdots=p(m,n-\lfloor\log(m-i)\rfloor-2)=0, p(m,n-\lfloor\log(m-i)\rfloor-1)\cdots p(m,n-1)$ is the binary notation of $m-i$, $p(\lfloor\frac{m}{2}\rfloor+1,n)=\cdots=p(m-\lfloor\log(n-j)\rfloor-2, n)=0, p(m-\lfloor\log(n-j)\rfloor-1,n)\cdots p(m-1,n)$ is the binary notation of $n-j\}$. See Fig. 11.

Proposition 7. *L is not recognizable in real time by 2-CAs with Moore neighborhood.*

Proof. We consider pictures A of size (n,n) and the set $U=\{(x,n): x=n-\lfloor\log(n-1)\rfloor-1,\ldots,n-1\}\cup\{(n,y): y=n-\lfloor\log(n-1)\rfloor-1,\ldots,n-1\}$. Note that the cardinality of $\mathrm{Neig}(R_t(U))/R_{t-1}(U)$ is lower than $4\lfloor\log(n-1)\rfloor+8$. Thus the cardinality of $\bigcup_{t=0}^{n-2}\mathrm{Neig}(R_t(U))/R_{t-1}(U)$ is lower than $(n-1)(4\lfloor\log(n-1)\rfloor+8)$. So if L is recognized by a Moore CA in real time whose the number of states is s then there is at most $s^{(n-1)(4\log(n-1)+8)}=2^{O(n\log(n))}U$ -equivalence classes for the portions of picture on A/U.

At each subset E of $\{(x,y): 1\leqslant x,y\leqslant n/2\}$ we associate the portion of picture p_E on A/U defined by $p_E(x,y)=0$ if $x>n/2$ or $y>n/2$ and for $1\leqslant x,y\leqslant n/2$ $p_E(x,y)=1$ if $(x,y)\in E$, else $p_E(x,y)=0$. Observe that for all distinct subsets E,F of $\{(x,y): 1\leqslant x, y\leqslant n/2\}$ there exists integers x,y such that (x,y) belongs to E/F or F/E. Now consider the portion of picture c on U such that $c(n,n-\lfloor\log(n-1)\rfloor-1)=\cdots=c(n,n-\lfloor\log(n-i)\rfloor-2)=0$, $c(n,n-\lfloor\log(n-i)\rfloor-1)\ldots c(n,n-1)$ is the binary notation of $n-i$, $c(n-\lfloor\log(n-1)\rfloor-1,n)=\cdots=c(n-\lfloor\log(n-j)\rfloor-2,n)=0$, $c(n-\lfloor\log(n-j)\rfloor-1,n)\ldots c(n-1,n)$ is the binary notation of $n-j$. Thus $p_E\oplus c$ belongs to L if and only if $p_F\oplus c$ does not belong to L. In other words p_E and p_F are not equivalent. To conclude note that the number of subsets of $\{(x,y): 1\leqslant x,y\leqslant n/2\}$ is $2^{n^2/4}$. Hence it is of order greater than $2^{O(n\log(n))}$.

Proposition 8. *L is recognizable in real time by a 2-CA with Von Neumann neighborhood.*

Proof. See Fig. 12. From $p(m,\lfloor\frac{n}{2}\rfloor+1),\ldots,p(m,n-1)=0\ldots0$ $\mathrm{bin}(m-i)$, if $1\leqslant i\leqslant m/2$ we have to select the line i. For that purpose we compute $\mathrm{bin}(m-k)$ on the right of the kth line for all $k=m-1,\ldots,1$ in the usual way. We consider that initially all cells are in a quiescent state β. The process begins on the cell $(m-1,n-1)$ which can be distinguished at time 2 and enters in state 1. Then the CA evolves according this rules:

$$\begin{matrix} & n & & \\ w & c & e & \to c= \\ & s & & \end{matrix}\begin{cases} 1 & \text{if } s=e=1 \text{ or } (e=0' \text{ and } s=0,\ \beta), \\ 0 & \text{if } e=1 \text{ and } s=0,\ 0', \\ 0' & \text{if } s=1 \text{ and } e=0',\ \beta. \end{cases}$$

So the tth bit of $\mathrm{bin}(m-k)$ is computed on the cell $(k,n-t)$ at time $m-k+t$. Simultaneously from the step 1 the lowest line is sent upward, so the element $p(m,t)$

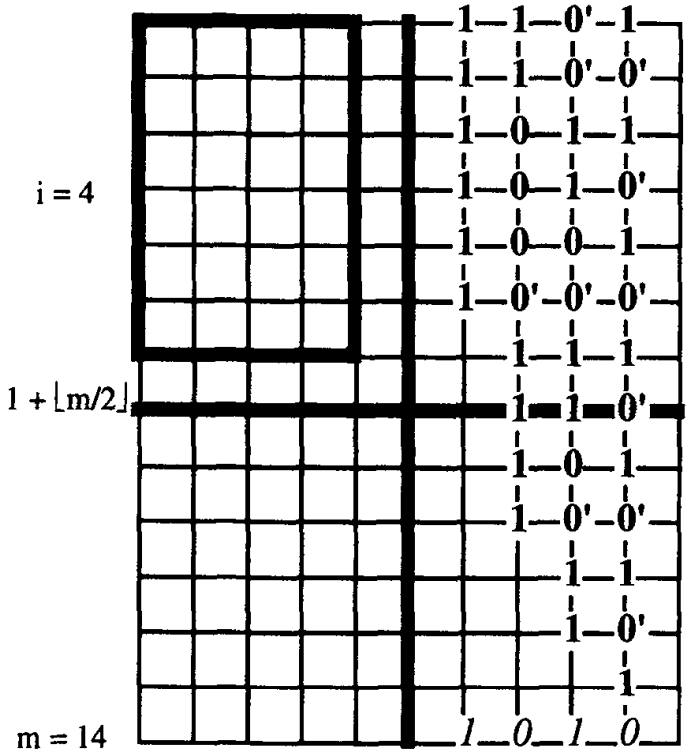


Fig. 12.

is known by the cell (k,t) at time $m-k+1$. In addition we can distinguish the line $1+\lfloor m/2\rfloor$ and the column $1+\lfloor n/2\rfloor$ by sending signals from both ends of each line and each column. Thus a comparison process can start on each line k with $k \leqslant n/2$ at time $m-k+1$ from the cell $(k,n-1)$ and propagates to the right until reaching the column $1+\lfloor n/2\rfloor$. The comparison succeeds on the line u. And from the site $(i,1+\lfloor n/2\rfloor)$ a signal of selection is sent to the right. Inverting the roles of the lines and the columns the same process is achieved in order to select the column j. In this way the cell (i,j) is selected at time $m-i+n-j$. To sent its content to the cell $(1,1)$ requires $i+j-2$ steps, hence this content reaches the cell $(1,1)$ at time $m+n-2$.

6.2. *The Von Neumann case*

Here we will restrict to a particular case where the distinguished cell is the cell $(m,1)$ for pictures of size $(2m-1,n)$. We consider the behavior of a 2-CA with Von Neumann neighborhood working in real time on input pictures of size $(2m-1,n)$. We need the following notations.

Let t be an integer less than n. We denote by A the set $\{(x,y)\colon 1 \leqslant x \leqslant 2m-1, 1 \leqslant y \leqslant n\}$, by U the set of t cells $\{(1,y)\colon n-t<y \leqslant n\}$, by V the set of t cells $\{(2m-1,y)\colon n-t<y \leqslant n\}$. We focus on the step $m-2$. $E_{m-2}=\mathrm{Neig}^n(\{(m,1)\})=\{(x,y)\colon 1 \leqslant y, x-y \geqslant m-n-1, x+y \leqslant m+n+1\}$ denotes the set of cells at step $m-2$ which can have an effect on the result of the computation (i.e. the cell $(m,1)$ at time $m+n-2$).

$\text{Neig}^{m-2}(U)=\{(x,y)\colon 1\leqslant x\leqslant m-1, 1\leqslant y\leqslant n, x-y\leqslant m-n+t-2\}$ (resp. $\text{Neig}^{m-2}(V)=\{(x,y)\colon m+1\leqslant x\leqslant 2m-1, 1\leqslant y\leqslant n, x+y\geqslant m+n-t+2\}$) represents the set of cells at time $m-2$ depending on the input bits supplied at time 0 in U (resp. V).

Note that at time $m-2$, $E_{m-2}/\text{Neig}^{m-2}(U\cup V)$ is independent of the input bits supplied at initial time in U and V, $\text{Neig}^{m-2}(U)$ is independent of V and $\text{Neig}^{m-2}(V)$ of U. So we could define the following relation of equivalence: two portions of picture p, p' on $A/(U\cup V)$ are (U,V)-*equivalent* if

(i) the states at time $m-2$ of $E_{m-2}/\text{Neig}^{m-2}(U\cup V)$ of the CA on the inputs p and p' are equal;

(ii) for all portions of picture u on U, the states at time $m-2$ of $E_{m-2}\cap\text{Neig}^{m-2}(U)$ of the CA on the inputs $p\oplus u$ and $p'\oplus u$ are equal;

(iii) for all portions of picture v on V, the states at time $m-2$ of $E_{m-2}\cap\text{Neig}^{m-2}(V)$ of the CA on the inputs $p\oplus v$ and $p'\oplus v$ are equal.

Fact 2. *The number of (U,V)-equivalence classes of portions of pictures on $A/(U\cup V)$ is bounded by $2^{O(n2^t)}$.*

Proof. Let s be the number of states of the CA. (i) As the cardinality of $E_{m-2}/\text{Neig}^{m-2}(U\cup V)$ is $n+(n-t)(n-t+1)$ there is at most $s^{n+(n-t)(n-t+1)}$ distinguishable pictures on $E_{m-2}/\text{Neig}^{m-2}(U\cup V)$. (ii) Note that $\text{Neig}^{m-2}(U)\cap E_{m-2}$ is composed first for $i=1,\ldots,t$ of the $(n-t+i/2)$ sites $\{(y+m-n+t-1,y)\colon 1\leqslant y\leqslant n-t+(i+1)/2)$ which depends on the bits supplied at initial time on the i cells $(1,n-t+1),\ldots,(1,n-t+i)$ but not on the $t-i$ cells $(1,n-t+i+1),\ldots,(1,n)$, second for $2\leqslant i\leqslant j\leqslant t$ and $i+j$ even of the sites $(m-1+(j-i)/2, n-t+(i+j)/2)$ which depend exactly on the bits supplied at initial time on the $j-i+1$ cells $(1,n-t+i),\ldots,(1,n-t+j)$. So the number of distinguishable pictures on $\text{Neig}^{m-2}(U)\cap E_{m-2}$ at time $m-2$ is at most $\prod_{1\leqslant i\leqslant t} s^{(n-t+\lfloor i/2\rfloor)2^i}\cdot\prod_{\substack{2\leqslant i\leqslant j\leqslant t\\ i+j \text{ even}}} s^{1+j-i}=2^{c(n2^t+t^3)}$. (iii) In the same way the number of distinguishable pictures on $\text{Neig}^{m-2}(U)\cap E_{m-2}$ at time $m-2$ is at most $2^{c(n2^t+t^3)}$. Thus the number of (U,V)-equivalence classes is at most $2^{c(n+(n-t)(n-t+1)}\cdot 2^{c(n2^t+t^3)}\cdot 2^{c(n2^t+t^3)}$ i.e. $2^{O(n2^t)}$.

We present now a language not real-time recognizable.

Let $a=\lceil\log n\rceil$, $b=\lceil\log 2m\rceil$ and $t=\lceil(a+b)/2\rceil$.

Let $L=\{p\in\{0,1\}^{**}\colon p$ is of size $(2m-1,n)$ and there exist i,j
such that $1\leqslant i\leqslant 2m-1, 1\leqslant j\leqslant n$,
$p(i,j)=1, p(1,n-t+1)\ldots p(1,n-t+a)$ is the binary notation of j,
$p(1,n-t+a+1)\ldots p(1,n)p(2m-1,n-t+1)\ldots p(2m-1,n)$
is the binary notation of $i\}$.

Proposition 9. *L is not recognizable in real time by a 2-CA with Von Neumann neighborhood whose cell $(m,1)$ is the distinguished cell.*

Proof. At each subset E of $\{(x,y)\colon 1<x<2m-1, 1\leqslant y\leqslant n\}$ we associate the portion of picture p_E on $A/(U\cup V)$ defined by $p_E(x,y)=1$ if $(x,y)\in E$ else $p_E(x,y)=0$. As $a=\lceil\log n\rceil$ and $b=\lceil\log 2m\rceil$ all lines and columns could be encoded. Hence for all distinct subsets E and F of $\{(x,y)\colon 1<x<2m-1, 1\leqslant y\leqslant n\}$, p_E and p_F are not (U,V)-equivalent. Note that as $t=(a+b)/2$ the number of (U,V)-equivalence classes for portions of pictures on $A/(U\cup V)$ is bounded by $2^{O(n2^{(a+b)/2})}$. In other part, the number of subsets of $\{(x,y)\colon 1\leqslant x\leqslant n, 1<y<2m-1\}$ is $2^{n(2m-3)}$. Hence, provided $m=O(2^b)$ is of greater order than $n=O(2^a)$, $2^{n(2m-3)}$ is of order greater than $2^{O(n2^{(a+b)/2})}$.

References

[1] A.C. Achilles, M. Kutrib, T. Worsch, On relations between arrays of processing elements of different dimensionality, Parcella 96, Akademie Verlag, Berlin, 1996, pp. 13–20.
[2] S.N. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machine, IEEE Trans. Comput. C-18 (1969) 349–365.
[3] O.H. Ibarra, S.M. Kim, S. Moran, Sequential machine characterization of trellis and cellular automata and application, SIAM J. Comput. 14 (1985) 426–447.
[4] O.H. Ibarra, M.A. Palis, Two-dimensional iterative arrays: characterizations and applications, Theoret. Comput. Sci. 56 (1988) 47–86.
[5] K. Inoue, A. Nakamura, Some properties of two-dimensional on-line tessalation acceptors, Inform. Sci. 13 (1977) 95–121.
[6] J. Kari, Reversibility of 2D cellular automata is undecidable, Physica D 45 (1990) 379–385.
[7] J. Mazoyer, N. Reimen, A linear speed-up theorem for cellular automata, Theoret. Comput. Sci. 101 (1992) 59–98.
[8] Zs. Roka, Simulations between cellular automata on Cayley graphs, Theoret. Comput. Sci. (1999), to appear.
[9] A.R. Smith, Real-time language recognition by one-dimensional cellular automata, J. ACM 6 (1972) 233–253.
[10] V. Terrier, Language not recognizable in real time by one-way celular automata, Theoret. Comput. Sci. 156 (1996) 281–287.

Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 369 (2006) 142–156
Theoretical Computer Science
www.elsevier.com/locate/tcs

Low complexity classes of multidimensional cellular automata

Véronique Terrier

GREYC, Campus II, Université de Caen, 14032 Caen, France

Received 9 February 2006; received in revised form 29 June 2006; accepted 16 July 2006

Communicated by Bruno Durand

Abstract

In this paper, multidimensional cellular automaton are considered. We investigate the hierarchy designed by the low complexity classes when the dimensionality is increased. Whether this hierarchy is strict, is an open problem. However, we compare different variants and study their closure properties. We present also a correspondence between a main variant of multidimensional real-time cellular automata and one-way multihead alternating finite automata.

Keywords: Multidimensional cellular automata; Real-time and linear time classes; Closure properties; One-way multihead alternating finite automata

1. Introduction

Cellular automaton (CA) is a major model of massively parallel computation. It consists of a regular array of uniformly interconnected identical finite automata which work synchronously. Famous examples [6,8], where the information is distributed and synchronized in a very efficient way, illustrate the ability of CA to do fast computation. On the other hand, the study of the limitations of these machines is not simple. Notably, for one-dimensional CA bounded in linear space, the basic question whether unrestricted time is more powerful than minimal time, remains unanswered.

By the way, remind that one-dimensional linear space bounded CA working in unrestricted time has the same computational ability as linear space bounded Turing machine. Here, we will look at the dimensionality of the array as resource. In this area, Cole has considered k-dimensional iterative arrays (DIAs) (a main restricted variant of CA with sequential input mode); he has proved that, for real-time (i.e. minimal time) computation, the computing capability increases as the dimensionality of the iterative arrays increases [5]. Other variants of k-dimensional CA have been investigated [1–3,9,13]. Here, we will focus on their lower complexity classes, since they can be simulated in linear space on Turing machine, as proved in [9]. Moreover, refined simulations have been exhibited. In [2], Chang et al. have shown that multidimensional CA working in linear time can be simulated by linear time bounded alternating Turing machine. Furthermore, it has been set that linear time bounded alternating Turing machine can be simulated by one-dimensional one-way CA [3]. And obviously, one-dimensional linear space bounded CA is at least as powerful as one-dimensional one-way CA. But whether these models have different computational ability, is unknown. In other words, to use as resource the dimensionality instead of the time, yields another hierarchy between minimal time and unrestricted time of one-dimensional linear space bounded CA. But, except for k-dimensional real-time iterative

E-mail address: veroniqu@info.unicaen.fr.

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.07.061

arrays which do not have the ability to simulate one-dimensional real-time CA, there is no evidence that for low complexity classes, increasing the dimensionality of the CA allows to increase the computation ability. Here, we will investigate more finely the low complexity classes of multidimensional CA. Among the numerous variants which can be defined according to the choice of the neighborhood, we will restrict to a few ones which seems representative to us.

The paper is organized as follow. Section 2 recalls the definitions and presents different variants of multidimensional CA. Section 3 deals with speed up results. Section 4 is devoted to comparisons between multidimensional CA types. In Section 5, closure properties are presented. In Section 6, the equivalence between one-way multihead alternating finite automata and one of the multidimensional real-time CA types is exhibited.

2. Definitions

A k-dimensional CA is a k-dimensional array of identical finite automata (the cells) indexed by $\mathbb{Z}^k$. In vector notation each cell is identified to its vector position $\mathbf{c} = (c_1, \ldots, c_k)$. The communication links are finite and uniform for every cell. They are specified by a finite subset of $\mathbb{Z}^k$ called the neighborhood. Each cell takes on a value from a finite set, the set of states. Cells evolve synchronously at discrete time steps according to their neighborhood and to a transition function. Formally, a k-dimensional CA is a triple $(S, \mathcal{V}, \delta)$ where S is a finite set of states, the neighborhood $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$ is a finite subset of $\mathbb{Z}^k$ and $\delta : S^{|\mathcal{V}|} \to S$ is the transition function. A site $(\mathbf{c}, t)$ denotes the cell $\mathbf{c}$ at time t and $\langle \mathbf{c}, t\rangle$ denotes its state. So at each step $t \geqslant 0$, the state is updated in this way: $\langle \mathbf{c}, t+1\rangle = \delta(\langle \mathbf{c}+\mathbf{v}_1, t\rangle, \ldots, \langle \mathbf{c}+\mathbf{v}_r, t\rangle : \mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_r\})$.

To specify language recognition, we distinguish two subsets of S: the alphabet Σ of the language and the subset S_{acc} of accepting states. We also distinguish a special state of S, designated by the quiescent state $\sharp$. Here, we assume that the computation is bounded in space: when the input size is n, the cells not in $\{\mathbf{c} : 0 \leqslant c_1, \ldots, c_k < n\}$ remain in the quiescent state $\sharp$ during all the computation. Furthermore, for $t < 0$, all cells in $\{\mathbf{c} : 0 \leqslant c_1, \ldots, c_k < n\}$ are in the quiescent state $\sharp$. Each of these cells remains in the quiescent state until the step it may be affected by the input.

We distinguish two modes to give the input, the parallel one and the sequential one. In parallel input mode, at initial time 0, the ith input symbol of the input $w = x_0 \ldots x_{n-1}$ of size n, is fed to the cell $(i, 0, \ldots, 0)$: $\langle i, 0, \ldots, 0, 0\rangle = x_i$. In sequential input mode, the input is given sequentially to a specific cell, the input cell indexed by $\mathbf{0} = (0, \ldots, 0)$. The symbol x_i of the input $w = x_0 \ldots x_{n-1}$ is gotten by the input cell $\mathbf{0}$ at time i; at time $t \geqslant n$, the cell $\mathbf{0}$ gets an end-marker \$. Further, in sequential input mode, the input cell $\mathbf{0}$ evolves according to a particular transition function $\delta^{\text{init}} : (\Sigma \cup \{\$\}) \times S^{|\mathcal{V}|} \to S$, so $\langle \mathbf{0}, t\rangle = \delta^{\text{init}}(x_t, \langle \mathbf{v}_1, t-1\rangle, \ldots, \langle \mathbf{v}_r, t-1\rangle : \mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_r\})$ (with $x_t = \$$ when $t \geqslant n$).

The output cell, which determines the acceptance, depends on the neighborhood. In case of two-way communication, as with the Von Neumann neighborhood $\{\mathbf{d} \in \mathbb{Z}^k : \Sigma|d_i| \leqslant 1\}$ or with the Moore neighborhood $\{\mathbf{d} \in \mathbb{Z}^k : \max|d_i| \leqslant 1\}$, one chooses the cell $\mathbf{0}$ as the output cell. In case of one-way communication, as with the one-way Von Neumann neighborhood $\{-\mathbf{d} : \mathbf{d} \in \mathbb{N}^k \text{ and } \Sigma d_i \leqslant 1\}$ or with the one-way Moore neighborhood $\{-\mathbf{d} : \mathbf{d} \in \mathbb{N}^k \text{ and } \max d_i \leqslant 1\}$, one chooses the cell $\mathbf{n} - \mathbf{1} = (n-1, \ldots, n-1)$ as the output cell.

A CA accepts a word w if, on input w, the output cell enters an accepting state at some time t. Let f be a function from $\mathbb{N}$ to $\mathbb{N}$. A CA recognizes a language L in time f, if it accepts exactly the words $w \in L$ of length $n = |w|$ at time $f(n)$. Among these time complexities, the real-time function $rt(n)$ corresponds to the minimal time for the output cell to read the whole input of size n. The linear time function corresponds to $f(n) = \tau\, rt(n)$, where τ is a constant strictly greater than 1.

We will denote the usual one-dimensional CA with parallel input mode by 1-PCA and 1-PCA($f(n)$) will refer to the class of languages recognized in time $f(n)$ by 1-PCA. In the sequel, we will focus on CA with one-way Von Neumann neighborhood. k-SOCA (resp. k-POCA) will stand for k-dimensional CA with sequential (resp. parallel) input mode and one-way Von Neumann neighborhood. See Fig. 1 which depicts these devices in case of dimension 2. Various denominations are used in different papers: here, we adopt the notations of [7]; 2-SOCA is named one way two-dimensional iterative array (2-DIA) in [1] and OIA in [13]; k-POCA is named k-dimensional one-way mesh connected array (k-OWMA) in [2].

Further, k-RSOCA (resp. k-LSOCA) will denote the class of languages recognized in real-time (resp. linear time) by k-SOCA. For parallel input mode, the corresponding classes are designated by k-RPOCA and k-LPOCA.

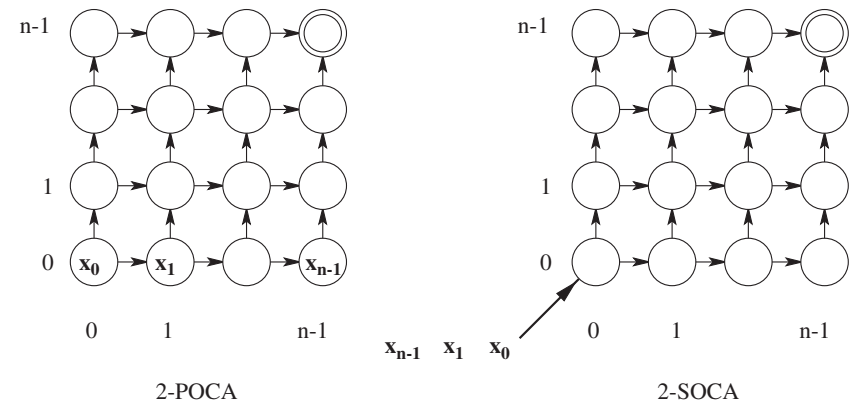


Fig. 1. Two-dimensional OCA.

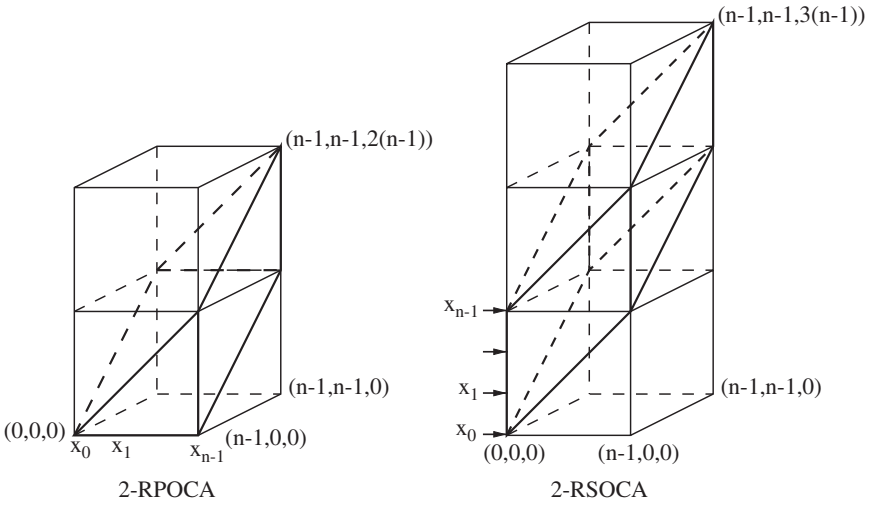


Fig. 2. Real-time one-way devices.

Note that the result of the computation is gotten, for k-RSOCA, on the output site $(\mathbf{n}-\mathbf{1}, (k+1)(n-1))$ and for k-RPOCA, on the output site $(\mathbf{n}-\mathbf{1}, k(n-1))$. As it will be verified in the next section, k-OCA admits linear acceleration. So, without loss of generality, in the sequel, we will assume that the result of the computation is gotten, for k-LSOCA, on the output site $(\mathbf{n}-\mathbf{1}, (k+2)(n-1))$ and for k-LPOCA, on the output site $(\mathbf{n}-\mathbf{1}, (k+1)(n-1))$ (Fig. 2).

To get some hints on the properties of these classes, we consider the working area composed of the relevant sites regarding the computation, that means the sites which both are influenced by the input and can have an effect on the output site. Note that, the set of sites affected by the input is $\{(\mathbf{c}, t) : t \geqslant c_2 + \cdots + c_k\}$ when the input mode is parallel and $\{(\mathbf{c}, t) : t \geqslant c_1 + c_2 + \cdots + c_k\}$ when the input mode is sequential. On the other hand the set of sites which can influence an output site $(\mathbf{n}-\mathbf{1}, \tau(n-1))$ is $\{(\mathbf{c}, t) : t \leqslant c_1 + \cdots + c_k + (\tau - k)(n-1)\}$. Hence, on an input of size n, the working area is:

- $\{(\mathbf{c}, t) : 0 \leqslant c_1, \ldots, c_k < n$ and $c_2 + \cdots + c_k \leqslant t \leqslant c_1 + \cdots + c_k\}$ for a k-RPOCA;
- $\{(\mathbf{c}, t) : 0 \leqslant c_1, \ldots, c_k < n$ and $c_2 + \cdots + c_k \leqslant t < c_1 + \cdots + c_k + n\}$ for k-LPOCA;
- $\{(\mathbf{c}, t) : 0 \leqslant c_1, \ldots, c_k < n$ and $c_1 + \cdots + c_k \leqslant t < c_1 + \cdots + c_k + n\}$ for k-RSOCA;
- $\{(\mathbf{c}, t) : 0 \leqslant c_1, \ldots, c_k < n$ and $c_1 + \cdots + c_k \leqslant t < c_1 + \cdots + c_k + 2n - 1\}$ for k-LSOCA.

The shape of these working areas can be partly described by the following characteristics: the maximal time h, the number nb_{all} of all the sites, the number nb_{beg} of the sites that depend on the first symbol x_0 (i.e. on the site $(\mathbf{0}, 0)$), the number nb_{end} of the sites that depend on the last symbol x_{n-1} (i.e. on the site $(0, \ldots, n-1, 0)$ in case of parallel input mode and on the site $(\mathbf{0}, n-1)$ in case of sequential input mode), and the number nb_{int} of the sites that depend

both on x_0 and x_{n-1}. The following table gives the asymptotic behavior of these characteristics for the classes that we will examine.

	h	nb_{all}	nb_{beg}	nb_{end}	nb_{int}
k-RPOCA	$\Theta(n)$	$\Theta(n^{k+1})$	$\Theta(n^k)$	$\Theta(n^k)$	$\Theta(n^{k-1})$
k-RSOCA	$\Theta(n)$	$\Theta(n^{k+1})$	$\Theta(n^{k+1})$	$\Theta(n^k)$	$\Theta(n^k)$
k-LPOCA	$\Theta(n)$	$\Theta(n^{k+1})$	$\Theta(n^{k+1})$	$\Theta(n^k)$	$\Theta(n^k)$
k-LSOCA	$\Theta(n)$	$\Theta(n^{k+1})$	$\Theta(n^{k+1})$	$\Theta(n^{k+1})$	$\Theta(n^{k+1})$

In the following, we will verify that, among these classes, one class $\mathcal{C}_1$ simulates another one $\mathcal{C}_2$ if we can embed the working areas of CA of type $\mathcal{C}_2$ into the working areas of CA of type $\mathcal{C}_1$. That will be possible when the order of each characteristic of $\mathcal{C}_1$ is greater or equal to the corresponding one of $\mathcal{C}_2$. So the relations k-RSOCA = k-LPOCA, k-RSOCA $\subseteq$ (k + 1)-RPOCA and (k − 1)-LPOCA $\subseteq$ k-RSOCA will be set. Further, the computation turns out to be symmetrical for classes whose nb_{beg} and nb_{end} are of same order. So we will prove that k-RPOCA and k-LSOCA are closed under reverse. Also, for 1-RPOCA, remark that nb_{int} is constant, besides some limits on the computational ability of 1-RPOCA have been established using counting arguments [17]. By the way, note that, in the case of the k-dimensional iterative arrays investigated by Cole [5], nb_{end} is also constant for all k.

To end this section, let us precise that the reverse w^R of a word w is the word w written backwards. For a language L, its reverse is $L^R = \{w^R : w \in L\}$ and for a class of languages $\mathcal{C}$, its reverse is $\mathcal{C}^R = \{L^R : L \in \mathcal{C}\}$.

3. Linear acceleration

In this section, we consider linear acceleration of the running time up to real-time. Observe that, in case of one-way communication, any two cells are not mutually interdependent. So the sites of the working area and their dependencies form a directed graph which is acyclic. This characteristic allows to speed up computation easily. Speed up results have already been proved for dimension 1 [12]. Fig. 3 depicts the scheme for dimension 1: once the cell gets the whole input part situated on its left, it can operate R times faster for any integer constant R. The principle can be generalized to any dimension and any neighborhood with one-way communication. Let us prove it for the one-way Von Neumann neighborhood, first when the input mode is sequential, second when the input mode is parallel.

Proposition 1. *Let $f, rt : \mathbb{N} \to \mathbb{N}$ be two functions where $rt(n) = (k+1)(n-1)$ is the real-time complexity for* k-SOCA. *If a language L is recognized by a* k-SOCA *in time $rt(n) + f(n)$ then, for any positive integer R, L is recognized by a* k-SOCA *in time $rt(n) + \lceil f(n)/R \rceil$.*

Proof. First remark that for any k-SOCA, the working area $W = \{(\mathbf{c}, t) : 0 \leqslant c_1, \ldots, c_k < n, t \geqslant 0\}$ associated with inputs of size n, can be divided into two regions $X = \{(\mathbf{c}, t) \in W : \sum c_i + n - 1 \geqslant t\}$ and $Y = \{(\mathbf{c}, t) \in W : \sum c_i + n - 1 < t\}$. For that, a wave is initiated on the initial cell $\mathbf{0}$, at time n, when $\mathbf{0}$ receives a first end-marker \$. This wave spreads at maximal speed and so characterizes the set of sites $F = \{(\mathbf{c}, t) \in W : t = \sum c_i + n\}$. Actually, this wave marks the first time where cells know they have got the whole input. This wave corresponds to the lower layer of Y and so allows to divide the working area into X and Y.

Now consider any k-SOCA $\mathcal{A}$ which recognizes the language L in time $rt(n) + f(n)$. Let us define the k-SOCA $\mathcal{B}$ which simulates $\mathcal{A}$. On X, $\mathcal{B}$ behaves like $\mathcal{A}$. On Y, the computation of $\mathcal{A}$ is mapped on $\mathcal{B}$ according to the transformation $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ defined by

$$g(\mathbf{c}, t) = \left(\mathbf{c}, \left\lceil \frac{(R-1)(n-1+\sum c_i) + t}{R} \right\rceil\right).$$

Let us verify the validity of the transformation. First, as the upper layer of X consists in sites $(\mathbf{c}, n-1+\sum c_i)$ and $g(\mathbf{c}, n-1+\sum c_i) = (\mathbf{c}, n-1+\sum c_i)$, the transition between X and Y is guaranteed. Second, note that g maps R sites of $\mathcal{A}$ into one site of $\mathcal{B}$; that ensures that the computational load on the sites of $\mathcal{B}$ remains bounded and can be performed by cells which have the computational power of a finite state automaton. Third, the elementary data movements on $\mathcal{A}$

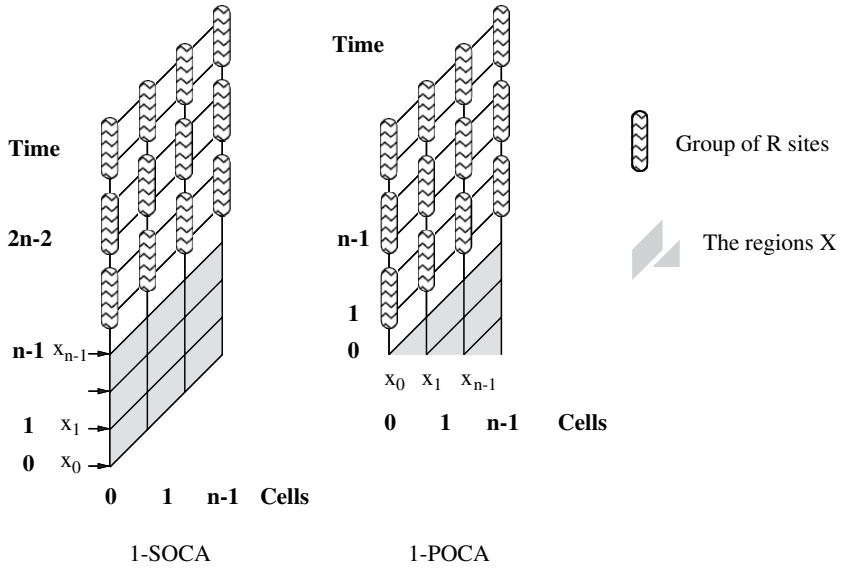


Fig. 3. One-way linear acceleration.

are the data movements induced in one time step by the one-way Von Neumann neighborhood. In other words they are vectors $(\gamma, 1)$ where $\gamma \in \{0, 1\}^k$ and $\sum \gamma_i \leqslant 1$. On $\mathcal{B}$, they are mapped into

$$(\varepsilon, \sigma) = g(\mathbf{c} + \gamma, t + 1) - g(\mathbf{c}, t)$$
$$= \left(\gamma, \left\lceil \frac{(R-1)(n-1+\sum c_i + \sum \gamma_i) + t + 1}{R} \right\rceil - \left\lceil \frac{(R-1)(n-1+\sum c_i) + t}{R} \right\rceil\right).$$

As $\sum \gamma_i \leqslant 1$, we have

$$\sigma \geqslant \left\lceil \frac{(R-1)(n-1+\sum c_i) + t}{R} + \sum \gamma_i \right\rceil - \left\lceil \frac{(R-1)(n-1+\sum c_i) + t}{R} \right\rceil \geqslant \sum \gamma_i.$$

Thus, on $\mathcal{B}$ governed by the one-way Von Neumann neighborhood, the dependencies constraints $\varepsilon_1, \ldots, \varepsilon_k \geqslant 0$ and $\sum \varepsilon_i \leqslant \sigma$ are fulfilled. Also remark that, as the computational load on the sites, the amount of data to be transmitted is bounded by a constant. Finally, observe that $g(\mathbf{n} - \mathbf{1}, rt(n) + f(n)) = (\mathbf{n} - \mathbf{1}, rt(n) + \lceil f(n)/R \rceil)$. □

In the same way, the computation on k-POCA can be speeded up.

Proposition 2. *Let $f, rt : \mathbb{N} \to \mathbb{N}$ be two functions where $rt(n) = k(n-1)$ is the real-time complexity for* k-POCA. *If a language L is recognized by a* k-POCA *in time $rt(n) + f(n)$ then, for any positive integer R, L is recognized by a* k-POCA *in time $rt(n) + \lceil f(n)/R \rceil$.*

Proof. First remark that for any k-POCA, the working area $W = \{(\mathbf{c}, t) : 0 \leqslant c_1, \ldots, c_k < n, t \geqslant 0\}$ can be divided into two regions $X = \{(\mathbf{c}, t) \in W : \sum c_i \geqslant t\}$ and $Y = \{(\mathbf{c}, t) \in W : \sum c_i < t\}$. For that, a wave is initiated on the initial cell $\mathbf{0}$, at time 1, when $\mathbf{0}$ knows its position as extremity. This wave spreads at maximal speed and so characterizes the set of sites $F = \{(\mathbf{c}, t) \in W : t = \sum c_i + 1\}$. Actually, this wave marks the first time where the cells know they have got the whole available part of the input. This wave corresponds to the lower layer of Y and so allows to divide the working area into X and Y.

Now consider any k-POCA $\mathcal{A}$ which recognizes the language L in time $rt(n) + f(n)$. Let us define the k-POCA $\mathcal{B}$ which simulates $\mathcal{A}$. On X, $\mathcal{B}$ behaves like $\mathcal{A}$. On Y, the computation of $\mathcal{A}$ is mapped on $\mathcal{B}$ according to the transformation $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ defined by

$$g(\mathbf{c}, t) = \left(\mathbf{c}, \left\lceil \frac{(R-1)\sum c_i + t}{R} \right\rceil\right).$$

The same arguments, used in the previous proposition, apply to verify the validity of the transformation. Finally, observe that $g(\mathbf{n}-\mathbf{1}, rt(n)+f(n)) = (\mathbf{n}-\mathbf{1}, rt(n)+\lceil f(n)/R\rceil)$. □

4. Simulation

In this section, we consider relationships between the real-time and linear time classes of k-OCA. All presented simulations consist in simple transformations of the working areas.

It has been set that 1-RSOCA and 1-LPOCA define the same class of languages [10]. Let us see that this equivalence extends to the higher dimensions.

Fact 1. k-RSOCA $\subseteq$ k-LPOCA.

Proof. For any k-RSOCA $\mathcal{A}$, let us define a k-LPOCA $\mathcal{B}$ which simulates it. Initially, the k-LPOCA $\mathcal{B}$ achieves a preprocessing phase to mimic the sequential input mode from the parallel input mode. Each symbol x_i fed on the cell $(i, 0, \ldots, 0)$ remains on the same cell. In this way, x_i which is fed on the site $(\mathbf{0}, i)$ of the k-RSOCA $\mathcal{A}$, is known on the site $(i, 0, \ldots, 0, i)$ of the k-LPOCA $\mathcal{B}$.

Further, as viewed in proof of Proposition 2, the k-LPOCA $\mathcal{B}$ can characterize the region $Y = \{(\mathbf{c}, t) : \sum c_i < t\}$. In Y, the sites of $\mathcal{A}$ are mapped according to the linear transformation $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ defined by

$$g(\mathbf{c}, t) = \left(t - \sum c_i, c_2, \ldots, c_k, t\right).$$

Let us verify that g maps the accepting (resp. non-accepting) computations of the k-RSOCA $\mathcal{A}$ into accepting (resp. non-accepting) computations of a k-LPOCA $\mathcal{B}$. First, note that $g(\mathbf{0}, i) = (i, 0, \ldots, 0, i)$. So, as x_i is known on the site $(i, 0\cdots, 0, i)$ of $\mathcal{B}$ and $(i, 0\ldots, 0, i)$ is positioned on the frontier with Y, the transition is guaranteed. Second, the injectivity of g guarantees that, on $\mathcal{B}$, the cells can cope the computational load requirement. Third, the elementary data movements on $\mathcal{A}$ are vectors $(\boldsymbol{\gamma}, 1)$ where $\boldsymbol{\gamma} \in \{0, 1\}^k$ and $\sum \gamma_i \leqslant 1$. On $\mathcal{B}$, they are mapped into $(\boldsymbol{\varepsilon}, \sigma) = g(\mathbf{c}+\boldsymbol{\gamma}, t+1) - g(\mathbf{c}, t) = (1 - \sum \gamma_i, \gamma_2, \ldots, \gamma_k, 1)$. They satisfy the dependencies constraints: $\varepsilon_1, \ldots, \varepsilon_k \geqslant 0$ and $\sum \varepsilon_i \leqslant \sigma$. Finally, $g(\mathbf{n}-\mathbf{1}, (k+1)(n-1)) = (\mathbf{n}-\mathbf{1}, (k+1)(n-1))$ guarantees the correspondence between the output sites. □

Fact 2. k-LPOCA $\subseteq$ k-RSOCA.

Proof. Let $\mathcal{A}$ be any k-LPOCA. Consider the transformation $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ defined by

$$g(\mathbf{c}, t) = \left(\left\lceil \frac{t - c_2 - \cdots - c_k}{2} \right\rceil, c_2, \ldots, c_k, \sum c_i + \left\lceil \frac{t - c_2 - \cdots - c_k}{2} \right\rceil\right).$$

Let us verify that g maps the accepting (resp. non-accepting) computations of the k-LPOCA $\mathcal{A}$ into accepting (resp. non-accepting) computations of a k-RSOCA $\mathcal{B}$. First, $g(i, 0, \ldots, 0) = (0, \ldots, 0, i)$ guarantees the transition from the parallel input mode into the sequential input mode. Second, note that g maps two sites on one site. That ensures that the computational load on the sites of the k-RSOCA remains bounded. Third, on $\mathcal{A}$, the elementary data movements $(\boldsymbol{\gamma}, 1)$ are such that $\boldsymbol{\gamma} \in \{0, 1\}^k$ and $\sum \gamma_i \leqslant 1$. They are mapped into $(\boldsymbol{\varepsilon}, \sigma) = g(\mathbf{c}+\boldsymbol{\gamma}, t+1) - g(\mathbf{c}, t) = (\alpha, \gamma_2, \ldots, \gamma_k, \alpha + \sum \gamma_i)$ with $\alpha = \lceil (t + 1 - c_2 - \cdots - c_k - \gamma_2 - \cdots - \gamma_k)/2\rceil - \lceil (t - c_2 - \cdots - c_k)/2\rceil$. They satisfy the dependencies constraints: $\varepsilon_1, \ldots, \varepsilon_k \geqslant 0$ and $\sum \varepsilon_i \leqslant \sigma$. Finally, $g(\mathbf{n}-\mathbf{1}, (k+1)(n-1)) = (\mathbf{n}-\mathbf{1}, (k+1)(n-1))$ guarantees the correspondence between the output sites. □

Corollary 1. k-RSOCA = k-LPOCA.

Remark 1. For dimension 1, it has also been set that 1-RPCA defines the same class of languages as 1-LPOCA and 1-RSOCA [4,10]. At first glance, for higher dimensions $k > 1$, it seems to be different. Indeed, for k-RPCA with Von Neumann neighborhood, the number of sites influenced by the rightest input symbol x_{n-1} is n, for all dimensions k.

So, when $k > 1$, we may establish that k-RPCA can be simulated by k-RSOCA but not the converse. On the other hand, we may exhibit simple transformations to show that k-RPCA with Moore neighborhood is equivalent to k-RSOCA (and k-LPOCA) with one-way Von Neumann neighborhood. Finally, just recall that Von Neumann and Moore neighborhood are identical in dimension 1.

Now, we will compare types of different dimensions.

Fact 3. k-RSOCA $\subseteq$ (k + 1)-RPOCA.

Proof. Let L be a language recognized by a k-RSOCA $\mathcal{A}$. Consider the linear transformation $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+2}$ defined by

$$g(\mathbf{c}, t) = \left(t - \sum c_i, c_1, \ldots, c_k, t\right).$$

Let us verify that g maps the accepting (resp. non-accepting) computations of a k-RSOCA into accepting (resp. non-accepting) computations of a (k + 1)-RPOCA. First, note that $g(\mathbf{0}, i) = (i, 0, \ldots, 0, i)$. So, as detailed in Fact 1, the input can be gotten on these sites. Second, the injectivity of g ensures that, on $\mathcal{B}$, the cells can cope the computational load requirement. Third, in $\mathcal{A}$, the data movements $(\boldsymbol{\gamma}, 1)$ are such that $\boldsymbol{\gamma} \in \{0, 1\}^k$ and $\sum \gamma_i \leqslant 1$. They are mapped into $(\varepsilon_1, \ldots, \varepsilon_{k+1}, \sigma) = g(\boldsymbol{\gamma}, 1) = (1 - \sum \gamma_i, \gamma_1, \ldots, \gamma_k, 1)$. They satisfy the dependencies constraints: $\varepsilon_1, \ldots, \varepsilon_{k+1} \geqslant 0$ and $\sum_{i=1}^{k+1} \varepsilon_i \leqslant \sigma$. Finally, $g(\mathbf{n} - \mathbf{1}, (k + 1)(n - 1)) = (\mathbf{n} - \mathbf{1}, (k + 1)(n - 1))$ guarantees the correspondence between the output sites. □

Fact 4. k-LSOCA $\subseteq$ (k + 1)-RSOCA.

Proof. For any k-LSOCA $\mathcal{A}$, we will define a (k + 1)-RSOCA $\mathcal{B}$ which simulates it. As seen in Proposition 1, the (k + 1)-RSOCA can distinguish the regions $X = \{(\mathbf{c}, t) : \sum c_i + n - 1 \geqslant t\}$ and $Y = \{(\mathbf{c}, t) : \sum c_i + n - 1 < t\}$. In the region X, the sites $(\mathbf{c}, t)$ of $\mathcal{A}$ such that $t \leqslant \sum c_i + n - 1$, are simply mapped on the sites $(0, c_1, \ldots, c_k, t)$ of $\mathcal{B}$. Then, in region Y, the sites $(\mathbf{c}, t)$ of $\mathcal{A}$ such that $t > \sum c_i + n - 1$, are mapped according to the affine transformation $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+2}$ defined by

$$g(\mathbf{c}, t) = \left(t - \sum c_i - n + 1, c_1, \ldots, c_k, t\right).$$

Let us verify the validity of the transformation. First, observe that $g(\mathbf{c}, \sum c_i + n - 1) = (0, c_1, \ldots, c_k, \sum c_i + n - 1)$. So, the transition between the two regions X and Y is guarantied. Second, observe that g corresponds to the linear transformation of the previous Fact 3 with the translation $(-n + 1, 0, \ldots, 0)$. So, in the same way, the load sites requirements and the dependencies constraints are satisfied. Finally, $g(\mathbf{n-1}, (k+2)(n-1)) = (n-1, \ldots, n-1, (k+2)(n - 1))$ ensures the correspondence between the output sites. □

Restricted to unary languages, 1-RPOCA is not more powerful than finite automaton (which may be considered as 0-RSOCA). For the same reasons, we obtain the following fact.

Fact 5. *Restricted to unary languages*, k-RSOCA *and* (k + 1)-RPOCA *have the same recognition ability.*

Proof. Let $\mathcal{A}$ be any (k + 1)-RPOCA whose input alphabet is unary. By definition, the sites $\{(c_1, \ldots, c_{k+1}, t) : c_2 + \cdots + c_{k+1} > t\}$ are in quiescent state. As the alphabet is unary, all sites $\{(i, 0, \ldots, 0, 0) : 0 \leqslant i < n\}$ are in the same state. So by induction on t, we may show that, for any fixed $c_2, \ldots, c_{k+1}$, t, the sites $\{(i, c_2, \ldots, c_{k+1}, t) : t - c_2 - \cdots - c_{k+1} \leqslant i < n\}$ share the same state. In particular, for all time $t < (k + 1)(n - 1)$, both states $\langle n - 2, c_2, \ldots, c_{k+1}, t\rangle_{\mathcal{A}}$ and $\langle n - 1, c_2, \ldots, c_{k+1}, t\rangle_{\mathcal{A}}$ are identical. So, observe that we may project the behavior of the (k + 1)-RPOCA $\mathcal{A}$ within the k-RSOCA $\mathcal{B}$ such that $\langle c_2, \ldots, c_{k+1}, t\rangle_{\mathcal{B}} = \langle n - 1, c_2, \ldots, c_{k+1}, t\rangle_{\mathcal{A}}$, since the missing neighbor state being the same as the current state. Finally, note that the output site of $\mathcal{B}$ enters the same state as the output site of $\mathcal{A}$: $\langle n - 1, \ldots, n - 1, (k + 1)(n - 1)\rangle_{\mathcal{B}} = \langle n - 1, n - 1, \ldots, n - 1, (k + 1)(n - 1)\rangle_{\mathcal{A}}$. □

5. Closure properties

In this section, we will consider closure properties of the classes k-RSOCA, k-RPOCA and k-LSOCA. These classes are obviously closed under boolean operations.

5.1. Closure under reverse

Closure under reverse has already been observed for k-RPOCA in case of dimension 1 and 2 [4,13], as well for k-LSOCA in case of dimension 1. For the higher dimensions, we will verify in the two next facts that the computation turns out to be also symmetrical for k-RPOCA and k-LSOCA. On the other hand, we ignore, for any dimension k, whether k-RSOCA is closed under reverse. Note that a negative answer, for some dimension k, will yield strict inclusions between k-RPOCA, k-RSOCA and k-LSOCA. Furthermore, it was shown that 1-RSOCA is closed under reverse if and only if 1-RSOCA is as powerful as 1-LSOCA [11]. We imagine that this result can be generalized for the higher dimensions, in using similar techniques.

Fact 6. k-RPOCA *is closed under reverse.*

Proof. Let L be a language recognized by a k-RPOCA $\mathcal{C}$. Consider the affine transformation $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ defined by

$$g(\mathbf{c}, t) = \left(n - 1 + t - \sum c_i, c_2, \ldots, c_k, t\right).$$

Let us verify that g maps the accepting (resp. non-accepting) computation of $\mathcal{C}$ on any input $w = x_0 \cdots x_{n-1}$ into an accepting (resp. non-accepting) computation of a k-RPOCA $\mathcal{D}$ on the input $w^R = x_{n-1} \cdots x_0$. First, $g(i, 0, \ldots, 0) = (n-1-i, 0, \ldots, 0)$ supports that, on $\mathcal{D}$, the symbol x_i of the input $w^R = x_{n-1} \cdots x_0$ is fed on the cell $(n-1-i, 0, \ldots, 0)$. Second, the injectivity of g guarantees that, on $\mathcal{D}$, the cells can cope the computational load requirements. Third, in the working area of $\mathcal{C}$, the data movements are vectors $(\boldsymbol{\gamma}, 1)$ where $\boldsymbol{\gamma} \in \{0, 1\}^k$ and $\sum \gamma_i \leqslant 1$. On $\mathcal{D}$, they are mapped into $(\boldsymbol{\varepsilon}, \sigma) = g(\mathbf{c}+\boldsymbol{\gamma}, t+1) - g(\mathbf{c}, t) = (1 - \sum \gamma_i, \gamma_2, \ldots, \gamma_k, 1)$ which satisfy the dependencies constraints: $\varepsilon_1, \ldots, \varepsilon_k \geqslant 0$ and $\sum \varepsilon_i \leqslant \sigma$. Finally, $g(\mathbf{n} - \mathbf{1}, k(n-1)) = (\mathbf{n} - \mathbf{1}, k(n-1))$ confirms that, on $\mathcal{C}$ and $\mathcal{D}$, the results are collected on the same output site. □

Fact 7. k-LSOCA *is closed under reverse.*

Proof. Let L be a language recognized by a k-LSOCA $\mathcal{A}$. We will define a k-LSOCA $\mathcal{B}$ which recognizes L^R. Initially, the k-LSOCA $\mathcal{B}$ achieves a preprocessing phase. Each symbol x_i of the input $w^R = x_{n-1} \ldots x_0$, fed on the site $(\mathbf{0}, n-1-i)$, is carried according to the following sequence of moves: one step to the neighbor cell in the direction of $(1, 0, \ldots, 0)$ and two steps on the same cell. In particular, x_i goes through the sites $\{(s, 0, \ldots, 0, n-1-i+3s-a) : a = 0, 1, s \geqslant 0\}$. Thus x_i reaches the line $\{(b, 0, \ldots, 0, n-1+b) : b \geqslant 0\}$ on the site $(\lceil i/2 \rceil, 0, \ldots, 0, n-1+\lceil i/2 \rceil)$. Further, as viewed in Proposition 1, the k-LSOCA $\mathcal{B}$ can characterize the region $Y = \{(\mathbf{c}, t) : \sum c_i + n - 1 < t\}$. In Y, the sites of $\mathcal{A}$ are mapped according to the transformation $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ defined by

$$g(\mathbf{c}, t) = \left(\left\lceil \frac{t - \sum c_i}{2} \right\rceil, c_2, \ldots, c_k, \left\lceil \frac{t + \sum c_i}{2} \right\rceil + n - 1\right).$$

Let us verify that the accepting (resp. non-accepting) computation of $\mathcal{A}$ on any input $w = x_0 \ldots x_{n-1}$ are mapped into an accepting (resp. non-accepting) computation of the k-RPOCA $\mathcal{B}$ on the input $w^R = x_{n-1} \cdots x_0$. First, observe that $g(0, \ldots, 0, i) = (\lceil i/2 \rceil, 0, \ldots, 0, n-1+\lceil i/2 \rceil)$, and x_i is known on this site according to the preprocessing phase. Second, note that g maps two sites into one site. That guarantees that, on $\mathcal{B}$, the cells can cope the computational load requirements. Third, in the working area of $\mathcal{A}$, the data movements are vectors $(\boldsymbol{\gamma}, 1)$ where $\boldsymbol{\gamma} \in \{0, 1\}^k$ and $\sum \gamma_i \leqslant 1$. On $\mathcal{B}$, they are mapped into $(\boldsymbol{\varepsilon}, \sigma) = (\alpha, \gamma_2, \ldots, \gamma_k, \alpha + \sum \gamma_i)$ with $\alpha = \lceil (t+1-\sum c_i - \sum \gamma_i)/2 \rceil - \lceil (t - \sum c_i)/2 \rceil$. They satisfy the dependencies constraints: $\varepsilon_1, \ldots, \varepsilon_k \geqslant 0$ and $\sum \varepsilon_i \leqslant \sigma$. Finally, $g(\mathbf{n}-\mathbf{1}, (k+2)(n-1)) = (\mathbf{n}-\mathbf{1}, (k+2)(n-1))$ confirms that, on $\mathcal{C}$ and $\mathcal{D}$, the results are collected on the same output site. □

5.2. Closure under concatenation

The questions of the closure under concatenation of k-RSOCA and k-LSOCA are open. However, as proved in case of dimension one in [11], the class of unary languages recognizes by k-RSOCA is closed under concatenation. On other hand, we know that 1-RPOCA is not closed under concatenation [17]. Surprisingly, we will see that k-RPOCA are closed under concatenation for the higher dimensions.

Fact 8. *For any dimension* $k > 1$, *if* L_1 *and* L_2 *are recognized by* k-RPOCA *then* $X = \{uv : u \in L_1, v \in L_2$ *and* $|u| \leqslant |v|\}$ *is recognized by* k-RPOCA.

Proof. Recall that, by definition, the sites $\{(c_1, c_2, \ldots, c_k, t) : t < c_2 + \cdots + c_k\}$ are in quiescent state, since they are not affected by the input. Then a characteristic of k-RPOCA is that the computation on any word $w = x_0 \ldots x_{n-1}$ contains the computation of all its subwords $x_i \cdots x_j$, where $0 \leqslant i \leqslant j < n$. In particular, the acceptance of $x_i \cdots x_j$ is determined on the site $(j, j-i, \ldots, j-i, k(j-i))$. So the proof comes down to construct a k-RPOCA $\mathcal{A}$ which characterizes the set of sites

$$\mathcal{I} = \{(j, j-i, \ldots, j-i, k(j-i)) : x_i \cdots x_h \in L_1, x_{h+1} \cdots x_j \in L_2 \text{ for some } h < (i+j)/2\}.$$

First, $\mathcal{A}$ simulates the both k-RPOCA which recognize L_1 and L_2. In this way, the sets of sites

$$\mathcal{E} = \{(b, b-a, \ldots, b-a, k(b-a)) : x_a \cdots x_b \in L_1\}$$

and

$$\mathcal{F} = \{(d, d-c, \ldots, d-c, k(d-c)) : x_c \cdots x_d \in L_2\}$$

are characterized. Now, from each site of $\mathcal{E}$, a signal S is initialized; it runs by a sequence of elementary moves $(1, 0, \ldots, 0, 1)$. So this family of signals S goes through the sites $\{(b+t, b-a, \ldots, b-a, k(b-a)+t) : x_a \cdots x_b \in L_1$ and $t \geqslant 0\}$. Parallelly, a family of signals T starts from the sites $(z, 0, \ldots, 0)$ with $0 \leqslant z < n$. Each signal T operates first an elementary move $(1, 0, \ldots, 0, 1)$, then a sequence of moves $(2, 1, \ldots, 1, k+1)$. So this family of signals T characterizes the sites $\{(z+1+2s, s, \ldots, s, 1+s(k+1)) : 0 \leqslant z < n, s \geqslant 0\}$. Then the signals S and T cross each other on the sites:

$$\mathcal{G} = \{(2b-a+1, b-a, \ldots, b-a, (k+1)(b-a)+1) : x_a \cdots x_b \in L_1\}.$$

From each site of $\mathcal{G}$, a signal U is initialized; it runs by a sequence of elementary moves $(1, \ldots, 1, k)$. So the family of signals U goes through the sites $\{(2b-a+1+t, b-a+t, \ldots, b-a+t, (k+1)(b-a)+1+kt) : x_a \cdots x_b \in L_1$ and $t \geqslant 0\}$.

On the other hand, from each site of $\mathcal{F}$ (the set of accepting sites associated to L_2), a signal V is initialized; it remains all the time on the same cell. So this family of signals V goes through the sites $\{(d, d-c, \ldots, d-c, k(d-c)+s) : x_c \cdots x_d \in L_2$ and $s \geqslant 0\}$. Then the signals U and V cross each other on the sites:

$$\mathcal{H} = \{(d, d-b-1, \ldots, d-b-1, k(d-b-1)+b-a+1) : x_a \cdots x_b \in L_1, x_{b+1} \cdots x_d \in L_2 \text{ and } b < (a+d)/2\}.$$

Finally from each site of $\mathcal{H}$, a signal starts and runs with elementary move $(0, 1, \ldots, 1, k-1)$, it reaches the output site $(d, d-x, \ldots, d-x, k(d-x))$. From $d-b-1+t = d-x$ and $k(d-b-1)+b-a+1+(k-1)t = k(d-x)$, we get $t = b-a+1$ and $x = a$. That means we know on the site $(d, d-a, \ldots, d-a, k(d-a))$ whether $x_a \ldots x_b \in L_1, x_{b+1} \ldots x_d \in L_2$ for $b < (a+d)/2$. So we have all the required information to characterize the expected set $\mathcal{I} = \{(d, d-a, \ldots, d-a, k(d-a)) : x_a \cdots x_b \in L_1, x_{b+1} \cdots x_d \in L_2$ for some $b < (a+d)/2\}$. □

Corollary 2. *For any dimension* $k > 1$, k-RPOCA *is closed under concatenation.*

Proof. Let L_1 and L_2 be two languages recognized by k-RPOCA. According to Fact 8,

$$X = \{uv : u \in L_1, v \in L_2 \text{ and } |u| \leqslant |v|\}$$

is recognized by k-RPOCA. Moreover, as k-RPOCA is closed under reverse,

$$Y = \{vu : v \in L_2^R, u \in L_1^R \text{ and } |v| \leqslant |u|\}$$

as well

$$Y^R = \{uv : u \in L_1, v \in L_2 \text{ and } |u| \geqslant |v|\}$$

are recognized by k-RPOCA. Thus, $X \cup Y^R = L_1 L_2$ is a k-RPOCA language. □

The closure under Kleene star simply derives from the closure under concatenation.

Fact 9. *For any dimension $k > 1$,* k-RPOCA *is closed under Kleene star.*

Proof. According to Fact 8 and Corollary 2, we can characterize, by means of signals, from the sets of sites

$$\mathcal{E} = \{(b, b-a, \ldots, b-a, k(b-a)) : x_a \cdots x_b \in L_1\}$$

and

$$\mathcal{F} = \{(b, b-a, \ldots, b-a, k(b-a)) : x_a \cdots x_b \in L_2\}$$

the set of sites

$$\mathcal{I} = \{(c, c-a, \ldots, c-a, k(c-a)) : x_a \cdots x_b \in L_1 \text{ and } x_{b+1} \cdots x_c \in L_2\}.$$

Now define recursively the set $\mathcal{F}$ as the sites of $\mathcal{E}$ and $\mathcal{I}$. Thus, the characterization of $\mathcal{F}$ allows to recognize the set $L_2 = L_1 + L_1 L_2$; i.e. $L_2 = L_1^+$. □

Putting Corollary 2, Facts 9 and 5 together, we obtain the following corollary.

Corollary 3. *For any positive integer k, the class of unary languages recognized by* k-RSOCA *is closed under concatenation and Kleene star.*

6. Relationship with alternating device

Relationships between CA and alternating devices are common. By instance, it has been shown that real-time one-dimensional CA can be simulated by two-dimensional alternating finite automata [14]. It is also known that real-time k-dimensional iterative arrays are equivalent through reverse to real-time one-way alternating k counter automata [18]. Here, we will be concerned with one-way multihead alternating finite automata. This simple alternating device was introduced by King [15]; see also [16] for further investigations. Precisely, we will show that k-RSOCA are equivalent through reverse to one-way alternating finite automata with $k + 1$ heads. This relationship stresses that k-RSOCA define significant complexity classes.

6.1. Preliminaries

To begin, we have to recall the definition of one-way multihead alternating finite automata. A *one-way alternating finite automata with k heads* (in short 1AFA(k)) is a sextuplet $(\Sigma, S_\mathcal{M}, \delta_\mathcal{M}, s_{\text{init}}, U_\mathcal{M}, F_\mathcal{M})$ where Σ is the input alphabet, $S_\mathcal{M}$ is the set of states, $\delta_\mathcal{M} : S_\mathcal{M} \times (\Sigma \cup \{\$\})^k \to \mathcal{P}(S_\mathcal{M} \times \{0,1\}^k)$ is the transition function, $s_{\text{init}} \in S_\mathcal{M}$ is the initial state, $U_\mathcal{M} \subset S_\mathcal{M}$ is the set of universal states, $S_\mathcal{M} \backslash U_\mathcal{M}$ is the set of existential states and $F_\mathcal{M} \subset S_\mathcal{M}$ is the set of accepting states. The input w is delimited with the end-marker \$ placed on its right. At initial time, M starts the computation in the initial state s_{init}, its k heads reading the first symbol ($\theta_1 = \cdots = \theta_k = 0$). At each step, M evolves according to the transition function δ, the current state s, the symbols $a_1, \ldots, a_k$ scanned by the k heads. For any $(s', (d_1, \ldots, d_k)) \in \delta(s, (a_1, \ldots, a_k))$, the state is updated to s' and the k heads move of $d_1, \ldots, d_k$.

A *configuration* describes a computation step of M on the input w; it is an element of $\{0,\dots,n-1\}^k \times S$ which codes the positions of the k heads and the current state. A configuration $\mathcal{J}$ is an immediate successor of a configuration $\mathcal{I} = ((\theta_1,\dots,\theta_k), s)$, written $\mathcal{I} \vdash \mathcal{J}$, if $\mathcal{J} = ((\theta_1+d_1,\dots,\theta_k+d_k), s')$ and $(s', (d_1,\dots,d_k)) \in \delta(s, (a_1,\dots,a_k))$. A configuration is universal (resp. existential) if its state s is universal (resp. existential). A configuration is accepting if its state s is accepting and the k heads read the end-marker ($\theta_1 = \cdots = \theta_k = n$). The initial configuration is $((0,\dots,0), s_{\text{init}})$.

An *accepting computation tree* of M on the input w is a labeled tree such that:

1. each node is labeled by a configuration;
2. the root is labeled by the initial configuration;
3. if an internal node is labeled by an universal configuration $\mathcal{I}$ whose all successors are $\{\mathcal{J}_1,\dots,\mathcal{J}_r\}$, it has exactly r children $\mathcal{J}_1,\dots,\mathcal{J}_r$;
4. if an internal node is labeled by an existential configuration $\mathcal{I}$, it has exactly one child $\mathcal{J}$ such that $\mathcal{I} \vdash \mathcal{J}$;
5. the leaves are labeled by accepting configurations.

An *accepting computation subtree* of M on the input w satisfies the same conditions as an accepting computation tree, except that the root is labeled by any configuration.

A 1AFA M accepts a word w, if there is an accepting computation tree of M on w. The language accepted by M is the set $\{w \in \Sigma^* : M \text{ accepts } w\}$.

For technical reasons, we will use the following fact.

Fact 10. *Let L be any language on an alphabet Σ and \$ be any symbol not in Σ. L is recognized by a* k-RSOCA *if and only if $\{\$\}L$ is recognized by a* k-RSOCA.

Proof. The only if part is trivial. From a k-RSOCA $\mathcal{A}$ which recognizes a language L, we can directly construct a k-RSOCA $\mathcal{B}$ which recognizes $\{\$\}L$. Indeed provided that the k-RSOCA $\mathcal{B}$ sends the input symbols from the cell $\mathbf{0}$ to the cell $\mathbf{1}$, what can be computed on the site $(\mathbf{c}, t)_{\mathcal{A}}$ can be computed in the same way on the site $(\mathbf{c}+\mathbf{1}, t+k+1)_{\mathcal{B}}$.

Conversely, consider a k-RSOCA $\mathcal{C}$ which recognizes $\{\$\}L$. To construct a k-RSOCA $\mathcal{D}$ recognizing L, it suffices to simulate the missing first symbol \$. For that, the sites bordering the working area will have extra work to achieve. Formally, the state $\langle \mathbf{c}, \sum c_i + t\rangle_{\mathcal{D}}$ will consist in the following states of the initial k-SOCA $\mathcal{C}$:

$$\left\{ \left\langle c_1+\varepsilon_1,\dots,c_k+\varepsilon_k, \sum(c_i+\varepsilon_i)+t+\varepsilon \right\rangle_{\mathcal{C}} : \right.$$
$$\left. \varepsilon_i = 0,1 \text{ if } c_i = 1,\ \varepsilon_i = 1 \text{ if } c_i > 1,\ \varepsilon = 0,1 \text{ if } t = 0,\ \varepsilon = 1 \text{ if } t > 0 \right\}.$$

That may be proved by recurrence on $r = \sum c_i + t$, but we do not give the details. So $\langle \mathbf{n}-\mathbf{1}, (k+1)(n-1)\rangle_{\mathcal{D}}$ consists in $\langle \mathbf{n}, (k+1)n\rangle_{\mathcal{C}}$ and $\mathcal{D}$ enters an accepting state on input w iff $\mathcal{C}$ enters an accepting state on input $\$w$. □

6.2. k-RSOCA $\subseteq$ 1AFA(k + 1)^R

In this paragraph, we will present how to code the evolution of a k-RSOCA by a 1AFA(k + 1) employing one head for each of the k dimensions and one head for the time.

We will use below the following notations. $\boldsymbol{\varepsilon}^0 = (0,\dots,0)$ denotes the vector whose all $k+1$ components are null and $\boldsymbol{\varepsilon}^i = (\varepsilon^i_1,\dots,\varepsilon^i_{k+1})$ denotes the vector whose all components are null except the ith one set to 1 ($\varepsilon^i_j = 1$ if $i = j$, 0 otherwise).

Proposition 3. *Let L be a language on the alphabet Σ and $\mathcal{A} = (\Sigma, S_A, \delta_A^{\text{init}}, \delta_A, F_A, \sharp)$ be a* k-RSOCA *which recognizes $\{\$\}L$. The reverse of L is accepted by the* 1AFA(k + 1) $\mathcal{M} = (\Sigma, S_{\mathcal{M}}, \delta_{\mathcal{M}}, s_{\text{init}}, U_{\mathcal{M}}, F_{\mathcal{M}})$ *where $U_{\mathcal{M}} = S_A^{k+1}$, $S_{\mathcal{M}} = U_{\mathcal{M}} \cup S_A \cup \{s_{\text{init}}\} \cup \{s_{\text{accept}}\}$, $F_{\mathcal{M}} = \{s_{\text{accept}}\}$ and the transition function $\delta_{\mathcal{M}}$ is defined by*:

- $\delta_{\mathcal{M}}(s_{\text{init}}, (a_1,\dots,a_{k+1})) = \{(s, \boldsymbol{\varepsilon}^0) : s \in F_A\}$.
- $\delta_{\mathcal{M}}(s, (\$,\dots,\$)) = \begin{cases} \{(s_{\text{accept}}, \boldsymbol{\varepsilon}^0)\} & \text{if } \delta_A^{\text{init}}(\$, \sharp) = s, \\ \emptyset & \text{else.} \end{cases}$

- $\delta_{\mathcal{M}}(s, (\$, \ldots, \$, a)) = \{(s', \boldsymbol{\varepsilon}^{k+1}) : \delta_A^{\text{init}}(a, s') = s\}$.
- *For* $(a_1, \ldots, a_k) \neq (\$, \ldots, \$)$, $\delta_{\mathcal{M}}(s, (a_1, \ldots, a_{k+1})) = \{((s_1, \ldots, s_{k+1}), \boldsymbol{\varepsilon}^0) : \delta_A(s_1, \ldots, s_{k+1}) = s\}$.
- $\delta_{\mathcal{M}}((s_1, \ldots, s_{k+1}), (a_1, \ldots, a_{k+1})) = \{(s_i, \boldsymbol{\varepsilon}^i) : 1 \leqslant i \leqslant k+1 \text{ and } s_i \neq \sharp\}$.

To prove Proposition 3, we will need the following fact which precises the correspondence between the roots of accepting computation subtrees of the 1AFA and the states of the SOCA.

Fact 11. *Let $w = x_0 \cdots x_{n-1}$ be any word on Σ. Consider the computation of $\mathcal{A}$ on the input $\$w$ and the computation of $\mathcal{M}$ on the input w^R. For $s \in S_A$ holds $((\theta_1, \ldots, \theta_{k+1}), s)$ is the root of an accepting computation subtree of $\mathcal{M}$ if and only if $s = \langle n - \theta_1, \ldots, n - \theta_k, \sum_{i=1}^{k+1}(n - \theta_i)\rangle_A$.*

Proof. The proof is done by induction on the sum $\sum_{i=1}^{k+1} \theta_i$. First, we prove the basis step when $\sum \theta_i = (k+1)n$. In this case, the positions of the $k+1$ heads are n. And $((n, \ldots, n), s)$ is the root of an accepting computation subtree if and only if it has one successor labeled by the accepting configuration. Hence $s = \delta_A^{\text{init}}(\$, \sharp) = \langle \mathbf{0}, 0\rangle_A$.

Now consider the inductive step. Let us assume the proposition be true when $\sum \theta_i > r$. And let us prove it for $\sum \theta_i = r$. We distinguish two cases according to the positions of the first k heads.

Case 1: $\theta_1 = \cdots = \theta_k = n$. In this case, $((n, \ldots, n, \theta_{k+1}), s)$ is the root of an accepting computation subtree if and only if it has one successor labeled by $((n, \ldots, n, \theta_{k+1} + 1), s')$ with $\delta_A^{\text{init}}(x_{n-\theta_{k+1}}, s') = s$. By hypothesis, $s' = \langle \mathbf{0}, n - \theta_{k+1} - 1\rangle_A$. It follows that $s = \delta_A^{\text{init}}(x_{n-\theta_{k+1}}, \langle \mathbf{0}, n - \theta_{k+1} - 1\rangle_A) = \langle \mathbf{0}, n - \theta_{k+1}\rangle_A$.

Case 2: $(\theta_1, \ldots, \theta_k) \neq (n, \ldots, n)$. In this case, the configuration $((\theta_1, \ldots, \theta_{k+1}), s)$ has one successor labeled by $((\theta_1, \ldots, \theta_k, \theta_{k+1}), (s_1, \ldots, s_{k+1}))$ with $\delta_A(s_1, \ldots, s_{k+1}) = s$. This universal configuration has the successors $((\theta_1, \ldots, \theta_i + 1, \ldots, \theta_k, \theta_{k+1}), s_i)$ for every $i \leqslant k$ such that $\theta_i < n$ and the successor $((\theta_1, \ldots, \theta_k, \theta_{k+1} + 1), s_{k+1})$ if $\theta_{k+1} < n$. By hypothesis, for all $i \leqslant k$, $s_i = (\langle n - \theta_1, \ldots, n - \theta_i - 1, \ldots, n - \theta_k, \sum (n - \theta_i) - 1\rangle_A$ if $\theta_i < n$, $s_i = \sharp$ otherwise, and $s_{k+1} = \langle n - \theta_1, \ldots, n - \theta_k, \sum (n - \theta_i) - 1\rangle_A)$ if $\theta_{k+1} < n$, $s_{k+1} = \sharp$ otherwise. □

From this fact, the proof of Proposition 3 follows immediately:

Proof of Proposition 3. $\$x_0 \cdots x_{n-1}$ is accepted by $\mathcal{A}$ if and only if on input $\$x_0 \cdots x_{n-1}$, $\mathcal{A}$ enters an accepting state on site $(\mathbf{n}, (k+1)n)_{\mathcal{A}}$. According Fact 11, it is equivalent to the existence of an accepting computation subtree of $\mathcal{M}$ on $x_{n-1} \cdots x_0$ with root $((0, \ldots, 0), s)$ where $s \in F_{\mathcal{A}}$. That is equivalent to the existence of an accepting computation subtree of $\mathcal{M}$ on $x_{n-1} \cdots x_0$ whose root $((0, \ldots, 0), s_{\text{init}})$ has one successor labeled by $((0, \ldots, 0), s)$ with $s \in F_{\mathcal{A}}$. That is equivalent to the fact that M accepts $x_{n-1} \cdots x_0$. □

6.3. 1AFA(k + 1) ⊆ k-RSOCA^R

Now we will translate the computation of 1AFA(k + 1) in terms of k-RSOCA.

Proposition 4. *Let L be a language which is accepted by a* 1AFA(k + 1). *Then there exists a* k-RSOCA *which recognizes* $\{\$\}L^R$.

Proof. Let $w = x_0 \cdots x_{n-1}$ be a given word and $\mathcal{M} = (\Sigma, S_{\mathcal{M}}, \delta_{\mathcal{M}}, s_{\text{init}}, U_{\mathcal{M}}, F_{\mathcal{M}})$ be any 1AFA(k + 1) which accepts L. For integers $\theta_1, \ldots, \theta_{k+1}$, with $0 \leqslant \theta_1, \ldots, \theta_{k+1} \leqslant n$, we denote by $Q(\theta_1, \ldots, \theta_{k+1})$ the set

$$\{s \in S_{\mathcal{M}} : ((\theta_1, \ldots, \theta_{k+1}), s) \text{ is the root of an accepting computation subtree}\}.$$

In Fact 14, we will show that there exists a k-RSOCA $\mathcal{A}$ which, on input $\$x_{n-1} \cdots x_0$, computes the set $Q(n - c_1, \ldots, n - c_k, \sum c_i + n - t)$ on each site $(\mathbf{c}, t)$ when $t \leqslant n + \sum c_i$. In particular, $\mathcal{A}$ computes the set $Q(0, \ldots, 0)$ on the output site $(\mathbf{n}, (k+1)n)$. Furthermore, $\mathcal{A}$ accepts the input $\$x_{n-1} \cdots x_0$ provided the state s_{init} belongs to $Q(0, \ldots, 0)$, that means whether there exists an accepting computation tree on $x_0 \ldots x_{n-1}$. □

It remains to establish Fact 14. First, we will need this technical fact.

Fact 12. *Let M be a* 1AFA *with $k+1$ heads. There exists a* 1AFA M' *with $k+1$ heads which accepts the same language and such that the $k+1$th head θ_{k+1} has the lowest position during all the computation*: $\theta_{k+1} = \min_{i=1,\ldots,k+1} \theta_i$.

Proof. King has proved that multihead finite automata have the ability to detect coincidence of heads [15]. So we may assume that when the head θ_{k+1} meets and will overtake another head, the roles of the two heads are swapped. □

Further, to prove Fact 14, we have to explicit how to construct the k-RSOCA $\mathcal{A}$. The following fact will describe the move of input symbols in $\mathcal{A}$.

Fact 13. *There exists a* k-RSOCA *such that on input $x_n x_{n-1} \ldots x_0$*:
1. *The sites* $(\mathbf{c}, t)$, *with* $t \geqslant \max\ c_i + \sum c_i$, *can be distinguished.*
2. *Any cell* $\mathbf{c}$ *receives the k symbols* $x_{n-c_1}, \ldots, x_{n-c_k}$, *at every time t with* $t \geqslant \max c_i + \sum c_i$.
3. *Any cell* $\mathbf{c}$ *receives the symbol* $x_{n-t+\sum c_i}$, *at every time t with* $\sum c_i \leqslant t \leqslant n + \sum c_i$.

Proof. For each direction $d = 1, \ldots, k$, each input symbol x_p, with $0 \leqslant p \leqslant n$, will be spread into the working area according to the following process. A signal O_d is initialized on the site $(\mathbf{0}, 0)$; it moves in direction d at speed $\frac{1}{2}$; so O_d visits the sites: $\{(\mathbf{c}, 2c_d) : c_d \geqslant 0 \text{ and } (c_i = 0 \text{ for } i \neq d)\}$. In parallel, each symbol x_p, received by the input cell $\mathbf{0}$ at time $n-p$, is carried by the following signal $A_{d,p}$. This signal $A_{d,p}$ starts on the site $(\mathbf{0}, n-p)$; it moves in direction d at maximal speed and visits the sites: $\{(\mathbf{c}, c_d + n - p) : c_d \geqslant 0 \text{ and } (c_i = 0 \text{ for } i \neq d)\}$. Then the two signals O_d and $A_{d,p}$ intersect on the site $P_{d,p} = (\mathbf{c}, 2(n-p))$ where $c_d = n - p$ and $c_i = 0$ for $i \neq d$. From this site $P_{d,p}$, the symbol x_p is spread in all directions except the direction d, at maximal speed. So the symbol x_p goes through the sites: $W_{d,p} = \{(\mathbf{c}, c_d + \sum c_i) : c_d = n - p \text{ and } c_i \geqslant 0\}$. Finally, from each site of $W_{d,p}$, x_p moves according to the time axis. So the symbol x_p goes through the sites: $V_{d,p} = \{(\mathbf{c}, c_d + \sum c_i + s) : c_d = n - p, c_i \geqslant 0 \text{ and } s \geqslant 0\}$.

Now observe that a cell $\mathbf{c}$ receives at time t, a signal of type V_d provided $t \geqslant \sum c_i + c_d$. Precisely, for any time $t \geqslant \max c_i + \sum c_i$, the cell $\mathbf{c}$ receives exactly the k input symbols $x_{n-c_1}, \ldots, x_{n-c_k}$ carried by signals of type V (respectively by $V_{1,n-c_1}, \ldots, V_{k,n-c_k}$). On the other hand, for any time $t < \max c_i + \sum c_i$, $\mathbf{c}$ receives strictly less than k signals of type V. Thus the two first points are satisfied.

For the last point, each symbol x_p, received by the input cell $\mathbf{0}$ at time $n-p$, is carried by the following signal U_p. U_p starts on the site $(\mathbf{0}, n-p)$ and spreads at maximal speed in all the directions. So the symbol x_p goes through the sites: $U_p = \{(\mathbf{c}, (n-p) + \sum c_i) : c_i \geqslant 0\}$. Now observe that the cell $\mathbf{c}$ receives at time t a signal of type U provided $t = n - p + \sum c_i$ for some $p = 0, \ldots, n$. That means the cell $\mathbf{c}$ receives at time t the input symbol $x_{n-t+\sum c_i}$ when t is such that $\sum c_i \leqslant t \leqslant n + \sum c_i$. □

Finally, let us complete the construction of the simulating k-RSOCA.

Fact 14. *Let $\mathcal{M}$ be a given* 1AFA(k + 1) *with input $x_0 \cdots x_{n-1}$. There exists a* k-RSOCA *such that on input $\$x_{n-1} \cdots x_0$, the site $(\mathbf{c}, t)$ computes $Q(n - c_1, \ldots, n - c_k, \sum c_i + n - t)$ when $t \leqslant n + \sum c_i$.*

Proof. According to Fact 12, we may suppose that the last head θ_{k+1} has always the last position during all the computation of $\mathcal{M}$. Thus if $\theta_{k+1} > \min_{i=1,\ldots,k} \theta_i$, we have $Q(\theta_1, \ldots, \theta_{k+1}) = \emptyset$. In other part, each site $(\mathbf{c}, t)$ with $t < \max c_i + \sum c_i$ must compute $Q(n - c_1, \ldots, n - c_k, \sum c_i + n - t)$ which is the empty set as $\sum c_i + n - t > n - \max c_i = \min(n - c_i)$. Note that such sites can compute the empty set since they can be characterized according to Fact 13.

Now it remains to verify the proposition for all sites $(\mathbf{c}, t)$ with $t \leqslant n + \sum c_i$ and $t \geqslant \max c_i + \sum c_i$. It is done by recurrence on t. At time $t = 0$, the only involved cell is the input cell $\mathbf{0}$. The computed value is $F_{\mathcal{M}}$ which is equal to $Q(n, \ldots, n)$. Next we suppose the proposition true up to t and we will prove it for $t+1$. The site $(\mathbf{c}, t)$ knows the values $Q(n - c_1 + 1, \ldots, n - c_k, \sum c_i + n - t), \ldots, Q(n - c_1, \ldots, n - c_k + 1, \sum c_i + n - t)$, $Q(n - c_1, \ldots, n - c_k, \sum c_i + n - t + 1)$ computed, respectively, on the sites $(c_1 - 1, \ldots, c_k, t-1), \ldots, (c_1, \ldots, c_k - 1, t-1), (c_1, \ldots, c_k, t-1)$. Moreover, according to Fact 13, the site receives the input symbols $x_{n-c_1}, \ldots, x_{n-c_k}, x_{n-t+\sum c_i}$. So it has all the required information to compute $Q(n - c_1, \ldots, n - c_k, \sum c_i + n - t)$. □

Corollary 4. *For any positive integer k,* 1AFA(k + 1) *and* k-RSOCA^R *recognize the same class of languages.*

Unfortunately, for 1AFA, the question whether adding heads increases the recognition power is unanswered. And closure under reverse of 1AFA(k + 1) is also unknown. Nevertheless, this correspondence enlightens the result of [2] which sets that linear time alternating Turing machine can simulate linear time multidimensional POCA.

Concerning context-free languages (CFL), it has been proved in [1], that CFL ⊂ 2-LSOCA. Actually, this inclusion can be refined.

Corollary 5. CFL ⊂ 2-RSOCA.

Proof. In [15], King has exhibited relationships between one-way multihead non-deterministic pushdown automata and one-way multihead alternating finite automata: 1NPDA(k) ⊂ 1AFA($3k$). In particular, CFL ⊂ 1AFA(3). Furthermore, CFL is closed under reverse. □

7. Conclusion

In this paper, we have investigated an other hierarchy between the class of languages recognized in minimal time and the class of languages recognized in unrestricted time by one-dimensional space bounded CA. This hierarchy was defined by increasing the dimensionality of the array. We have concentrated on the low complexity classes of two representative types: the k-SOCA and the k-POCA. Fig. 4 summarizes the inclusion relationships inside $\bigcup$ k-RSOCA. They have been shown in the previous sections or are inferred from the closure property under reverse of k-RPOCA and k-LSOCA.

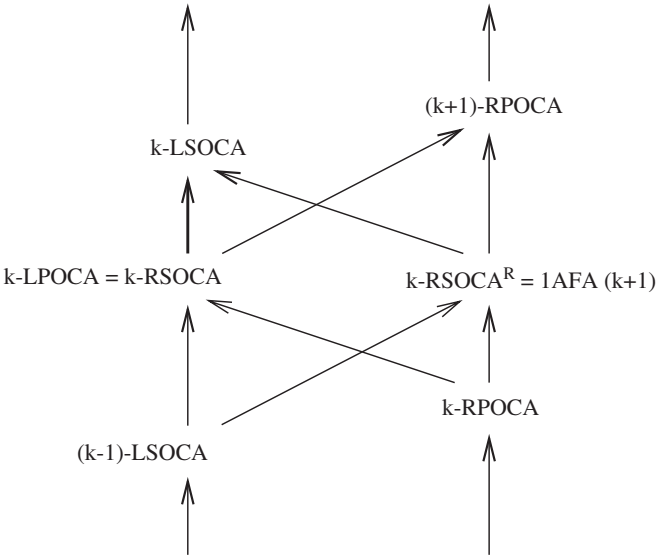


Fig. 4. Inside $\bigcup_k$ k-RSOCA.

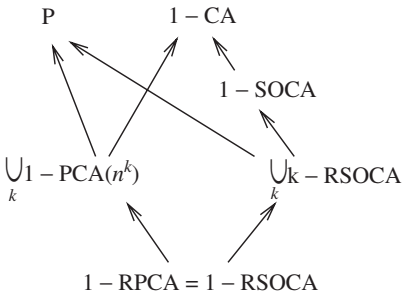


Fig. 5. Complexity classes around $\bigcup_k$ k-RSOCA.

And Fig. 5 shows the position of the class $\bigcup$k-RSOCA relatively to other complexity classes. 1-CA refers to the class defined by unrestricted time one-dimensional CA which is also the class defined by linear space bounded Turing machine. 1-SOCA refers to the class defined by unrestricted time one-way one-dimensional CA. And P refers to the class defined by polynomial time bounded Turing machine. Which of these inclusions are strict, is unknown. We just know that, at least one of these two inclusions $\bigcup$ 1-PCA$(n^k) \subseteq$ P and $\bigcup$ 1-PCA$(n^k) \subseteq$ 1-CA is strict, as P $\neq$ 1-CA, and also, at least one of these two inclusions $\bigcup$k-RSOCA $\subseteq$ 1-SOCA and $\bigcup$k-RSOCA $\subseteq$ P is strict, as 1-SOCA $\neq$ P. Besides, the equality between $\bigcup$ 1-PCA(n^k) and $\bigcup$k-RSOCA seems unlikely. Indeed, the known simulation of a 2-RSOCA by a 1-CA requires exponential time and the known simulation of a 1-PCA(n^2) on a k-SOCA requires more than linear time. That gives us some supplementary beliefs that, for one-dimensional space bounded CA, minimal time is less powerful than unrestricted time.

References

[1] J.H. Chang, O.H. Ibarra, M.A. Palis, Parallel parsing on a one-way array of finite state machines, IEEE Trans. Comput. C-36 (1) (1987) 64–75.
[2] J.H. Chang, O.H. Ibarra, M.A. Palis, Efficient simulations of simple models of parallel computation by time-bounded ATMs and space-bounded TMs, Theoret. Comput. Sci. 68 (1) (1989) 19–36.
[3] J.H. Chang, O.H. Ibarra, A. Vergis, On the power of one-way communication, J. ACM 35 (3) (1988) 697–726.
[4] C. Choffrut, K. Culik II, On real-time cellular automata and trellis automata, Acta Inform. 21 (4) (1984) 393–407.
[5] S.N. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machine, IEEE Trans. Comput. 18 (1969) 349–365.
[6] K. Culik II, Variations of the firing squad problem and applications, Inform. Process. Lett. 30 (3) (1989) 153–157.
[7] M. Delorme, J. Mazoyer, Cellular automata as languages recognizers, in: M. Delorme, J. Mazoyer (Eds.), Cellular Automata: a parallel model. Mathematics and its Applications, Kluwer, Dordrecht, 1999.
[8] P.C. Fischer, Generation of primes by one-dimensional real-time iterative array, J. ACM 12 (1965) 388–394.
[9] L.M. Goldschlager, A universal interconnection pattern for parallel computers, J. ACM 29 (4) (1982) 1073–1086.
[10] O.H. Ibarra, T. Jiang, On one-way cellular arrays, SIAM J. Comput. 16 (6) (1987) 1135–1154.
[11] O.H. Ibarra, T. Jiang, Relating the power of cellular arrays to their closure properties, Theoret. Comput. Sci. 57 (2–3) (1988) 225–238.
[12] O.H. Ibarra, S.M. Kim, S. Moran, Sequential machine characterizations of trellis and cellular automata and applications, SIAM J. Comput. 14 (2) (1985) 426–447.
[13] O.H. Ibarra, M.A. Palis, Two-dimensional iterative arrays: characterizations and applications, Theoret. Comput. Sci. 57 (1) (1988) 47–86.
[14] A. Ito, K. Inoue, I. Takanami, A relationship between one-dimensional bounded cellular acceptors and two-dimensional alternating finite automata, Informatik-Skripten 21 (1988) 60–76.
[15] K.N. King, Alternating multihead finite automata, Theoret. Comput. Sci. 61 (2–3) (1988) 149–174.
[16] H. Petersen, Alternation in simple devices, Lecture Notes in Computer Science, Vol. 944, Springer, Berlin, 1995, pp. 315–323.
[17] V. Terrier, On real time one-way cellular array, Theoret. Comput. Sci. 141 (1–2) (1995) 331–335.
[18] V. Terrier, Characterization of real time iterative array by alternating device, Theoret. Comput. Sci. 290 (3) (2003) 2075–2084.

Theoretical Computer Science 411 (2010) 266–276

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Simulation of one-way cellular automata by boolean circuits

Véronique Terrier

GREYC, Campus II, Université de Caen, 14032 Caen, France

ARTICLE INFO

Article history:
Received 8 October 2007
Received in revised form 28 September 2009
Accepted 10 October 2009
Communicated by B. Durand

Keywords:
Parallel computation
One-way cellular automata
Boolean circuit

ABSTRACT

We present a relationship between two major models of parallel computation: the one-way cellular automata and the boolean circuits. The starting point is the boolean circuit of small depth designed by Ladner and Fischer to simulate any rational transducer. We extend this construction to simulate one-way cellular automata by boolean circuits.

1. Introduction

The models of massively parallel computation are many and various. Among them, boolean circuits and cellular automata are famous. Simple links between these two models exist. Indeed the graph of communication dependencies of a cellular automaton is a directed acyclic graph. So a cellular automaton can be viewed as a particular type of boolean circuit. Conversely, a classic way to construct a universal cellular automaton (in dimension 2) consists in the simulation of a boolean circuit [7]. Paradoxically, although "natural" relationships exist between these two massively parallel models, the comparison of their computational abilities is not well known. Here we will investigate this subject.

In this paper, we will focus on one-dimensional cellular automata with the simplest communication pattern. On the line of cells, the information flow is either two-way or one-way according to the cells are connected to both its left and right neighbors or simply to its left neighbor. We differentiate two-way cellular automata (CA) when the information flow is two-way from one-way cellular automata (OCA) when the information moves in only one direction. Even if the information flow is restricted, the recognition ability of OCA has been stressed in several papers [5,3,4,2]. By instance, OCA can accept *P*Space-complete languages, it can simulate any alternating Turing machine bounded in linear time [5,3]. But the difference between one-way and two-way communication is not well understood. We do not know whether the inclusion between OCA and CA is strict or not. And even worse, we do not know whether OCA restricted to linear time is less powerful than unrestricted time CA (both bounded in linear space). However, OCA have characteristics that are not shared by CA. In particular, on an OCA, the states sequence of the cell c only depends on the initial state of the cell c and on the states sequence of its left cell $c-1$. The computation of this sequence is of a sequential nature. Precisely, the cell c acts as a rational transducer operating on the states sequence of its left cell $c-1$.

Here, we will exploit this "sequential" feature to simulate OCA by boolean circuits. The key starting result due to Ladner and Fischer is the simulation of any rational transducer by boolean circuits of small depth [6]. The circuit built as a succession of c such circuits, simulates an OCA working on c cells. The main task will be to evaluate the depth of this circuit.

This paper is organized as follows. Section 2 introduces basic notions. Section 3 recalls the circuit designed by Ladner and Fischer to simulate any rational transducer. Section 4 shows how to extend this construction to simulate any OCA and gives an empirical estimation of the depth complexity of the circuit. Section 5 justifies this estimation.

E-mail address: veronique.terrier@info.unicaen.fr.

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.10.005

2. Definitions

A *boolean circuit* with n input bits $\{x_1, \ldots, x_n\}$ is a directed acyclic graph with labeled nodes. Different types of nodes are distinguished. The nodes whose input degree is zero are either called *input gates* if they are labeled by an input value x_i or called *constant gates* if they are labeled by the constants 0 or 1. There are exactly n input gates, each with a different label x_i. The nodes whose input degree is not zero are referred as the *computation gates*, they are labeled by a boolean function (And, Or, Not). In addition, one gate is designated as the *output gate*. The size of a circuit is the number of its nodes. The depth of a gate is the length of a longest path connecting this gate to an input gate. The depth of a circuit is the depth of its output gate.

As model of computation, we have to consider family of circuits. A circuit family is a sequence $C = \{C_0, C_1, \ldots\}$ of boolean circuits where C_i is a circuit with i input variables. A circuit family decides a language L, if for every input w: $w \in L$ if and only if $C_{|w|}$ on input w outputs 1. The depth of the circuit family is a function d from $\mathbb{N}$ into $\mathbb{N}$ such that $d(n)$ is the depth of the circuit C_n. And its size is a function z from $\mathbb{N}$ into $\mathbb{N}$ such that $z(n)$ is the size of the circuit C_n.

A *one-way cellular automaton* (OCA) is a one-dimensional array of identical finite automata (cells) numbered $1, 2, \ldots$ from left to right, and working synchronously at discrete time steps. Each cell is only connected to its left neighbor and takes on a value from a finite set S, the *set of states*. At each step, the state of each cell is updated according to a *transition function* δ involving its own state and the state of its left neighbor. Formally denoting $\langle c, t\rangle$ the state of the cell c at time t, we have $\langle c, t\rangle = \delta(\langle c-1, t-1\rangle, \langle c, t-1\rangle)$. Because the first cell 1 has no left neighbor, we use a special state $\sharp$ not in S as a border state: $\langle 1, t\rangle = \delta(\sharp, \langle 1, t-1\rangle)$.

As language recognizer, we have to specify two subsets of S: the input alphabet Σ and the set of accepting states S_{accept}. The input mode is parallel. At initial time 1, the ith bit of the input word $w = x_1 \cdots x_n$ is fed to the cell i: $\langle i, 1\rangle = x_i$. The output cell where the result of the computation is displayed, is the cell numbered by $|w|$ the size of the input. That is the first cell which can get all the information of the input w. An OCA accepts a word w, if on input w the output cell enters an accepting state at some time t. Let f be a function from $\mathbb{N}$ into $\mathbb{N}$. An OCA accepts a language L in time f, if it accepts exactly the words $w \in L$ of length $|w|$ at time $t \leq f(|w|)$.

Let us emphasize how each cell c of an OCA (S, δ) behaves as a finite transducer. This finite transducer is specified in this way: the set of states, the input alphabet as well as the output alphabet is S, the transition function as well as the output function is δ and the initial state is $\langle c, 1\rangle$ the state of the cell c at initial time 1. On input word $\langle c-1, 1\rangle \cdots \langle c-1, t\rangle$, the cell c starting in state $\langle c, 1\rangle$, successively (enters in states and) outputs symbols $\langle c, 2\rangle \cdots \langle c, t+1\rangle$. Besides, we can interpret the partial runs of the cell using the maps δ_u defined as follows. For any $u = u_1 \cdots u_n \in S^*$, the map $\delta_u : S \to S$ is obtained by applying the transition function δ on the successive symbols of u: $\delta_u(s) = \delta(u_n, \delta(u_{n-1}, \ldots, \delta(u_2, \delta(u_1, s)) \cdots))$. Observe that the set of these maps $\mathcal{F} = \{\delta_u : u \in S^*\}$ is finite, as S is finite. And since $\delta_v \circ \delta_u = \delta_{uv}$, the set $\mathcal{F}$ with composition forms a finite monoid. Now, the behavior of the OCA (S, δ) can be expressed in terms of these maps δ_u. See Fig. 1. Let $u = \langle c-1, t_1\rangle\langle c-1, t_1+1\rangle \cdots \langle c-1, t_2-1\rangle$ be the states sequence of the cell $c-1$ at consecutive steps $t_1, t_1+1, \ldots, t_2-1$: we have $\langle c, t_2\rangle = \delta_u(\langle c, t_1\rangle)$.

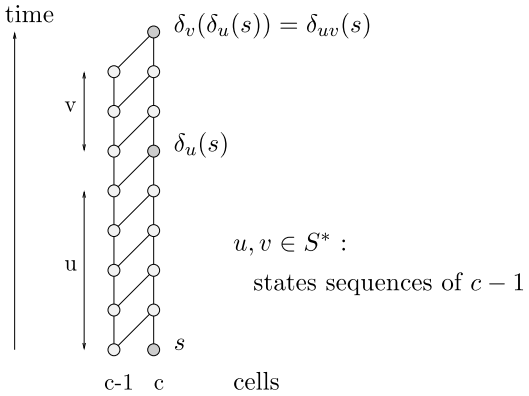


Fig. 1. The map δ_u.

As far as we know, only one work has investigated relationships between CA and boolean circuits [1]. Mainly, it is shown that a CA working in time t, unbounded in space, can be simulated by a circuit family of size t^2 and depth t. Moreover the circuit family is uniform: there is a logarithmic space Turing machine which generates a description of the nth circuit on input 1^n. Let us just recall the idea in the case of OCA although it is the same principle for CA with two-way communication.

Claim 1. *An OCA which works in time t can be simulated by a circuit family of depth $O(t)$.*

Proof. The sites (c, t) which represent the cells c at times t and their dependencies (i.e. the edges $((c-1, t-1), (c, t))$ and $((c, t-1), (c, t))$) constitute the communication graph of the OCA. This graph is a directed acyclic graph. So it is easy to transform the graph into a circuit as follows. Each transition $\langle c, t\rangle = \delta(\langle c-1, t-1\rangle, \langle c, t-1\rangle)$ performed on the site (c, t) can be carried out with a constant size boolean circuit where the inputs are the binary encodings of the states $\langle c-1, t-1\rangle$, $\langle c, t-1\rangle$ and the output is the binary encoding of the state $\langle c, t\rangle$. Moreover, as the communication graph is regular, the circuit family is uniform. □

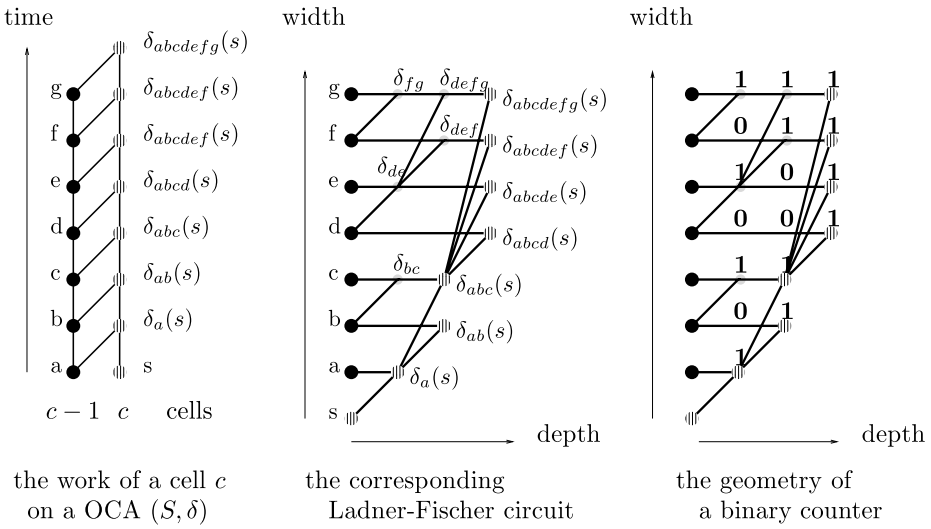


Fig. 2. The Ladner–Fischer circuit.

3. The Ladner–Fischer circuit

In [6], Ladner and Fischer have designed a small boolean circuit to simulate any finite state transducer. This circuit will be the key component to realize the simulation of OCA by boolean circuits. Precisely, this circuit can simulate the work of any one cell of an OCA.

Let us have a look at this Ladner–Fischer circuit. See Fig. 2. In the previous section, we have stressed that the behavior of an OCA (S, δ) can be expressed in terms of maps δ_u for $u \in S^*$. Precisely the cell c up to time t goes through the states $\langle c, w\rangle = \delta_{\langle c-1,1\rangle\cdots\langle c-1,w-1\rangle}(\langle c, 1\rangle)$ for all steps $w = 2, \ldots t$. Actually, to simulate the work of the cell c up to time t, the Ladner–Fischer circuit will compute the maps δ_u for all prefixes u of $\langle c-1, 1\rangle \cdots \langle c-1, t-1\rangle$. For that purpose, using a "divide and conquer" strategy, the circuit computes some intermediate values δ_v where v are subwords of $\langle c-1, 1\rangle \cdots \langle c-1, t-1\rangle$. In the Ladner–Fischer circuit, each computed value δ_v, where $v = \langle c-1, i\rangle \cdots \langle c-1, j\rangle$, is carried out at depth $\log(j+1-i)$ and width j. Notably, there exists at most one value computed at a given depth d and a given width w. Furthermore, if we mark every position (d, w) either by '1' or by '0' depending on whether the circuit computes or not a value at depth d and width w, then the circuit yields the geometry of a binary counter.

Concretely, the depth and the size of the boolean circuit are of the same order than the depth and the size of the dependency graph. Indeed, the nodes of the dependency graph computes either from a binary representation of a state $s \in S$ a binary representation of δ_s, either from two binary representations of maps δ_u and δ_v a binary representation of $\delta_v \circ \delta_u$ or from the binary representations of a map δ_u and a state s a binary representation of $\delta_u(s)$. As the set of states S and the set of maps $\mathcal{F} = \{\delta_u : u \in S^*\}$ are finite, these types of nodes can be carried out by constant size boolean circuits over {And, Or, Not}. Hence for any OCA, the calculation of n steps of any cell can be done by a boolean circuit of depth $O(\log(n))$.

As immediate consequence of the result of Ladner and Fischer, we get:

Claim 2. *An OCA with time complexity $t(n)$ can be simulated by a circuit family of depth $O(n \log(t(n)))$.*

Proof. An OCA which works in time $t(n)$ acts as a succession of n transducers operating on words of length at most $t(n)+n$. Thus n Ladner–Fischer circuits can be linked up to simulate the OCA. The depth of the resulting circuit is in $O(n \log(t(n)))$. □

4. The layered circuit

Now let us improve the above simulation in describing a circuit with smaller depth. Fig. 3 depicts such a circuit. It simulates the computations of the three first OCA cells on a given input. Their initial states $\langle 1, 1\rangle$, $\langle 2, 1\rangle$, $\langle 3, 1\rangle$ specified by this input, are available initially at depth 0; and, for convenience, they are set respectively at widths 0, -1, -2. This circuit can be broken down into three layers, each one simulating one cell.

In respect to the leftmost cell 1, recall that its left neighbor state is considered to be constantly the border state $\sharp$ and so is available initially. Depending on these border states and the initial state $\langle 1, 1\rangle$, the work of the cell 1 is simulated by the means of the Ladner–Fischer circuit. Concretely, we have a first layer which realizes a binary counter: at any given width w, the 1s and 0s which mark the existence or the absence of nodes for every depth, depict the binary writing of w. In addition, the most significant one at every width w, located on the right border, indexes the output $\langle 1, w+1\rangle$.

The computation of the second cell depends on its initial state $\langle 2, 1\rangle$ and the states sequence $(\langle 1, w\rangle)_w$, i.e., the outputs of the first layer. Note that we do not need to wait the last output of the first layer to begin the computation of the states sequence of the second cell. So the "divide and conquer" strategy is adapted in order to exploit the

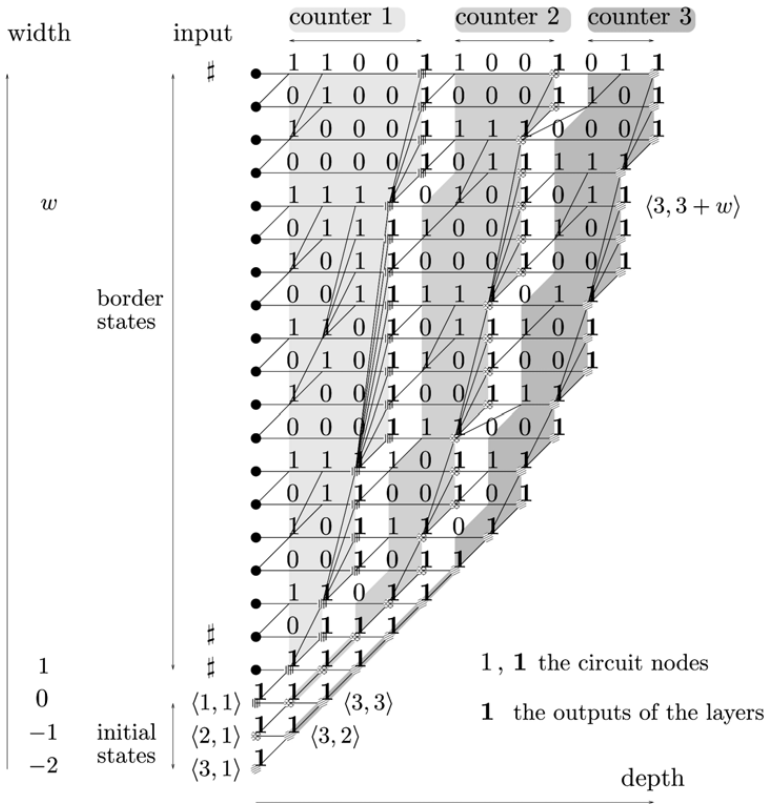


Fig. 3. The layered circuit.

outputs of the first layer as soon as available. Now, if we mark again every position (d, w) by a '1' or by a '0' depending on whether there exits a node at depth d and width w or not, we may again consider the values writing in binary on the successive widths. It is worth observing that these successive values correspond to a binary counter altered by the first layer. The counter is normally incremented by one except when the length of the first layer is increased; in this case, the counter passes the half of the current value plus one. Moreover the nodes marked by the rightmost ones compute the OCA states: the right border outputs at width w the state $\langle 2, w + 2\rangle$. Notice also that the depth grows exactly when the values are powers of 2.

The third layer shares the same design. More generally, the "divide and conquer" strategy makes that each layer behaves exactly like a binary counter altered when the previous one enters a power of 2.

Formally, we denote by $f(c, t)$ the value writing in binary on the counter (i.e., the layer) c at width t. We assume that the input node encoding the input value $\langle c, 1\rangle$ which is available at depth 0, is set at width $1 - c$ (i.e., negative widths are admitted). In this way, $f(c, t)$ can be expressed by recurrence as follows.

$$f(0, t) = 0 \text{ for } t \geq 1$$
$$f(c, 1 - c) = 1 \text{ for } c \geq 1$$
$$f(c, t) = \begin{cases} 1 + f(c, t-1) & \text{if } f(c-1, t) \text{ is not a power of 2} \\ 1 + \lfloor f(c, t-1)/2 \rfloor & \text{if } f(c-1, t) \text{ is a power of 2} \end{cases} \text{ for } c \geq 1,\ t > 1 - c.$$

The detailed circuit construction is rather technical and will be given in Appendix. In what follows, we just have to understand some characteristics of the circuit geometry. First, there exists at most one node at a given depth and a given width. Second, at a given width t, the word formed from the concatenation of the binary writings of the values $f(1, t), f(2, t), \ldots$, with the significant digits on the right side, depicts the existence or the absence of nodes: the existence of a node at depth d matches the occurrence of a one at the position d in the word. More specifically, the rightmost (and most significant) one in the binary writings of the counter value $f(c, t)$ matches the node which outputs the OCA cell $\langle c, t + c\rangle$. Notably, any circuit which simulates an OCA processing an input of size n in time $t(n)$, has its output node located on the right border of the counter n at width $t(n) - n$. So in order to evaluate the depth of the circuit, the task will be to determine the positions (i.e., the depths) taken by the rightmost one of the counters. As the circuit nodes are depicted by the successive words composed, for the successive widths t, of the values $f(1, t), \ldots, f(c, t), \ldots$, the rightmost one of any counter c has the following feature. Its position at a given width t remains the same as its position at the previous width except when $f(c, t)$ is a power of 2; in this case, its position moves one to the right.

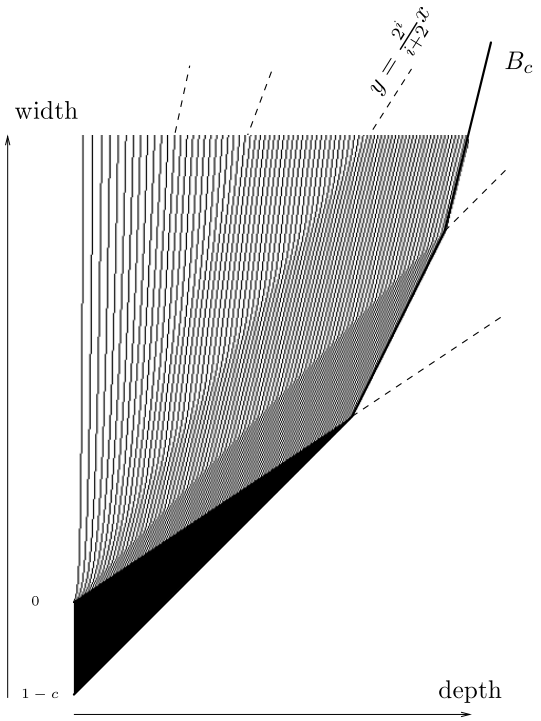


Fig. 4. A global point of view.

4.1. An empirical estimation of the depth of the circuit

First we present an empirical estimation of the depth from a global point of view. Fig. 4 outlines the circuit with a large number of counters. Only the rightmost nodes of the counters are marked. For each counter c, these nodes draw a "continuous" line which bounds the counter. We refer to this line as B_c. Moreover, we observe areas demarcated by straight lines D_i of equation $y = \frac{2^i}{i+2}x$. Except the noise around the lines D_i, the line B_c moves regularly: between D_i and D_{i+1}, it makes one move toward the depth while making 2^i moves toward the width. So empirically, the line B_c passes through the positions $\{(i+2)c, 2^i c) : i \in \mathbb{N}\}$. In particular, as the output node is located on the counter n at width $t(n) - n$, this node is on the line B_n at width $2^i n = t(n) - n$ and depth $(i+2)n = (\log(t(n)/n - 1) + 2)\, n$. Thus at a rough guess, the depth of the circuit family which simulates an OCA working in time $t(n)$ is in $O(n(1 + \log \frac{t(n)}{n}))$.

5. An upper bound of the depth of the circuit

Now we have to come back at the local point of view in order to justify the above estimation. In order to get an upper bound of the depth of the circuit, we have to specify the positions of the right borders of the counters c. Note that (the right border of) the counter c makes one move to the right every width i that $f(c, i)$ is a power of 2. In other words, the depth of the counter c is increased by 1 every width i that $f(c, i)$ is a power of 2:

$$depth(c, t) = |\{i : i \leq t \text{ and } f(c, i) \text{ is a power of } 2\}| .$$

But we have no explicit formula for $f(c, i)$ and then no exact expression for $depth(c, t)$. However we observe that the key values of the counters are the powers of 2. Indeed, not only the depth of the counter c is increased by 1 every width i that $f(c, i)$ is a power of 2, but also a counter is altered whenever the previous one enters a power of 2.

In practice to evaluate $depth(c, t)$, we will estimate the number of powers of 2 that the counter c enters below width t. To illustrate how these powers on the successive counters are interlinked, we will make use of diagrams with counters on the horizontal axis and width on the vertical axis which show at coordinates (c, t) whether $f(c, t)$ is a power of 2 or not (in Fig. 5, the black and grey squares depict the powers of 2). We will focus more specifically on the first occurrence of each distinct power of 2 on every counter (the black squares in Fig. 5).

In what follows, we will first observe that the sequence of powers of 2 on any counter c is increasing and that the width between two consecutive occurrences depends on their exponent. Then in order to estimate the number of powers of 2 with the same exponent, we will see how these numbers on two successive counters are correlated.

5.1. Local relationships of powers of 2

The following proposition emphasizes some properties relating two consecutive occurrences of powers of 2 on a counter.

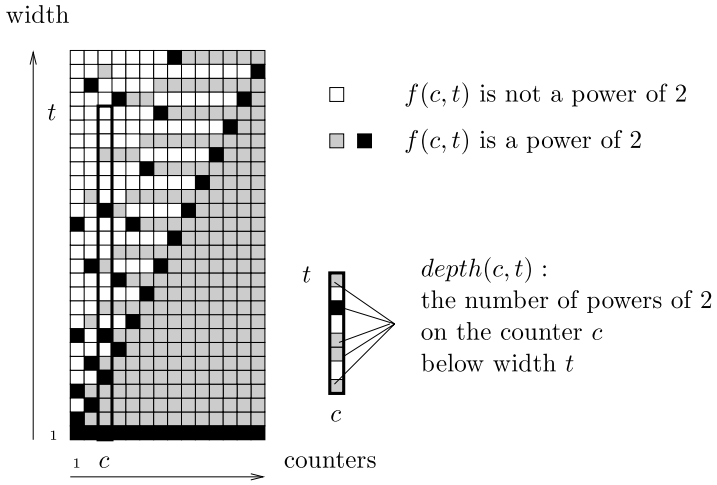
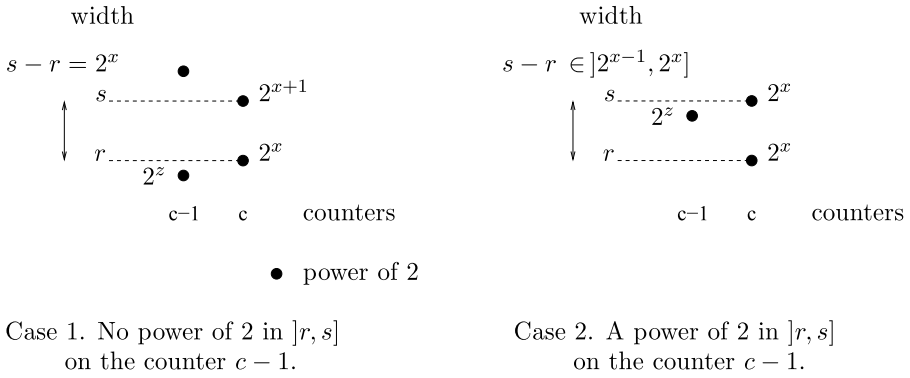


Fig. 5. The powers of 2.

Proposition 1. *On the counter $c \geq 1$, we consider two consecutive powers of 2: let widths r, s and exponents x, y be such that $r < s$, $f(c, r) = 2^x$, $f(c, s) = 2^y$ and $f(c, t)$ is not a power of 2 for all $t \in]r, s[$. We also consider, when $c > 1$, the power of 2 which occurs on the counter $c - 1$, just before width s: let width u and exponent z be such that $u \leq s$, $f(c-1, u) = 2^z$ and $f(c-1, t)$ is not a power of 2 for all $t \in]u, s[$.*
We have the following properties:

(P1) *On the counter c, the sequence of powers of 2 is increasing: y is either x or $x + 1$.*
(P2) *Between two consecutive powers of 2 on the counter c, there is zero or one power of 2 on the counter $c - 1$. And the distance between two consecutive powers of 2 is bounded:*
- *If $y = x + 1$ then $s - r = 2^x$ and $f(c-1, t)$ is not a power of 2 for all $t \in]r, s[$.*
- *If $y = x$ then $2^{x-1} < s - r \leq 2^x$ and there exists a unique $t \in]r, s]$ such that $f(c-1, t)$ is a power of 2.*

(P3) *The current exponent on the counter $c - 1$ is greater than or equal to the current exponent on the counter c: when $c > 1$, we have $z \geq y$.*



Case 1. No power of 2 in $]r, s]$ on the counter $c - 1$.

Case 2. A power of 2 in $]r, s]$ on the counter $c - 1$.

Fig. 6. Properties relating consecutive powers of 2.

Proof. See Fig. 6. The demonstration is done by recurrence on the counters c. It is true for the counter $c = 1$. Indeed $f(0, t) = 0$ and $f(1, t) = t$. So for each x, there is exactly one occurrence of 2^x at width 2^x and properties (P1) and (P2) are simply verified.
For the recurrence step, we distinguish two cases depending on whether there exists or not one power of 2 on the counter $c - 1$ between the two consecutive occurrences on the counter c.
Case 1. $u \leq r < s$ (no occurrence).
In this case, $f(c, s) = f(c, r) + s - r$. So $s - r = 2^x$, $y = x + 1$ and properties (P1) and (P2) are satisfied. Moreover, consider on the counter $c - 1$ the next width v where a power of 2 occurs after width u. We have $u \leq r < s < v$. So $v - u > s - r = 2^x$. Hence, due to the recurrence hypothesis, property (P2) ensures that $z \geq x + 1$. Thus property (P3) is satisfied.
Case 2. $r < u \leq s$ (at least one occurrence).
Let w and z be integers such that $f(c-1, w) = 2^z$ is the first power of 2 on the counter $c - 1$ after width r ($r < w \leq u \leq s$). First remark that $w - r \leq 2^x$. Otherwise the counter c enters a power of 2 before width s. Second observe that the exponent z is greater or equal than the current exponent on the counter c: $z \geq x$. Indeed, by the use of induction over the width, we know that the power of 2 which occurs on the counter $c - 1$ just before width r, has exponent at least x. Furthermore, by

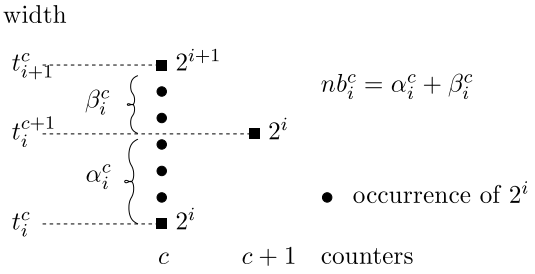


Fig. 7. Definitions of t_i^c, nb_i^c, α_i^c and β_i^c.

hypothesis of recurrence, the sequence of powers of 2 on $c-1$ is increasing. So $z \geq x$. Also using hypothesis of recurrence, we get that the next power of 2 on counter $c-1$ following $f(c-1, w) = 2^z$ occurs at width $w' \geq w+2^{z-1} \geq w+2^{x-1}$. Now note that the counter c is not altered by the counter $c-1$ until width w. So $f(c, w-1) = f(c, r)+w-1-r = 2^x+w-1-r$ and $f(c, w) = 2^{x-1} + \lfloor (w-1-r)/2 \rfloor + 1$. Moreover, as the next power of 2 on the counter $c-1$ does not occur before width $w+2^{x-1}$ which is strictly greater than $w+2^{x-1}-1-\lfloor (w-1-r)/2 \rfloor$, we have $f(c, s) = 2^x$ for $s = w+2^{x-1}-1-\lfloor (w-1-r)/2 \rfloor$. It follows that $s-r = w+2^{x-1}-1-\lfloor (w-1-r)/2 \rfloor - r = 2^{x-1}+\lceil (w-1-r)/2 \rceil \leq 2^x$. To sum up $y = x \leq z$ and $s-r \leq 2^x$. Hence the three properties are satisfied. □

5.2. At medium level

Now we will pay attention on the successive powers of 2 with the same exponent.

Definition 1. See Fig. 7. For any counters $c \geq 1$ and exponent $i \geq 0$, we denote by

- t_i^c the least integer t such that $f(c, t) = 2^i$;
- nb_i^c the number of integers t such that $f(c, t) = 2^i$;
- α_i^c the number of integers t such that $f(c, t) = 2^i$ and $t < t_i^{c+1}$;
- β_i^c the number of integers t such that $f(c, t) = 2^i$ and $t \geq t_i^{c+1}$.

We introduce also:

- $k(c, t)$ the current exponent of the counter c at width t, i.e., the exponent i such that $t_i^c \leq t < t_{i+1}^c$.

According to these notations, we have: $depth(c, t_k^c - 1) = \left|\{t : f(c, t) \text{ is a power of 2 and } t < t_k^c\}\right| = \sum_{i=0}^{k-1} \mathrm{nb}_i^c$. More generally, $depth(c, t) \leq \sum_{i=0}^{k(c,t)} \mathrm{nb}_i^c$. So, the strategy to get an asymptotic upper bound of $depth(c, t)$ is twofold. We present in Proposition 2, an asymptotic upper bound of the number of powers of 2 with exponent at most k: $\sum_{i=0}^{k} \mathrm{nb}_i^c \in O(ck)$; and in Proposition 3, an upper bound of the current exponent of the counter c at width t: $k(c, t) < 1 + \log(t/c)$. That would imply that $depth(c, t)$ is in $O(c(1 + \log(t/c)))$.

5.3. An upper bound of the number of powers of 2 with exponent at most k

In order to estimate the value of $\sum_{i=0}^{k} \mathrm{nb}_i^c$, we will take a closer look on the correlations between the powers of 2 on two successive counters and establish the following Facts 1 and 2.

Fact 1. $\mathrm{nb}_i^c = 1 + \beta_i^{c-1} + \alpha_{i+1}^{c-1}$.

Proof. See Fig. 8. By definition, nb_i^c is the number of times the counter c enters the value 2^i. According to property $(P1)$ of Proposition 1, the sequence of powers of 2 is increasing. So all of these values 2^i occur at widths in the interval $[t_i^c, t_{i+1}^c - 1]$. Furthermore, property $(P2)$ of Proposition 1 ensures, on the one hand, that between two consecutive occurrences of 2^i there is exactly one power of 2 on the counter $c-1$ and, on the other hand, that between the last occurrence of 2^i and the first occurrence of 2^{i+1} there is zero power of 2 on the counter $c-1$. It means that in the interval $[t_i^c, t_{i+1}^c - 1]$ the counter $c-1$ enters a power of 2 exactly $\mathrm{nb}_i^c - 1$ times. Thus by definitions of α_i^c and β_i^c, we get $\mathrm{nb}_i^c - 1 = \beta_i^{c-1} + \alpha_{i+1}^{c-1}$. □

Fact 2. $\alpha_k^c \leq k$

Proof. See Fig. 9. By definition, α_k^c is the number of times that the counter c has value 2^k before width t_k^{c+1}. Let $s_1, \ldots, s_a$ (with for short $a = \alpha_k^c$) the respective widths of these occurrences. Considering also the last $a+1$ occurrences of 2^{k-1} on the counter $c+1$, we denote by $r_1, \ldots, r_{a+1}$ their respective widths. Then $r_1 < s_1 \leq r_2 < s_2 \leq \cdots < s_a \leq r_{a+1}$, $f(c+1, r_i) = 2^{k-1}$, $f(c, s_i) = 2^k$ and $s_i - r_i \leq 2^{k-1}$. Now consider for $i = 1, \ldots, a$, the integers v_i defined by $s_i - r_i = 2^{k-1} + 1 - v_i$. Observe that $1 \leq v_i \leq 2^{k-1}$ whatever $i = 1, \ldots, a$. Then remark that, in the interval $[r_1 + 1, r_2]$, the only power of 2 on the counter

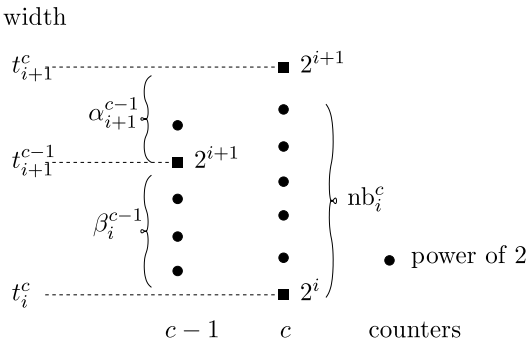


Fig. 8. $\mathrm{nb}_i^c = 1 + \beta_i^{c-1} + \alpha_{i+1}^{c-1}$.

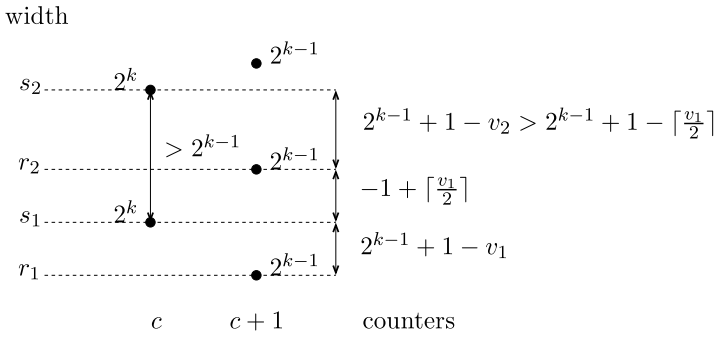


Fig. 9. An upper bound of α_k^c.

c is at width s_1. Thus $f(c+1, s_1-1) = f(c+1, r_1) + s_1 - r_1 - 1 = 2^{k-1} + s_1 - r_1 - 1 = 2^k - v_1$. Moreover $f(c+1, s_1) = 2^{k-1} - \lceil v_1/2 \rceil + 1$ and $r_2 - s_1 = f(c+1, r_2) - f(c+1, s_1) = \lceil v_1/2 \rceil - 1$. Yet $s_2 - s_1 = s_2 - r_2 + r_2 - s_1 = 2^{k-1} - v_2 + 1 + \lceil v_1/2 \rceil - 1$. Furthermore, $s_2 - s_1 > 2^{k-1}$ according to property ($P2$) of Proposition 1. It follows that $v_2 < \lceil v_1/2 \rceil \leq 2^{k-2}$. So inductively on $i > 1$, we can show that $v_i < 2^{k-i+1}$. To conclude, recall that $v_i \geq 1$ whatever $i = 1, \ldots, \alpha_k^c$. Thus $\alpha_k^c \leq k$. □

Let us notice that the bound $\alpha_k^c \leq k$ is not tight, experimentally α_k^c is either 1 or 2. Fortunately, this imprecision has no impact on the asymptotic upper bound of the number of powers of 2 given in the following proposition.

Proposition 2. $\sum_{i=0}^k \mathrm{nb}_i^c \in O(ck)$

Proof. By the use of the previous facts, we get:

$$\begin{array}{lll}
\sum_{i=1}^k \mathrm{nb}_i^c & = & k + \sum_{i=1}^k (\beta_i^{c-1} + \alpha_{i+1}^{c-1}) \qquad \text{as } \mathrm{nb}_i^c = 1 + \beta_i^{c-1} + \alpha_{i+1}^{c-1} \text{ according to Fact 1} \\
& = & k + \sum_{i=1}^k \mathrm{nb}_i^{c-1} + \alpha_{k+1}^{c-1} - \alpha_1^{c-1} \\
& = & k(c-1) + \sum_{i=1}^k \mathrm{nb}_i^1 + \sum_{r=1}^{c-1} (\alpha_{k+1}^r - \alpha_1^r) \\
& \leq & kc + \sum_{r=1}^{c-1} (\alpha_{k+1}^r - 1) \qquad \text{since } \mathrm{nb}_i^1 = 1 \text{ and } \alpha_1^r \geq 1 \\
& \leq & 2kc \qquad \text{as } \alpha_{k+1}^r \leq k+1 \text{ according to Fact 2.}
\end{array}$$

Finally a simple recurrence argument can be used to verify that $\mathrm{nb}_0^c = 2c - 2$ if $c > 1$ and $\mathrm{nb}_0^1 = 1$. □

5.4. An upper bound on the current exponent $k(c, t)$

To state an upper bound on the current exponent of the counter c at width t, we will need the following fact presenting a lower bound on the minimal width t_k^c when the counter value reaches 2^k.

Fact 3. $t_k^c \geq (c+1)2^{k-1}$

Proof. See Fig. 10. First observe that $t_k^c \geq t_k^{c-1} + 2^{k-1}$. Indeed, according to property ($P3$) of Proposition 1, the first occurrence of 2^k on the counter $c-1$ occurs before the first occurrence of 2^k on the counter c: $t_k^{c-1} \leq t_k^c$. And property ($P2$) of Proposition 1 ensures that, between these two occurrences, there is one occurrence of 2^{k-1} on the counter c at width w such that $t_k^{c-1} \leq w < t_k^c$ and $t_k^c - w = 2^{k-1}$. Thus $t_k^c - t_k^{c-1} \geq t_k^c - w = 2^{k-1}$ and $t_k^c \geq t_k^{c-1} + 2^{k-1}$. It follows that $t_k^c \geq t_k^1 + (c-1)2^{k-1}$. As $t_k^1 = 2^k$, we get $t_k^c \geq (c+1)2^{k-1}$. □

Proposition 3. *The current exponent of the counter c at width t verifies: $k(c, t) < 1 + \log(t/c)$.*

Proof. Simply note that $t \geq t_{k(c,t)}^c$. So from Fact 3, we obtain $t \geq (c+1)2^{k(c,t)-1}$. □

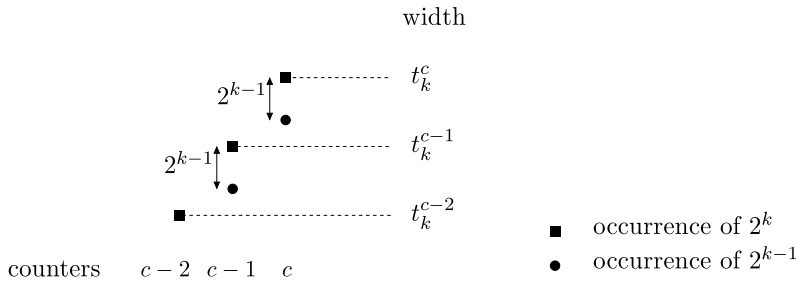


Fig. 10. An upper bound on the current exponent.

5.5. End of the proof

Putting Propositions 2 and 3 together provides the following asymptotic upper bound on the depth of the circuit.

Corollary 1. *depth*(c, t) *is in* $O(c(1 + \log(t/c)))$.

Now we may conclude.

Claim 3. *An OCA with time complexity* $t(n)$ *can be simulated by a circuit family of depth* $O(n(1 + \log \frac{t(n)}{n}))$.

Proof. Recall that the output node of the circuit is related to the binary counter n at width $t(n) - n$. It corresponds to the most significant one of the binary value $f(n, t(n) - n)$. So its depth is given by $depth(n, t(n) - n)$. Hence the depth of the circuit is in $O(n(1 + \log \frac{t(n)}{n}))$ according to Corollary 1. □

6. Conclusion

As corollary of a construction due to Ladner and Fischer, we have noted that an OCA working in bounded space n and in bounded time $t(n)$ can be simulated by a boolean circuit of depth in $O(n \log t(n))$. Then we have presented a better simulation of OCA by boolean circuits, the depth obtained is in $O(n \log \frac{t(n)}{n})$. It gives us a non-trivial relationship between OCA and boolean circuits in case of small complexities.

However further developments should be expected in the simulation of cellular automata by boolean circuits. Likewise, we may wonder how the NC complexity classes are related to the CA complexity classes. In brief, we would like to better understand the relationships of these two major models of massively parallel computation.

Appendix

We will give here the description of the circuit and the complexity of its construction.

A.1 Description of the circuit

To describe the circuit, we will state first the set of nodes, second the set of edges and finally the labels of the nodes.

The set of nodes and their position. The position of each node is specified by a couple (d, w) where d represents its depth and w its width (which may be negative). Furthermore, there exists at most one node at a given depth and a given width. At a given width t, the word formed from the concatenation of the binary writings of the values $f(1, t)$, $f(2, t)$, ..., (with the significant digits on the right side) codes the existence or the absence of nodes: the existence of a node at depth d matches the occurrence of a one at the position d in this word. Let us review the different types of nodes. The constant nodes denoted by $\gamma(0, t, 1)$ with $t \geq 1$ are set at positions $(0, t)$. The input nodes denoted by $\gamma(c, 1 - c, 1)$ with $c \geq 1$ are set at positions $(0, 1-c)$. Regarding the computation nodes, we will use the following notations. For $c \geq 1$ and $t \geq 1-c$, $\lambda(c, t)$, denotes the length of the binary writing of $f(c, t)$ and $\kappa(c, t)$ its number of ones. In other words, when $f(c, t)$ is of shape $2^{e_1} + \cdots + 2^{e_k}$ with $e_1 > \cdots > e_k$, $\lambda(c, t) = e_1 + 1$ and $\kappa(c, t) = k$. For convenience, we set $\lambda(c, t) = 0$ when $t < 1 - c$. Now, for $c \geq 1$, $t > 1 - c$ and $i = 1, \ldots, \kappa(c, t)$, $\gamma(c, t, i)$ refers to the computation node matching the ith one in the binary writing of $f(c, t)$. Its position is $\left(\sum_{r=1}^{c-1} \lambda(r, t) + 1 + e_i, t\right)$. In particular, $\gamma(c, t, 1)$ is the node on the right border of the counter c at width t. By extension, $\gamma(c, t, \kappa(c, t) + 1)$ denotes $\gamma(c - 1, t, 1)$.

The set of edges. The fan-in of the nodes is 2 and the fan-out is unbounded. Recall that the behavior of any OCA (S, δ) can be expressed in terms of the maps $\delta_u : s \in S \mapsto \delta(u, s) \in S$ for $u \in S^*$. Mainly, as in the design of Ladner–Fischer circuit, a node of the circuit computes a map δ_u by composition of two maps δ_x and δ_y get from two antecedents nodes and such that $u = xy$. Let us note how the nodes are linked in order to get the required information to perform the compositions. See Fig. 11. In the backward unrolling of the counter c, the ith one of the binary writing of $f(c, t)$ draws a trace from the width t until the width s such that $f(c, s)$ has less than i ones in its binary writing (i.e., $f(c, s)$ corresponds to $f(c, t)$ without the $\kappa(c, t) - i + 1$ less significant digits). Such value s will be denoted by $\omega(c, t, i)$. Formally, $\omega(c, t, i) = \max\{s : s \leq t \text{ and } \kappa(c, s) = i - 1\}$

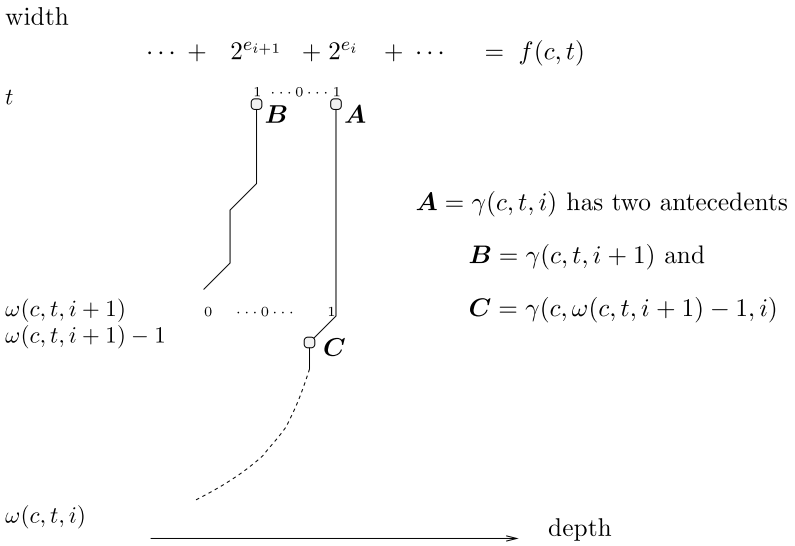


Fig. 11. The two antecedents of a node.

where $i = 2, \ldots \kappa(c,t)+1$. We also set $\omega(c,t,1) = 0$. We will see later that the map computed on the node $\gamma(c,t,i)$ is δ_u with $u = \langle c-1, c-1+\omega(c,t,i)\rangle \cdots \langle c-1, c-1+t\rangle$. But first let us specify the antecedents of the computation nodes. We distinguish two cases. If $c \geq 1$ and $1-c < t \leq 0$, the node $\gamma(c,t,1)$ has two antecedents $\gamma(c-1,t,1)$ and $\gamma(c,t-1,1)$ with indeed smaller depths. Second, if $c \geq 1$, $t \geq 1$ and $i = 1, \ldots, \kappa(c,t)$, the node $\gamma(c,t,i)$ has two antecedents $\gamma(c,t,i+1)$ and $\gamma(c,\omega(c,t,i+1)-1,i)$. Clearly, $\gamma(c,t,i+1)$ has a smaller depth than $\gamma(c,t,i)$. Regarding $\gamma(c,\omega(c,t,i+1)-1,i)$, its depth is exactly the depth of $\gamma(c,t,i)$ minus one. Indeed, the depth of the ith one is not altered while the counter has more than i ones in its binary writing. So the depth of $\gamma(c,t,i)$ is the same than the depth of $\gamma(c,\omega(c,t,i+1),i)$ and is one more than the depth of $\gamma(c,\omega(c,t,i+1)-1,i)$.

The label of the nodes. In order to proceed, we introduce the word $u(c,t,i)$ composed of the successive states of the cell $c-1$ between the steps $c-1+\omega(c,t,i)$ and $c-1+t$: $u(c,t,i) = \langle c-1, c-1+\omega(c,t,i)\rangle \cdots \langle c-1, c-1+t\rangle$. Furthermore, we verify the following fact.

Fact 4. $u(c,t,i) = u(c,\omega(c,t,i+1)-1,i)\,u(c,t,i+1)$.

Proof. First observe that $\omega(c,s,i) = \omega(c,t,i)$ for all s such that $\omega(c,t,i) \leq s \leq t$. Moreover $\omega(c,t,i+1)$ verifies $\omega(c,t,i) < \omega(c,t,i+1) \leq t$. Hence $\omega(c,\omega(c,t,i+1)-1,i) = \omega(c,t,i)$. It follows that $u(c,\omega(c,t,i+1)-1,i)u(c,t,i+1) = u(c,t,i)$. □

We label the nodes by the result of their computation which is either the state $\langle c, c+t\rangle$ for the nodes $\gamma(c,t,1)$ or the map $\delta_{u(c,t,i)}$ for the nodes $\gamma(c,t,i)$ when $i > 1$. Implicitly, the state label s may be also considered as the map label δ_s. Let us review the different types of labels. The constant nodes $\gamma(0,t,1)$ are labeled by the border state $\sharp$. The input nodes $\gamma(c,1-c,1)$ are labeled by the state $\langle c,1\rangle$ which is equal to x_c the cth bit of the input word. For the computation nodes, we distinguish three cases. First, in the case $c \geq 1$ and $1-c < t \leq 0$, the node $\gamma(c,t,1)$, from the labels $\langle c-1, c+t-1\rangle$ and $\langle c, c+t-1\rangle$ of its antecedents $\gamma(c-1,t,1)$ and $\gamma(c,t-1,1)$, computes the transition $\delta(\langle c-1, c+t-1\rangle, \langle c, c+t-1\rangle)$ and then outputs the state $\langle c, c+t\rangle$. Second, in the case $c \geq 1$, $t \geq 1$ and $i > 1$, the node $\gamma(c,t,i)$ from the labels $\delta_{u(c,t,i+1)}$ and $\delta_{u(c,\omega(c,t,i+1)-1,i)}$ of its antecedents $\gamma(c,t,i+1)$ and $\gamma(c,\omega(c,t,i+1)-1,i)$ computes the composition $\delta_{u(c,t,i+1)} \circ \delta_{u(c,\omega(c,t,i+1)-1,i)}$ whose result is the map $\delta_{u(c,t,i)}$ according to Fact 4. Finally, in the case $c \geq 1$, $t \geq 1$ and $i = 1$, from the labels $\delta_{u(c,t,2)}$ and $\langle c, c+\omega(c,t,2)-1\rangle$ of the antecedents $\gamma(c,t,2)$ and $\gamma(c,\omega(c,t,2)-1,1)$, the node $\gamma(c,t,1)$ computes $\delta_{u(c,t,2)}(\langle c, c+\omega(c,t,2)-1\rangle)$ whose result is $\langle c, c+t\rangle$ since $u(c,t,2) = \langle c-1, c-1+\omega(c,t,2)\rangle \cdots \langle c-1, c-1+t\rangle$. To conclude, observe that the three types of computations performed on the nodes can be carried out by constant size boolean circuits over {And, Or, Not}.

So we have designed a circuit $\mathscr{C}$ where every site $\langle c,t\rangle$ of the OCA is simulated on the node $\gamma(c,t-c,1)$. As a result, we get the following proposition.

Proposition 4. *An OCA working in time $t(n)$ can be simulated by the circuit family $C = \{C_1, C_2, \ldots\}$ with C_n is the subcircuit of $\mathscr{C}$ where the set of input nodes is $\{\gamma(c,1-c,1) : 1 \leq c \leq n\}$, the set of constant nodes is $\{\gamma(0,t,1) : 1 \leq t \leq t(n)-n\}$ and the set of computation nodes is $\{\gamma(c,t,i) : 1 \leq c \leq n, -c < t \leq t(n)-n$ and $1 \leq i \leq \kappa(c,t)\}$. The output node is $\gamma(n,t(n)-n,1)$. The depth of the circuit family C is of the same order as the depth of $\gamma(n,t(n)-n,1)$.*

Remark 1. This construction applies as well to simulate any OCA with sequential input mode (with such input mode, the ith bit of the input word $w = x_1 \cdots x_n$ is supplied to the cell 1 at time i). The only modification is to take as the set of input nodes $\{\gamma(0,t,1) : 1 \leq t \leq n\}$ and as the set of constant nodes $\{\gamma(c,1-c,1) : 1 \leq c \leq n\} \cup \{\gamma(0,t,1) : n < t \leq t(n)-n\}$. This construction works also to simulate any OCA which computes a function. In this case, the set of output nodes will include all the ones corresponding to the cells which communicate the output on the OCA.

A.2 Complexity of the circuit construction

To complete the presentation of this circuit family, it remains to evaluate the complexity of its construction. First let us describe how to set up the family of binary counters by the use of the following finite transducer. The set of states is $S = \{0, 1, \overline{\mathbf{1}}\}$ with 1 as initial state. Σ refers to the input alphabet as well to the output alphabet and is identical to S. The alphabet symbols code the current digits of the binary counter with the peculiar symbol $\overline{\mathbf{1}}$ used to mark the most significant digit of each counter. The states code the carry values ($\overline{\mathbf{1}}$ refers to the carry of the most significant digit). The transition function $\delta : S \times \Sigma \to S$ handles the carry propagation and the output function $o : S \times \Sigma \to \Sigma$ generates the current value. They are defined by the following tables.

$S \setminus \Sigma$	0	1	$\overline{\mathbf{1}}$
0	0	0	1
1	0	1	$\overline{\mathbf{1}}$
$\overline{\mathbf{1}}$	1	1	$\overline{\mathbf{1}}$

The transition function δ.

$S \setminus \Sigma$	0	1	$\overline{\mathbf{1}}$
0	0	1	$\overline{\mathbf{1}}$
1	1	0	0
$\overline{\mathbf{1}}$	$\overline{\mathbf{1}}$	$\overline{\mathbf{1}}$	$\overline{\mathbf{1}}$

The output function o.

Moreover, when the finite transducer ends in state $\overline{\mathbf{1}}$, it outputs $\overline{\mathbf{1}}$. In this way, whatever $c \geq t$ and $t \geq 1$, the transducer reading the sequence $f(1, t) \cdots f(c, t)$ writing in binary with the most significant digit of each $f(i, t)$ coded by $\overline{\mathbf{1}}$, outputs the sequence $f(1, t + 1) \cdots f(c, t + 1)$ coded in the same manner.

The following proposition states how "efficiently" the circuit family can be generated.

Proposition 5. *Let $t(n)$ be a time constructible function and $C = \{C_1, C_2, \ldots\}$ be the circuit family which simulates an OCA working in time $t(n)$. There exists a Turing machine which outputs a representation of C_n on input 1^n. The memory size is in $O(\log(z(n)) \times d(n))$ and the time is in $O(z(n) \times \log(z(n)))$ where $z(n)$ and $d(n)$ are the size and the depth of C_n.*

Proof. For $1 \leq c \leq n$ and $1 - c < t \leq 0$, it is straightforward to give a tuple representation of the nodes $\gamma(c, t, 1)$ and their antecedents. For $c \geq 1$ and $t > 0$, the Turing machine will simulate the finite transducer. The TM starts with the sequence $\overline{\mathbf{1}}^n$. Then it produces successively the sequences $f(1, t) \cdots f(n, t)$ writing in binary for t in range 2 to $t(n)$; remark that $t(n)$ is assumed to be time constructible. Furthermore, attached to the ith one of $f(c, t)$ which marks the node $\gamma(c, t, i)$, the TM records the numbering of the nodes $\gamma(c, t, i)$ and $\gamma(c, \omega(c, t, i) - 1, i)$. Notice that the TM is able to identify the node $\gamma(c, \omega(c, t, i) - 1, i)$ from the shape of $f(c, t)$. In this way, the TM can output a tuple representation of the node $\gamma(c, t, i)$ and its antecedents for each occurrence of one. If the size of the circuit is $z(n)$, the TM produces $z(n)$ tuples whose lengths are in $O(\log z(n))$. Moreover, at each step, the TM stores at most two successive sequences $f(1, t) \cdots f(c, t)$ and the numbering of the nodes attached to the occurrences of the ones. Thus the memory size is in $O(\log(z(n)) \times d(n))$ and the time is in $O(z(n) \times \log(z(n)))$. □

References

[1] H. Ben-Azza, Automates cellulaires et pavages vus comme des réseaux booléens, Ph.D. Thesis, Université de Lyon I, 1995.
[2] T. Buchholz, M. Kutrib, On time computability of functions in one-way cellular automata, Acta Informatica 35 (4) (1998) 329–352.
[3] J.H. Chang, O.H. Ibarra, A. Vergis, On the power of one-way communication, Journal of the ACM 35 (3) (1988) 697–726.
[4] O.H. Ibarra, T. Jiang, On one-way cellular arrays, SIAM Journal on Computing 16 (6) (1987) 1135–1154.
[5] O.H. Ibarra, M.A. Palis, S.M. Kim, Some results concerning linear iterative (systolic) arrays, Journal of Parallel and Distributed Computing 2 (1985) 182–218.
[6] R.E. Ladner, M.J. Fischer, Parallel prefix computation, Journal of the ACM 27 (4) (1980) 831–838.
[7] N. Ollinger, Universalities in cellular automata; a (short) survey, In: Proceedings of the First Symposium on Cellular Automata "Journées Automates Cellulaires" — JAC 2008, Uzès : France, 102–118 (2008).