Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

Separating Real-Time and Linear Space Recognition of Languages on One-Dimensional Cellular Automata

Victor Poupet

Feb 2006

Research Report N° 2006-10

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

Separating Real-Time and Linear Space Recognition of Languages on One-Dimensional Cellular Automata

Victor Poupet

Feb 2006

Abstract

In this article we will focus on a famous open question about algorithmic complexity classes on one dimensional cellular automata, and we will show that if all problems recognizable in space n (where n is the length of the input) can be recognized in time n then for every space-constructible function f all the problems that are recognizable in space f can be recognized in time f.

Keywords: Cellular automata, language recognition, real-time, linear space.

Résumé

Nous nous penchons, dans cet article, sur une question ouverte concernant les classes de complexité algorithmique sur automates cellulaires uni-dimensionels, et nous montrons que si tous les problèmes reconnaissables en espace linéaire sont reconnaissables en temps-réel alors pour toute fonction f constructible en espace, tout problème reconnaissable en espace f est reconnaissable en temps f.

Mots-clés: Automates cellulaires, reconnaissance de langages, temps-réel, espace linéaire.

1 Introduction

Cellular automata (CA) are a simple yet powerful and complex massively parallel computing model. It is known to be Turing-complete, but the parallelism of the computation make it quite different from usual sequential models (Turing machines, RAM machines etc.) in terms of algorithmic complexity. Many examples have shown how it can lead to very efficient computations [4, 5].

For these reasons it is interesting to investigate the properties of cellular automata as language recognizers, and a lot of work has been done specifically on the lower complexity classes: real-time (RT), linear time (LINTIME) and linear space (LINSPACE). It is obvious that

$$\text{RT} \subseteq \text{LINTIME} \subseteq \text{LINSPACE}$$

but whether or not these inclusions are strict is still unknown. Significant progress has been made, for example relating the equality of complexity classes to their closure properties [6] or working on weaker versions of CA [1, 2, 3].

In this article we will work on the two extremal classes RT and LINSPACE. We will show how the possible (although unlikely) equality implies the equality of a more general family of classes, namely that what is recognizable by a one-dimensional cellular automaton in space f is also recognizable in time f, for any space-constructible function f.

The article is structured as follows: in section 2 we give all necessary definitions and remind some basic and useful properties of language recognition by cellular automata. In section 3 we state the main theorem of the article and prove it, and in section 4 we discuss the main consequences of this theorem.

2 Definitions and Useful Properties

2.1 Cellular Automata

In this article, we will only consider one-dimensional cellular automata working on the classical neighborhood.

Definition 1 *A* cellular automaton *(CA) is a pair $\mathcal{A} = (\mathcal{Q}, \delta)$ where $\mathcal{Q}$ is a finite set called* set of states *containing a special state B, and $\delta : \mathcal{Q}^3 \to \mathcal{Q}$ is the* transition function *such that $\delta(B, B, B) = B$ (B is a* quiescent *state).*

For a given automaton $\mathcal{A}$, we call *configuration* of $\mathcal{A}$ any function $\mathcal{C} : \mathbb{Z} \to \mathcal{Q}$. From the local function δ we can define a global function

$$\Delta : \left\{ \begin{array}{ccc} \mathcal{Q}^{\mathbb{Z}} & \to & \mathcal{Q}^{\mathbb{Z}} \\ \mathcal{C} & \mapsto & \mathcal{C}' \end{array} \right. \quad | \quad \forall x \in \mathbb{Z}, \mathcal{C}'(x) = f(\mathcal{C}(x-1), \mathcal{C}(x), \mathcal{C}(x+1))$$

Elements of $\mathbb{Z}$ are called *cells*. Given a configuration $\mathcal{C}$, we'll say that a cell c is in state q if $\mathcal{C}(c) = q$.

If at time $t \in \mathbb{N}$ the CA is in a configuration $\mathcal{C}$, we'll say that at time $(t+1)$ it is in the configuration $\Delta(\mathcal{C})$. This enables us to define the *evolution* of a CA from a configuration. This evolution is completely determined by the initial configuration $\mathcal{C}$ and the automaton.

Definition 2 (Finite Configuration) *A configuration $\mathcal{C} : \mathbb{Z} \to \mathcal{Q}$ is said to be* finite *if there exists $x, y \in \mathbb{Z}$ $(x \leq y)$ such that for all $n \notin [\![x, y-1]\!]$, $\mathcal{C}(n) = B$.*

The size *of the configuration is the minimal value of $(y - x)$ for all (x, y) statisfying the definition.*

It is obvious (because the state B is quiescent) that for every finite configuration $\mathcal{C}$, $\Delta(\mathcal{C})$ is also finite. In this article we will only consider finite initial configurations so at all times the configuration of the automaton will be finite.

Definition 3 (Signals) *A* signal *in a CA is a special set of states that "moves" in a direction at a given speed. For example, a signal s moving at speed 1 to the right is a pair of states $\{s, \overline{s}\}$ such that $\delta(\overline{s}, s, \overline{s}) = \overline{s}$ and $\delta(s, \overline{s}, \overline{s}) = s$ (if a cell sees that its left neighbor is in state s, it becomes of state s at the next time, so in some sense the s state has moved right). In this example, $\overline{s}$ is a neutral state.*

To have a signal move at speed $1/k$ $(k \in \mathbb{Z}^+)$, we will use $k+1$ states $\{s_1, \ldots, s_k, \overline{s}\}$, and the rules

$$\begin{array}{rcll} \delta(\overline{s}, s_i, \overline{s}) & = & s_{i+1} & i \in [\![1, k-1]\!] \\ \delta(\overline{s}, s_k, \overline{s}) & = & \overline{s} & \\ \delta(s_k, \overline{s}, \overline{s}) & = & s_1 & \end{array}$$

so that the state s_i stays on a cell during k steps before moving to the right (again, all cells that don't hold the signal are in state $\overline{s}$).

Signals are a very useful tool for computations on cellular automata. Given an automaton $\mathcal{A} = (\mathcal{Q}, \delta)$, adding a signal $S = \{s_1, \ldots, s_k, \overline{s}\}$ corresponds to making a new cellular automaton whose states are $\mathcal{Q} \times S$. Any finite number of signals can be added to an automaton (while still having a finite number of states). Signals can then evolve according to their move rule (as explained in the definition) but can also interact the ones with the others when they meet on a cell (disappear, generate new signals etc.), or even change the main evolution of the automaton on a given cell when it receives a given signals (and that's what they are useful for).

Proposition 3 will give a good example of the use of signals.

Definition 4 (Space-Time Diagram) *Given a CA $\mathcal{A} = (\mathcal{Q}, \delta)$ and a configuration $\mathcal{C}$, we can represent by a two-dimensional diagram the complete evolution of the automaton from $\mathcal{C}$. The initial configuration is drawn horizontally on the bottom line and all the following configurations are drawn successively (time goes from bottom to top). Such a diagram is called* space-time diagram *of $\mathcal{A}$ from configuration $\mathcal{C}$.*

For obvious reasons, we only represent a finite part of a space-time diagram (finite in both space and time) however since we'll only consider finite configurations and computations over a finite time we will be able to represent all the necessary information.

Figure 1 shows part of a space-time diagram for a cellular automaton with two states. A specific neighborhood (white,black,white) has been highlighted to illustrate the fact that the rule of the automaton is completely deterministic (each time we have this neighborhood, the next state of the central cell is black because we have $\delta(w, b, w) = b$).

The space-time diagram of a CA is a discrete diagram. However, we will sometimes represent it as a continuous figure. In this case the signals will be represented as line segments,

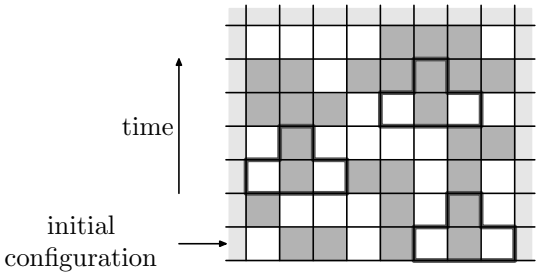

Figure 1: An example of a space-time diagram

and they'll partition the diagram in polygonal parts. This "continuous" representation can most of the times be seen as a limit of space-time diagrams when the size of the input increases, and when the size of the cells becomes negligible (and therefore, a continous space-time diagram is scale-free). Figure 2 illustrates such a process. It represents three space-time diagrams of a cellular automaton that marks the origin at time $3n/2$ when given an input word of length n with the use of three different signals (two that move at speed 1 from each end of the word the one towards the other to find the cell $n/2$ and the last one that moves at speed $1/2$ towards the origin to mark it at time $3n/2$). The two first diagrams are discrete representations of the behavior of the automaton on two words of different lengths, and the third one is the continous representation of the space-time diagram.

Figures 3 and 4 also represent continous space-time diagrams.

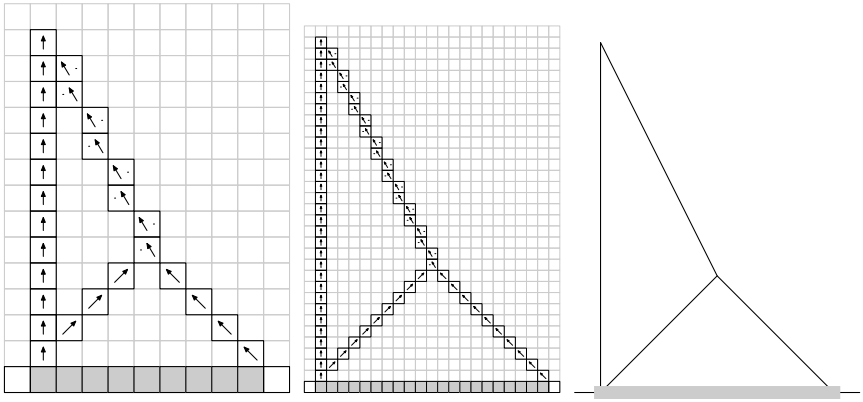

Figure 2: Discrete and continuous space-time diagrams

2.2 Language Recognition

Definition 5 (Word Recognition) *We consider a CA* $\mathcal{A} = (\mathcal{Q}, \delta)$ *and an* accepting state $q_f \in \mathcal{Q}$ *such that for all* $q_1, q_2 \in \mathcal{Q}$ *we have* $\delta(q_1, q_f, q_2) = q_f$ *(if a cell is in state* q_f *at one point, it stays in this state forever, such a state is called* persistent*). Let* $w = w_0 w_1 \dots w_{l-1}$

be a word on a finite alphabet $\Sigma \subseteq \mathcal{Q}$. We define the configuration $\mathcal{C}_w$ as follows.

$$\begin{array}{rccll} \mathcal{C}_w : & \mathbb{Z} & \to & \mathcal{Q} & \\ & \left\{ \begin{array}{lcll} x & \mapsto & w_x & \text{if } 0 \leq x < l \\ x & \mapsto & B & \text{otherwise} \end{array} \right. & & & \end{array}$$

We'll say that the CA $\mathcal{A}$ recognizes *the word w with accepting state q_f in time $t_w \in \mathbb{N}$ and space $s_w \in \mathbb{N}$ if, starting from the configuration $\mathcal{C}_w$ at time 0, the cell 0 is in state q_f at time t_w and no cell other than the ones in $[\![0, s_w - 1]\!]$ was ever in a state other than B.*

Definition 6 (Language Recognition) *Let $\mathcal{A} = (\mathcal{Q}, \delta)$ be a CA, $L \subseteq \Sigma^*$ a language on the alphabet $\Sigma \subseteq \mathcal{Q}$ and $q_f \in \mathcal{Q}$ a persistent state. Given two functions $T : \mathbb{N} \to \mathbb{N}$ and $S : \mathbb{N} \to \mathbb{N}$, we'll say that the language L is* recognized *by $\mathcal{A}$ in time T and space S with accepting state q_f if for every word $w \in \Sigma^*$ of length l, the CA $\mathcal{A}$ recognizes w with accepting state q_f in time $T(l)$ and space $S(l)$ if and only if $w \in L$.*

We will denote as CATIME(T) the class of languages recognizable in time T and as CASPACE(S) the class of languages recognizable in space S.

Definition 7 (Real-Time) *Let $\mathcal{A} = (\mathcal{Q}, \delta)$ be a CA and L a language on $\Sigma \subseteq \mathcal{Q}$. We'll say that the CA $\mathcal{A}$ recognizes L in* real-time *if it recognizes L in time $x \mapsto x$. The class of all languages that are recognizable by a CA in real-time will be denoted as* RT.

Definition 8 (Linear Space) *Let $\mathcal{A} = (\mathcal{Q}, \delta)$ be a CA and L a language on $\Sigma \subseteq \mathcal{Q}$. We'll say that the CA $\mathcal{A}$ recognizes L in* linear space *if it recognizes L in space $x \mapsto kx$ for some $k \in \mathbb{N}$. The class of all languages that are recognizable by a CA in linear space will be denoted as* LINSPACE.

2.3 Functions

Definition 9 (Time-Constructible Function) *Given a function $f : \mathbb{N} \to \mathbb{N}$, we'll say that f is* time-constructible *if there exists a CA $\mathcal{A} = (\mathcal{Q}, \delta)$ and two states $q_1, q_f \in \mathcal{Q}$ such that for every $n \in \mathbb{N}$, starting from the initial configuration $\mathcal{C}_{q_1^n}$ (the unary encoding of n into a finite configuration of $\mathcal{A}$) at time 0, the cell 0 is in state q_f for the first time at time $f(n)$.*

The above definition simply means that, given an integer n in unary form, the automaton can "count" $f(n)$ steps and mark the origin when it's done.

Definition 10 (Space-Constructible Function) *Given a function $f : \mathbb{N} \to \mathbb{N}$, we'll say that f is* space-constructible *if there exists a CA $\mathcal{A} = (\mathcal{Q}, \delta)$ and two states $q_1, q_f \in \mathcal{Q}$ such that for every $n \in \mathbb{N}$, starting from the initial configuration $\mathcal{C}_{q_1^n}$, after some time all cells ranging from 0 to $f(n) - 1$ switch to the state q_f, no cell was in this state before and during the whole computation none of the cells c such that $c < 0$ or $c \geq f(n)$ was ever in a state other than B.*

This definition means that, on input n (encoded in unary), the automaton will compute $f(n)$ in unary without using more cells in the process than the ones needed to write the output (the ones between 0 and $f(n) - 1$). The synchronization is not a problem because we can apply the firing squad technique at the end of the computation (proposition 1).

Remark. It is obvious, according to this definition, that any function f that is space-constructible is such that $f(x) \geq x$ for all x.

Remark. A function f is space-constructible according to the above definition if and only if there is a one semi-infinite tape, one head Turing machine that, on input n in unary, computes $f(n)$ in unary without ever moving its head past cell $f(n)$. This result is a consequence of proposition 5 (and proposition 1 for the synchronization part).

2.4 Basic Properties

Proposition 1 (Firing Squad) *It is possible to synchronize a segment of k adjacent cells (have them enter a specific state for the first time at the same time). It can be done in time $ak + b$ for any $a \geq 2$ and $b \geq 0$, starting from the time when the leftmost cell emits the "synchronization" signal. The rightmost cell need only be created at time $(a-1)k$.*

The firing squad problem has been well studied, and many ingenious solutions have been found. In this article we will need a solution that can be delayed so that the synchronization takes exactly ak steps for some integer $a \geq 2$ and such that the rightmost cell can be constructed as late as possible (at time $(a-1)k$). Such a solution is a particular case of the general "delayed" solutions explained in [7].

Proposition 2 (Grouping and Scaling) *Similarly to the acceleration theorems in the Turing case it is possible on cellular automata to "compress" the information of k cells into a single one (this requires the use of $|Q|^k$ states if the original automaton before compression had states Q). A finite configuration of length l is therefore compressed into a finite configuration of length $\lceil l/k \rceil$, and the computation can be accelerated by a multiplicative factor k (k steps of the original computation can be performed in only one step of the compressed computation).*

Using this grouping technique, if a function f is time-constructible, it is possible to create an automaton that, on input n compressed by a factor k, will mark the origin at time $\lceil f(n)/k \rceil$.

Proposition 3 *Let $f : \mathbb{N} \to \mathbb{N}$ be a time-constructible function. If for all $x \in \mathbb{N}$, we have $f(x) \geq 3x$ then the function $x \mapsto f(x) - x$ is also time-constructible.*

Proof. We use the construction illustrated in figure 3 where all dashed signals move at speed $1/2$.

The first step is to mark the cell $n/3$[1], this can be done at time $2n/3$ by sending a signal at speed $1/2$ to the right from the origin and a signal at speed 1 to the left from the rightmost letter of the word. Then, with a firing squad technique between the origin and the cell $n/3$ (see proposition 1) and a compression of the information it is possible to start the computation of $f(n)$ at scale $1/3$ from time n. This will mark the time $n + f(n)/3$ on the origin, and from this time, a signal s_1 appears and moves to the right at speed 1

Meanwhile, we mark the time $2n$ on the origin, and from there a signal s_2 starts going right at speed $1/2$. This signal meets the signal s_1 at time $2f(n)/3$ on cell $f(n)/3 - n$ (the signals meet only if s_2 was generated before s_1, which means that $2n \leq n + f(n)/3$, i.e. $3n \leq f(n)$), and from here a new signal starts moving towards the origin at speed 1, this last

[1] Obviously this point is in fact located on a cell of integer coordinates ($\lfloor n/3 \rfloor$) but has also the information n mod 3 so that the automaton has enough information to handle the point as if it were at rational coordinates. The same applies for all the other points that will be said to be at non-integer coordinates.

signal will arrive at the origin at time $2f(n)/3 + f(n)/3 - n = f(n) - n$.

□

Proposition 4 *Let $f : \mathbb{N} \to \mathbb{N}$ be a time-constructible function and $k \geq 2$ an integer. If for all $x \in \mathbb{N}$ we have $f(x) \geq 2kx$ then the function $x \mapsto f(x)/k$ is also time-constructible.*

Proof. We use the construction illustrated in figure 4 to construct $F : x \mapsto f(x)/k + x$ and then use proposition 3 to construct $x \mapsto F(x) - x = f(x)/k$.

This construction is simpler than the previous one, all we have to do is mark the cell $f(n)/k$ at time $(k-1)f(n)/k$ with the use of a signal moving to the right at speed $1/k$ and one going to the left at speed 1, and then with a firing squad technique start the computation of $f(n)$ at scale $1/k$ to mark the origin at time $f(n)/k + n$.

The combination of both constructions requires $f(x) \geq 2kx$.

□

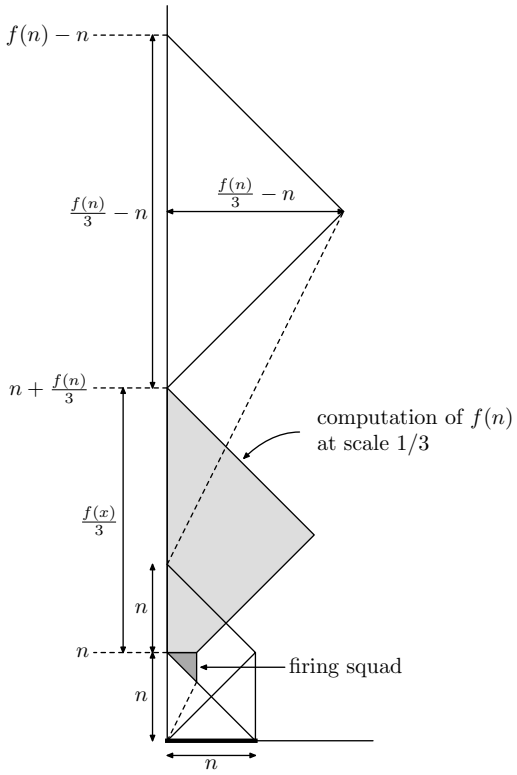

Figure 3: Time-constructibility of $x \mapsto f(x) - x$

Proposition 5 (Turing-Cellular Comparison) *Let P be a problem on Σ^*. If P is solvable in time $T : \mathbb{N} \to \mathbb{N}$ and space $S : \mathbb{N} \to \mathbb{N}$ by a Turing machine using one tape and one head, then it is recognizable in time T and space S by a cellular automaton.*

Moreover, if P is recognized by a cellular automaton in time T and space S then it is recognized by a one-tape Turing Machine in time $T \times S$ and space S.

Proof. The first part of the proposition is proved by simulating a Turing machine of states $\mathcal{Q}$ and tape alphabet Γ with a cellular automaton of states $\mathcal{Q} \times \{\Gamma \cup \{-\}\}$.

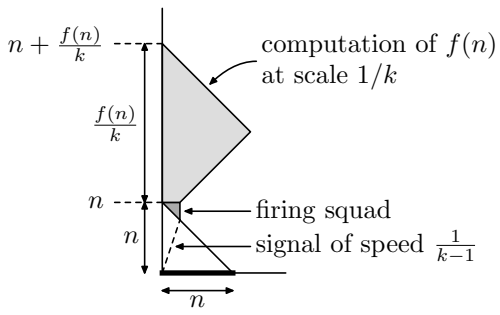

Figure 4: Time-constructibility of $x \mapsto f(x)/k + x$

A configuration of the Turing Machine

					q				
					$\downarrow$				
$\ldots$	B	a_1	a_2	$\ldots$	a_k	$\ldots$	a_n	B	$\ldots$

will be represented on the automaton as

	$-$	$-$	$-$		q		$-$	$-$	
$\ldots$	B	a_1	a_2	$\ldots$	a_k	$\ldots$	a_n	B	$\ldots$

and from there it is easy to make it so that the automaton doesn't change its states except in the immediate vincinity of the location of the head of the Turing machine, where it can apply the rule of the Turing machine.

It is immediate that this simulation doesn't require more space than the computation on the Turing machine, and that it takes exactly the same time.

The second part of the proposition is proved by a converse simulation. To simulate a cellular automaton of states $\mathcal{Q}$ by a Turing machine, we will use $\mathcal{Q}$ as tape alphabet and have the head of the Turing machine move across the whole configuration (we assume that all configurations are finite) and apply the transition rule of the automaton on each cell, then move across the configuration the other way and apply the rule of the automaton, etc. This simulation doesn't use more space than the computation on the CA, but the time needed to simulate one step of the automaton is the size of the current configuration, therefore if all configurations in the computation are of size bounded by S, the total time to simulate the computation is at most $T \times S$.

More details on these simulations can be found in [9].

□

Proposition 6 *For every problem P and every integer k, if P is solvable in space $x \mapsto kx$ then it is solvable in space $x \mapsto x$. In other words* LINSPACE = CASPACE($x \mapsto x$).

Proof. This proposition can be seen as a consequence of the previous proposition since it is well known that we have this result in the Turing case. It can also be proved in purely cellular terms the same way it is proved in the Turing case. Assuming we can solve a problem P in space $x \mapsto kx$ for a given $k \in \mathbb{N}$ with a CA of states $\mathcal{Q}$, we'll create a new CA of states Q^k and "fold" the configurations of length kn (where n is the size of the input) into configurations of length n as shown in figure 5.

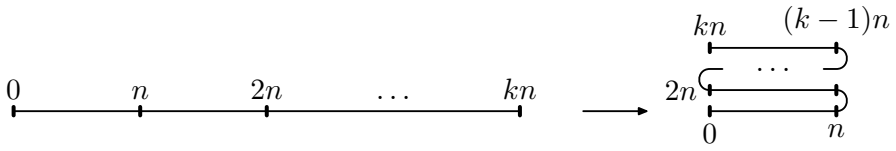

Figure 5: Folding a configuration of length kn into a configuration of length n

It is then easy to see how the initial configuration of the automaton translates into the folded version (the lower layer receives the letters of the input, all other layers have blank states) and check that the automaton can compute the other configurations by applying the rule of the initial one (all dependencies remain local).

□

Proposition 7 *For every problem P solvable in time T and space S on a cellular automaton, there is a CA that solves P in time $T' \leq T$, $S' \leq S$ and such that all cells $c < 0$ remain in the blank state B during all the computation (for every word w).*

Proof. This proposition can also be seen as a consequence of proposition 5 since it is known that what can be done on a bi-infinite tape Turing machine can also be done on a semi-infinite tape Turing machine. The method is the same in cellular terms, we create an automaton of states $\mathcal{Q}^2$ and consider that all configurations of the initial automaton are folded in two on the origin so that all states are on positive cells. The rule of the automaton can be applied on such a folded configuration, so there is no loss in time and all negative cells remain blank at all times.

□

3 Main Theorem

In this section, we will state and prove the main theorem of the article. Other consequences of this theorem will be discussed in the next section.

Theorem 1 (Main Theorem) *Given a space-constructible function f, if* RT = LINSPACE *then all problems that are solvable in space f are solvable in time f.*

To prove this, we will consider a space-constructible function f, a problem P over the alphabet Σ that can be recognized in space f and we will assume that RT = LINSPACE. We will then prove that P can be recognized in time f.

3.1 The problem $\widetilde{P}$

Lemma 1 *Under the hypothesis that* RT = LINSPACE*, every function f that is space-constructible is also time-constructible.*

Proof. Given a space-constructible function f, the problem

$$P_f = \{1^x \#^{f(x)-x} \mid x \in \mathbb{N}\}$$

is obviously recongnizable in space $f(x)$ (the size of the input): the machine reads x, computes $f(x)$ and compares it to the input word. If more space is needed then the word is rejected.

Because of the assumption that RT = LINSPACE P_f is solvable in real-time. From here it is easy to compute $f(x)$ in real-time: on input 1^x, the automaton simulates the recognition of P_f in real-time, assuming that all symbols after the rightmost "1" of the input are #. This computation is continued until the automaton accepts a word. When it does, it means that it has recognized the word $1^x\#^{f(x)-x}$ and it does so at time $f(x)$.

□

Lemma 2 *The problem* $\widetilde{P} = \{w\#^{f(|w|)-|w|} \mid w \in P\}$*, where # is a new symbol not in* Σ*, is recognizable in real-time.*

Proof. First we show that $\widetilde{P}$ is in LINSPACE. We will use two layers. On the first one we check that w is in P. This doesn't require more than $f(|w|)$ cells, so it can be done in LINSPACE. On the second layer, we compute $f(|w|)$ (which can be done in space $f(|w|)$ since the function f is space-constructible) and check that the number of # symbols after w matches the definition of $\widetilde{P}$. If at some point in the computation of either layer we need to use more cells than the ones of the input, it means that the input doesn't have enough # and so the word is rejected.

To conclude, since we have assumed that LINSPACE = RT, $\widetilde{P}$ is recognizable in real-time.

□

From now on, we will assume that we have a CA $\mathcal{A}$ that recognizes $\widetilde{P}$ in real-time, and we will explain how to construct a CA $\mathcal{A}'$ that, on input w of length n, simulates the behavior of $\mathcal{A}$ on input $w\#^{f(n)-n}$ without any loss of time so that $\mathcal{A}'$ recognizes P in time f.

3.2 The Space-Time Diagram of $\mathcal{A}$

Let's have a closer look at a typical space-time diagram of $\mathcal{A}$ on input $w\#^{f(n)-n}$ (figure 6).

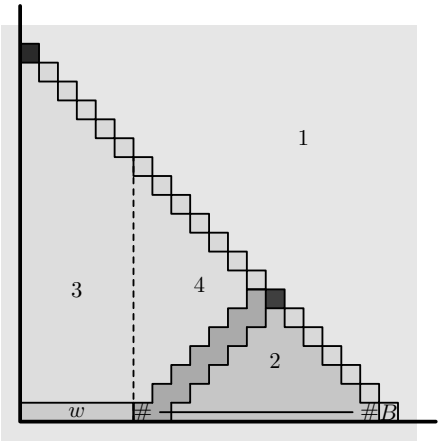

Figure 6: The Space-Time Diagram of $\mathcal{A}$ on input $w\#^{f(n)-n}$

3.2.1 Where There Is Little to Say.

The initial configuration (the bottom line) consists of the word w (of length n), followed by just enough # cells so that the total length is exactly $f(n)$. All other cells in this configuration (infinitely many to the right and to the left) are blank cells (in state B).

The only part that we'll be interested in is the area of the diagram that can affect the answer of the automaton located on the origin at time $f(n)$ (the dark cell on top of the figure). All cells outside of the triangle (area 1) have no importance. Moreover, we can assume that the cells on the left of the origin remain blank at all times (proposition 7).

The area of the diagram that is vertically over the word w (area 3) is very easy to simulate, provided we are able to compute correctly (by means we will discuss later) the vertical line that is on the right side of the dashed vertical line. Indeed, if we know at every step what is on the left of the segment (only blank cells) and on the right of it, we can apply the rule of $\mathcal{A}$ and compute the states in this area.

3.2.2 Problems Arise in the Right Triangle.

We have seen that it is not really necessary to simulate exactly all of the area in the triangle right of the previous zone (area 4) since only the vertical left border will be used for the rest of the computation.

To compute the states in this area, we'll need both the left and right borders. The left one should not be a problem since we have seen earlier how to compute it simultaneously with this one. The right one is a bit harder, and has two parts: the diagonal segment (stair-like in the figure) and the single point where the segment ends (darker).

The area number 2 (including the diagonal segment) is the set of all cells in the space-time diagram that are only affected by the # cells of the initial configuration (therefore it's a triangle on top of these cells). Because the rule of the automaton is homogeneous, this area is made of successive horizontal lines of identical states. The state on the bottom line is #, the state on the next line is $q_1 = \delta(\#, \#, \#)$, the state of the next is $q_2 = \delta(q_1, q_1, q_1)$ etc. Since the number of possible states is finite and that one state depends only on the previous one, this sequence is ultimately periodic (and only depends on the rule of the automaton, not on the input configuration). For this reason, the diagonal line that we've mentionned earlier is a well-known (assuming that the rule of $\mathcal{A}$ is known *a priori*) ultimately periodic sequence of states that will be easily created by $\mathcal{A}'$.

The only important information that cannot be obtained that way is the signal that comes from the first blank cell and that indicates the end of the red diagonal segment (the dark point). This point holds information about $f(n)$ that could be important for the computation of $\mathcal{A}$. Like before the state on this cell is easy to know (the possible states form another pre-determined ultimately periodic sequence) but the exact space-time location of the point is harder to know.

Let's define $g(n) = f(n) - n$. The dark point (that we'll name β is at space-time coordinates $(n + g(n)/2, g(n)/2)$. It is easy to prove, by properties of the maximal traveling speed of information, that this point cannot be constructed when starting from the initial configuration w unless g is bounded (which is not a very interesting case). We will now see how we can use a compression on the space-time diagram of $\mathcal{A}$ to solve this problem.

3.3 Compression of Area 4

3.3.1 Definition of the Compression.

For this section we'll change the coordinates on the space-time diagram of $\mathcal{A}$ so that the origin is now the leftmost # in the initial configuration (as shown in figure 7). The point β cannot be constructed by $\mathcal{A}'$, but since g is time-constructible if $f(n) \geq 3n$ (lemma 1 and proposition 3) the point $\gamma = (g(n)/3, 2g(n)/3)$ can be constructed if $g(n) \geq 6n$ (proposition 4, see figure 7 for the construction) meaning that $f(n) \geq 7n$. Let's define the transformation

$$\sigma : \begin{cases} \mathbb{N}^2 & \to & \mathbb{N}^2 \\ (x, y) & \mapsto & (\frac{2x}{3}, y + \frac{x}{3}) \end{cases}$$

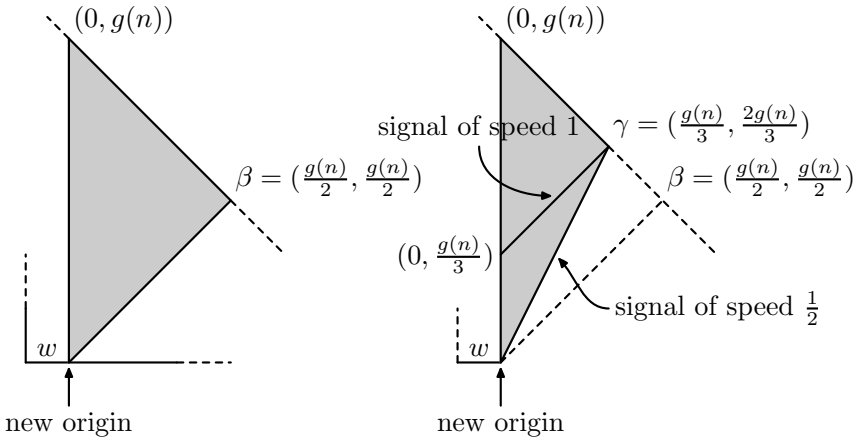

Figure 7: The compression of the triangle

The vertical axis is invariant by σ, $\sigma(\beta) = \gamma$ and the image of the diagonal segment (from figure 6) is a segment of slope 2. Figure 8 illustrates where all the information in the cells of the space-time diagram of $\mathcal{A}$ goes after compression by σ. We see that all cells receive one or two states from the diagram of $\mathcal{A}$ arranged in a very regular pattern (cells in columns of even index receive 1 state, cells in columns of odd index receive 2 states). In the figure, we've only represented what happens to the triangle that was area 4 in figure 6.

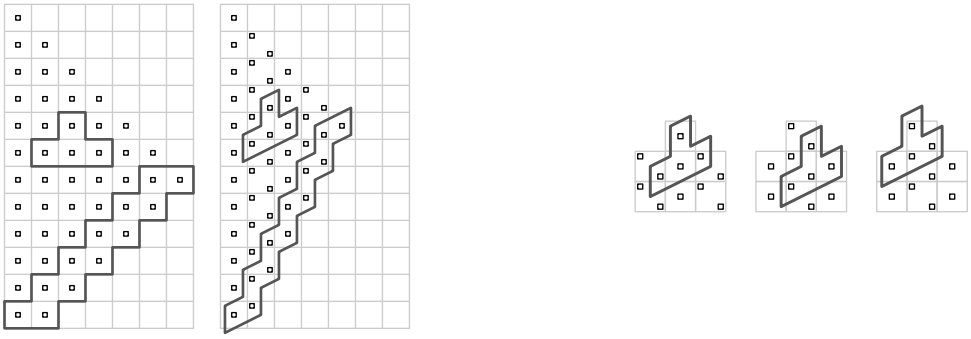

Figure 8: Effect of the compression on the information of each cell

Now we will see how the automaton can compute the compressed version of area 4, and therefore compute correctly all states on the vertical line (right side of the dashed line in figure 6) that are needed for the computation of area 3 (that leads to the solution of P).

3.3.2 How it works.

We will now assume (until further notice) that the function f is such that $f(n) \geq 7n$ for all n. With this hypothesis, the point γ can be marked on the space-time diagram by the automaton $\mathcal{A}'$.

Moreover the ultimately periodic sequence of states that were on the diagonal segment of figure 6 can be generated by $\mathcal{A}'$ and placed on the correct cells (the ones on the segment of slope 2, see figure 8). We have also assumed that $\mathcal{A}'$ knows the states that are next to the left side of the triangle, computed over the word w simultaneously with the triangle.

The right side of figure 8 illustrates how $\mathcal{A}'$ can compute all states inside the triangle. A point in the diagram of $\mathcal{A}$ can be in one of three different locations in the compressed triangle: alone in a cell (column of index $3k$), or with another state in which case it can be on the "top left" corner (column of index $3k + 1$) or the "bottom right" corner (column of index $3k + 2$) as illustrated in the left part of figure 8.

The first illustration shows how a cell that is "alone" can compute its new state since what it needs to see (drawn on the figure) is contained in its neighbors. The second illustration covers the case of a "lower right" point and is slightly more complicated because one of the states it needs (the leftmost one) is not really on its neighbor, but it was during the previous step. This means that the cells (at least the ones that have double information after the compression) need to memorize their old states when they change (they only need to know one generation in the past so they only need a finite memory). And finally the third illustration shows what information is needed to compute the new state on an "upper left" point, and we see that it needs to know the state of the "lower right" point that's in the same square, but that's not a problem since we've seen that it can be computed with what it sees (provided the square that's under it has memorized its older states).

We have therefore shown that the computation of the triangle after compression can be made by $\mathcal{A}'$. The compression doesn't modify the left side so it doesn't affect the rest of the computation. Thanks to the compression, we can now mark the point γ (that corresponds to the point β before compression) and so we can have all the necessary information to "close" the triangle and compute correctly the upper side, exactly like it's done by $\mathcal{A}$.

3.4 When f is Too Small

In the previous subsection, the construction of the point γ was essential, and it relied on the fact that $f(n) \geq 7n$. However it is not a problem if $f(x)$ is lower than $7x$ for some values of x. Indeed if a problem can be solved in space $x \mapsto 7x$ then it can be solved in space $x \mapsto x$ (proposition 6), therefore with our hypothesis that RT = LINSPACE it can be solved in real-time. So all we have to do is consider an automaton $\mathcal{A}_1$ that recognizes P in space f, and define the problem

$$P_< = \{w \in P \mid \text{the recognition of } w \text{ by } \mathcal{A}_1 \text{ uses less than } 7|w| \text{ cells}\}$$

It is obvious (by definition of space complexity) that for all x such that $f(x) \leq 7x$ and all words w of length x, $w \in P \Leftrightarrow w \in P_<$, and $P_<$ can be recognized in real-time.

All we have to do now is use both the automaton $\mathcal{A}'$ that we have defined earlier that will tell whether a word w is or not in P in time $f(|w|)$ if $f(|w|) \geq 7|w|$ and the automaton that recognizes $P_<$ in real-time that can answer for all other words.

We have therefore finished the proof of theorem 1 by constructing an automaton (the product of $\mathcal{A}'$ and the one that recognizes $P_<$ for the smaller values of f) that recognizes P in time f.

4 Consequences in the Turing Case

In this section we will discuss some consequences of theorem 1 concerning Turing models complexity. For a given function $f : \mathbb{N} \to \mathbb{N}$, we will denote as DTIME(f) the class of languages recognizable on a deterministic one-tape Turing machine in time f, and DSPACE(f) the class of languages recognizable on a deterministic one-tape Turing machine in space f.

Note that, as a consequence of proposition 5, DSPACE(f) = CASPACE(f). In the whole section, we will only consider deterministic one-tape Turing machines, that we will simply call Turing machines.

With the straightforward simulation of a Turing machine by a cellular automaton (proposition 5, and [9]), we can easily translate the result of theorem 1 in classical Turing terms.

If RT = LINSPACE then for every space-constructible function f (in the usual Turing sense since space complexities are the same for CA and Turing machines) every problem P that is solvable in space f is solvable in time f^2, in other words:

$$\text{DTIME}(f) \subseteq \text{DSPACE}(f) \subseteq \text{DTIME}(f^2)$$

Whether there exists or not a function f that contradicts these inclusions is still an open question. An important consequence of this is that P = PSPACE. Of course, the fact that RT = LINSPACE implies P = PSPACE is not new since it is an immediate consequence of the PSPACE-completeness of QSAT, that is known to be in LINSPACE, but the proof we have here doesn't require the existence of PSPACE-complete problems.

However, the most interesting consequence of the equality RT = LINSPACE is a very strong equivalence between the cellular and the Turing computing models obtained with the use of a theorem by Michael Paterson:

Theorem 2 (Paterson [8]) *For every function $f : \mathbb{N} \to \mathbb{N}$ such that $\forall x, f(x) \geq x$, we have* DTIME(f^2) $\subseteq$ DSPACE(f).

If RT = LINSPACE, then we already know that for every space-constructible function f,

$$\text{DSPACE}(f) = \text{CASPACE}(f) \subseteq \text{CATIME}(f) \subseteq \text{DTIME}(f^2)$$

and from the above theorem, we get

$$\text{CATIME}(f) = \text{DTIME}(f^2) = \text{DSPACE}(f) = \text{CASPACE}(f)$$

We strongly conjecture that these equalities aren't true in the general case, but again none of them has been proved false as of today.

5 Conclusion

In the article we have therefore shown how a result on two specific complexity classes (RT and LINSPACE) can be extended to a much more general family of classes (all CATIME(f) and CASPACE(f) for f space-constructible) and to a very strong equivalence between the computational complexity of one-dimensional cellular automata and one-tape Turing machines in time and space. Even though this is not sufficient to prove that the two initial classes are different, it strengthens the conviction that they are, and gives possible new ways to prove it.

References

[1] Choffrut, C., Čulik II, K.: On real-time cellular automata and trellis automata. Acta Informatica **21** (1984) 393–407

[2] Cole, S.N.: Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE Transactions on Computers **C-18** (1969) 349–365

[3] Čulik II, K., Gruska, J., Salomaa, A.: Systolic automata for vlsi on balanced trees. Acta Inf. **18** (1982) 335–344

[4] Čulik II, K.: Variations of the firing squad problem and applications. Inf. Process. Lett. **30** (1989) 153–157

[5] Fischer, P.C.: Generation of primes by one-dimensional real-time iterative array. Journal of the Assoc. Comput. Mach. **12** (1965) 388–394

[6] Ibarra, O., Jiang, I.: Relating the power of cellular arrays to their closure properties. Theoretical Computer Science **57** (1988) 225–238

[7] La Torre, S., Napoli, M., Parente, D.: Synchronization of a line of identical processors at a given time. Fundamenta Informaticae **34** (1998) 103–128

[8] Paterson, M.S.: Tape bounds for time bounded Turing machines. JCSS **6** (1972) 116–124

[9] Smith III, A.R.: Simple computation-universal cellular spaces. J. ACM **18** (1971) 339–353