A Padding Technique on Cellular Automata to Transfer Inclusions of Complexity Classes

Victor Poupet

LIP (UMR CNRS, ENS Lyon, INRIA, Univ. Claude Bernard Lyon 1),
École Normale Supérieure de Lyon,
46 allée d'Italie 69364 LYON cedex 07 FRANCE

Abstract. We will show how padding techniques can be applied on one-dimensional cellular automata by proving a transfer theorem on complexity classes (how one inclusion of classes implies others). Then we will discuss the consequences of this result, in particular when considering that all languages recognized in linear space can be recognized in linear time (whether or not this is true is still an open question), and see the implications on one-tape Turing machines.

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 makes 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 study cellular automata as language recognizers. Because of the parallelism it is very hard to prove non-trivial time lower bounds for the recognition of languages: there is currently no known language that can be recognized in linear space that cannot be recognized in linear time.

Also, some techniques that seem simple on sequential models such as Turing machines can be more complicated on cellular automata. In this article we will see how "padding techniques" can be used on cellular automata to transfer an inclusion between two complexity classes to more general classes. This technique has been widely known and used in the case of Turing machines, but it does not work immediately on cellular automata (since all cells work at each step, the cells in the "padded" region also work from the beginning).

The article is structured as follows: in section 2 we give the necessary definitions and recall some useful properties of cellular automata. Section 3 states the transfer theorem and proves it by explaining how to make padding work on one-dimensional cellular automata. Finally, in section 4 we study some consequences of this result, mainly concerning the long time open question of deciding whether or not there exist languages that can be recognized in linear space but not in linear time, and how it is linked to a tight equivalence between one-tape Turing machines and one-dimensional cellular automata.

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

$$\Delta : \begin{cases} \mathcal{Q}^{\mathbb{Z}} \to \mathcal{Q}^{\mathbb{Z}} \\ \quad \mathcal{C} \mapsto \mathcal{C}' \quad \mid \quad \forall x \in \mathbb{Z}, \mathcal{C}'(x) = \delta(\mathcal{C}(x-1), \mathcal{C}(x), \mathcal{C}(x+1)) \end{cases}$$

Elements of $\mathbb{Z}$ are called *cells*. Given a configuration $\mathcal{C}$, we will 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 will say that at time $(t+1)$ it is in the configuration $\Delta(\mathcal{C})$. This defines 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 $n \notin [\![x, y]\!]$, $\mathcal{C}(n) = B$.*

The size *of the configuration is the minimal value of $(y - x + 1)$ 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

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

so that the state s_i stays on a cell during k steps before moving to the right (again, all cells that do not 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 $S = \{s_1, \dots, s_k, \overline{s}\}$ 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 is what they are useful for).

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 will only consider finite configurations and computations over a finite time we will be able to represent all the necessary information.

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 (some specific states that move through the configuration) will be represented as line segments, and they will partition the diagram in polygonal parts.

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.*

$$\mathcal{C}_w : \quad \mathbb{Z} \to \mathcal{Q}$$
$$\begin{cases} x \mapsto w_x & \textit{if } 0 \le x < l \\ x \mapsto B & \textit{otherwise} \end{cases}$$

We will 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 will 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.

2.3 Functions

Definition 7 (Time-Constructible Function). *Given a function $f : \mathbb{N} \to \mathbb{N}$, we will 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 is done.

Definition 8 (Space-Constructible Function). *Given a function $f : \mathbb{N} \to \mathbb{N}$, we will 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 (see 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. Any time-constructible function f such that $\forall x, f(x) \geq x$ is space-constructible : we can modify the automaton so that it first compresses its input to a word of half its initial length, and then performs the computation of f using half the space (all states correspond to two states of the initial automaton). While this computation occurs, a signal moves at speed $1/2$ from the origin to the right. When the time $f(n)$ is computed on the origin, a new signal appears that moves to the right at speed 1. These two signals meet at the cell $f(n)$, and no more space has been used during the computation.

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. *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. The first step is to mark the cell $n/3$. 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 signal will arrive at the origin at time $2f(n)/3 + f(n)/3 - n = f(n) - n$.

Proposition 3. *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. 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 $f(n)/k + n$. We then use Proposition 2 to construct $x \mapsto f(x)/k$.

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

Proposition 4 (Turing-Cellular Comparison). *Let L be a language in Σ^*. If L is recognized 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 L 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. This property follows from straightforward simulations of one-tape Turing machines by cellular automata and *vice versa*. Details on these simulations can be found in [9].

Proposition 5. *For every language L and every integer k, if L can be recognized in space $x \mapsto kx$ then it can be recognized in space $x \mapsto x$. In other words $\bigcup_{k \in \mathbb{N}} \mathrm{CASPACE}(k.\,\mathrm{Id}) = \mathrm{CASPACE}(\mathrm{Id})$.*

The proof is identical to that of the same result in the Turing case.

Proposition 6. *For every language L recognizable in time T and space S on a cellular automaton, there is a CA that recognizes L in time $T' \leq T$ and space $S' \leq S$ and such that all cells $c < 0$ remain in the blank state B during all the computation (for every word w).*

The proof of this proposition is identical to the proof that semi-infinite tape Turing machines are equivalent to bi-infinite tape Turing machines.

3 Transfer Theorem

In this section, we will state and prove the transfer theorem. Some consequences of this theorem will be discussed in the next section.

Theorem 1. *Given two space-constructible functions $f, g : \mathbb{N} \to \mathbb{N}$ if $\mathrm{CASPACE}(f) \subseteq \mathrm{CATIME}(g)$ then for any time-constructible function $h : \mathbb{N} \to \mathbb{N}$ such that $\forall x, h(x) \geq 6x$ we have*

$$\mathrm{CASPACE}(f \circ h) \subseteq \mathrm{CATIME}(g \circ h)$$

To prove this, we will consider three functions f, g and h that satisfy the hypothesis of Theorem 1 and we will assume that

$$\mathrm{CASPACE}(f) \subseteq \mathrm{CATIME}(g)$$

Moreover we consider a language L over the alphabet Σ that can be recognized in space $f \circ h$. We will show that L can be recognized in time $g \circ h$.

3.1 The Language $\widetilde{L}$

Lemma 1. *The language $\widetilde{L} = \{w\#^{h(|w|)-|w|} \mid w \in L\}$, where # is a new symbol not in Σ, is recognizable in time g.*

Proof. We first show that $\widetilde{L}$ is in $\mathrm{CASPACE}(f)$. To do so we have to check that w is in L and that there are exactly the right amount of # symbols. Because L is in $\mathrm{CASPACE}(f \circ h)$ and h is space-constructible both verifications can be done (if more space is needed it means that there were not enough #). The conclusion follows from the hypothesis that $\mathrm{CASPACE}(f) \subseteq \mathrm{CATIME}(g)$.

From now on, we will assume that we have a CA $\mathcal{A}$ that recognizes $\widetilde{L}$ in time g. We will now construct a CA $\mathcal{A}'$ that recognizes L in time $g \circ h$.

3.2 Compression of the Space-Time Diagram

The aim of this section is to explain how it is possible to construct the cellular automaton $\mathcal{A}'$ that will, on input w, simulate the behavior of $\mathcal{A}$ on input $w\#^{h(|w|)-|w|}$ without any loss of time. The initial configuration of $\mathcal{A}$ is very simple, and the only information that $\mathcal{A}'$ is missing is the exact location of the cell $(h(|w|)-1)$.

Let us have a look at a typical space-time diagram of $\mathcal{A}$ on input $\omega = w\#^{h(|w|)-|w|}$ (see figure 1). According to proposition 6 we can consider that no computation takes place on the negative cells. We will now consider separately the area of the space-time diagram that is on top of the initial word w (the cells on which the letters were initially written) and the rest of the diagram, starting from the cell $|w|$ (these two areas are separated by a dashed line on the figure). If we assume that $\mathcal{A}'$ is able to compute correctly all the states on the column that is immediately to the right of this dashed line (all the states of the cell $|w|$), it will mean that it is also capable of computing all the states in the area corresponding to the cells from 0 to $(|w|-1)$ because the initial states are known (the letters of w) and that at each time the states of these cells only depend on the previous states of these cells and the cell $|w|$.

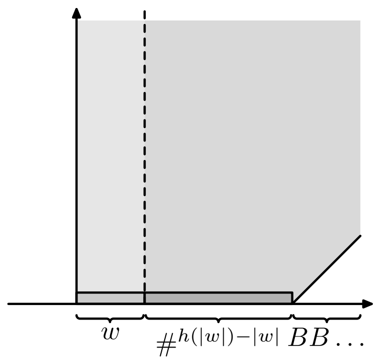

Fig. 1. Space-time diagram of $\mathcal{A}$.

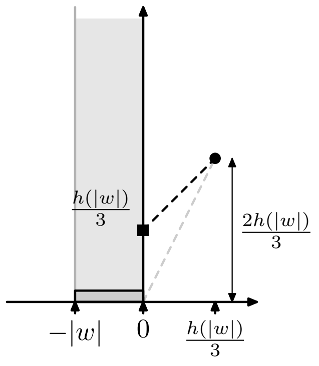

Fig. 2. Construction of the point $(h(|w|)/3, 2h(|w|)/3)$ at time $2h(|w|)/3$ (with shifted coordinates).

To compute the states on the cell $|w|$, we will operate a geometric transformation of the rightmost part of the space-time diagram (the part that is represented on the right of the dashed line) that will preserve this column and let the automaton $\mathcal{A}'$ construct the missing portion of the initial configuration of $\mathcal{A}$: the segment of $\#$ symbols of length $h(|w|)-|w|$.

Definition of the compression Let us shift the system of coordinates on the diagram so that the new origin is now the cell $|w|$ (the cell that was previously referred to as c is now the cell $(c - |w|)$). Let us consider the transformation

$$\sigma : \begin{cases} \mathbb{N}^2 \to (\frac{1}{3}\mathbb{N})^2 \\ (x, y) \mapsto (\frac{x}{3}, y + \frac{2x}{3}) \end{cases}$$

The vertical axis is invariant by σ and the image of the horizontal axis (the initial configuration) is now a line of slope 2. Figure 3 illustrates the effects of σ on the right part of the space-time diagram of $\mathcal{A}$. We see that each cell receives, after compression, the states of 3 cells before compression (except for the origin that keeps one single state). We have alse represented on the figure the neighborhood of a cell and its successor, both before and after compression.

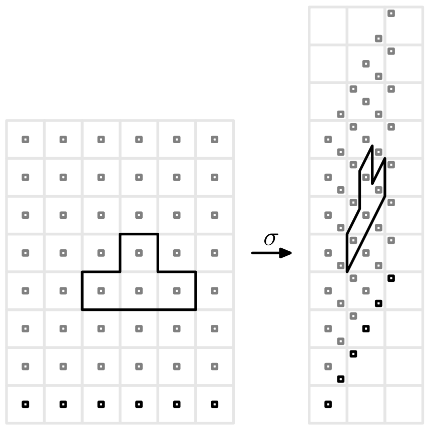

Fig. 3. Effects of σ on the right part of the space-time diagram.

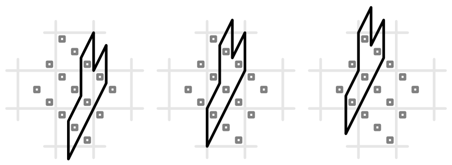

Fig. 4. The 3 possible cases when trying to apply the transition rule of $\mathcal{A}$ after compression by σ.

All we have to do now is show that $\mathcal{A}'$, on input w, can simulate the behavior of $\mathcal{A}$ by constructing in real-time the image by σ of the right part of the space-time diagram of $\mathcal{A}$.

How the Simulation Works Since we have assumed that for all $x \in \mathbb{N}$, $h(x) \geq 6x$, the function $h/3$ is time-constructible (proposition 3) which means that the origin (still with shifted coordinates) can be marked at time $h(|w|)/3$ which means we can mark the cell $h(|w|)/3$ at time $2h(|w|)/3$ by sending signals as illustrated on figure 2 (the black dashed line is a signal that moves at speed 1 whereas the grey dashed line is a signal moving at speed 1/2). The space-time point $(h(|w|)/3, 2h(|w|)/3)$ is exactly the image by σ of the point $(h(|w|), 0)$ that happens to be the last cell in state # in the initial configuration of $\mathcal{A}$.

Therefore, even if $\mathcal{A}'$ does not have all the information held by ω in its initial configuration, it can construct the image by σ of this initial configuration.

Indeed, by propagating a signal to the right at speed 1/2 from the shifted origin (the right border of the input word) it can write # symbols on the segment that is the image of the # segment in the initial configuration of $\mathcal{A}$ and we have just seen that $\mathcal{A}'$ is capable of deciding exactly where this segment must end.

We now have to check that $\mathcal{A}'$ can apply the transition rule of $\mathcal{A}$ to the information that is held by the cells after applying σ to the space-time diagram.

The states on the space-time diagram of $\mathcal{A}$ can be in one of 3 different situations after compression by σ as shown by figure 3: upper-left, center or lower-right. Figure 4 illustrates the situation for each of these cases.

In each part of the figure we have represented which states must be known in order to compute the next state on a given position. We see that the information needed to compute the "next state" on a lower-right position is located on the current cell and its right neighbor. To compute the next state on a central position, the cell must look at her right neighbor and also know the next state on the lower-right position, which, as we have seen, can be computed with the information that is accessible to the cell. Finally, computing a state in the upper-left position requires to see a state held by the left neighbor of the cell, and also computing the next state on the central position.

In all three cases, we have shown that if the space-time diagram of $\mathcal{A}'$ contains the image by σ of the configuration of $\mathcal{A}$ at time t then $\mathcal{A}'$ can compute the image by σ of the configuration of $\mathcal{A}$ at time $(t+1)$. Since the image of the initial configuration of $\mathcal{A}$ can be recreated by $\mathcal{A}'$, it is possible to compute correctly everything that happens on the right part of the space-time diagram of $\mathcal{A}$ (on the right of the last letter of the input word).

3.3 End of the Computation

We have seen how $\mathcal{A}'$ can simulate the computation of $\mathcal{A}$ on the area that is on top of the input word without any distortion if it can compute correctly all the states on the column $|w|$ (back on the original coordinates system), and how this column can be computed by compressing the space-time diagram of $\mathcal{A}$ (but the compression does not affect the column $|w|$).

The automaton $\mathcal{A}$ on input w can therefore simulate completely the behavior of $\mathcal{A}$ on input $\omega = w\#^{h(|w|)-|w|}$, which means that at time $g(|\omega|) = g \circ h(|w|)$ it can decide whether or not ω is in $\widetilde{L}$ and hence whether or not w is in L.

The language L is recognized in time $g \circ h$, which concludes the proof of theorem 1.

4 Linear Time, Linear Space

4.1 The Open Questions

As in the Turing case, it is very hard on cellular automata to prove lower bounds of complexity. Moreover, very little is known about comparing space and time complexities. If it is easy to show that for any function f, CATIME$(f) \subseteq$

CASPACE(f), showing that the inclusion is strict (or that it is an equality) is much harder.

In particular, extensive work has been made on the lower complexity classes:

- Real time: CATIME(Id);
- Linear time: $\bigcup_{k\in\mathbb{N}}$ CATIME(k. Id);
- Linear space: $\bigcup_{k\in\mathbb{N}}$ CASPACE(k. Id);

Space and time speed-up theorems show that the linear class time is equal to the class CATIME(2 Id) and that the linear time class is equal to CASPACE(Id). There is also an immediate sequence of inclusion between these three classes but it is still unknown if any of these inclusions is strict.

The questions have been first stated by A. R. Smith III in 1972 [10] and have remained open since. Some significant improvements have been made though, for example relating the equality of complexity classes to their closure properties [6] or working on weaker versions of cellular automata [1,2,3]. These questions are in many ways similar to the well known open question "P = PSPACE ?".

In this section we will use the theorem 1 to show some consequences of the assumptions that CATIME(Id) = CASPACE(Id) or CATIME(2 Id) = CASPACE(Id).

4.2 Stronger Version of the Main Theorem

We have the following results:

Lemma 2. *Under the hypothesis that* CATIME(Id) = CASPACE(Id)*, every function f that is space-constructible is also time-constructible.*

Proof. We show that the language $L_f = \{1^x\#^{f(x)-x} \mid x \in \mathbb{N}\}$ is in CASPACE(Id) and thus in CATIME(Id). Then we can make a cellular automaton that, on input 1^x will work as if all symbols on the right of the last '1′ were # and continue until the first (and only) word of the form $1^k\#^*$ is accepted (at time t the automaton considers that it has received all the relevant information, and therefore knows whether or not the word $1^x\#^{t-x}$ is in L_f). It will happen at time $f(x)$, so f is time-constructible.

Proposition 7. *If* CATIME(Id) = CASPACE(Id) *then for any space-constructible function $h : \mathbb{N} \to \mathbb{N}$ we have*

$$\text{CASPACE}(h) \subseteq \text{CATIME}(h)$$

Proof. The proof of this proposition is very similar to that of theorem 1. The only difference is that the hypothesis on h is now weaker because of lemma 2.

4.3 Equivalence of Cellular and Turing Models

For a given function $f : \mathbb{N} \to \mathbb{N}$, we will denote as $\mathrm{DTIME}(f)$ the class of languages recognizable on a deterministic one-tape Turing machine in time f, and $\mathrm{DSPACE}(f)$ the class of languages recognizable on a deterministic one-tape Turing machine in space f. It is known that $\mathrm{DSPACE}(f) = \mathrm{CASPACE}(f)$. Here we will only consider deterministic one-tape Turing machines, that we will simply call Turing machines.

If $\mathrm{CASPACE}(\mathrm{Id}) \subseteq \mathrm{CATIME}(2\,\mathrm{Id})$ then, because of Theorem 1 and Proposition 4, for every time-constructible function f every language L that is recognized in space f (in the usual Turing sense since space complexities are the same for CA and Turing machines) can be recognized in time f^2, in other words :

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

Whether there exists or not a function f that contradicts these inclusions is still an open question. An important consequence of this is that $\mathrm{P} = \mathrm{PSPACE}$. Of course, the fact that $\mathrm{CASPACE}(\mathrm{Id}) \subsetneq \mathrm{CATIME}(2\,\mathrm{Id})$ implies $\mathrm{P} = \mathrm{PSPACE}$ is not new since it is an immediate consequence of the PSPACE-completeness of TQBF, that is known to be in $\mathrm{CASPACE}(\mathrm{Id})$, but the proof we have here does not require the existence of PSPACE-complete problems.

Remark. For any function f such that $\forall x, f(x) \geq 2x$, we have $\mathrm{CATIME}(f) = \mathrm{CATIME}(2f)$ (because of a linear acceleration theorem on CA).

However, the most interesting consequence of the inclusion $\mathrm{CASPACE}(\mathrm{Id}) \subseteq \mathrm{CATIME}(2\,\mathrm{Id})$ is a very strong equivalence between the cellular and the Turing computing models obtained with the use of a theorem by M. Paterson:

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

If $\mathrm{CASPACE}(\mathrm{Id}) \subseteq \mathrm{CATIME}(2\,\mathrm{Id})$, then we already know that for every time-constructible function f,

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

and from the above theorem, we get

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

This would mean that any sequential algorithm (on a single-tape Turing machine) can be speed-up by a quadratic factor when considering it on a parallel model (cellular automaton). Since we know that this speed-up factor is optimal (what can be done in time f on a CA can be done in time f^2 on a TM), it would mean that the two models are very tightly equivalent in terms of time complexity, since knowing the exact complexity of a problem on one model would give the exact complexity on the other model.

Conversely, to show that $\mathrm{CASPACE}(\mathrm{Id}) \not\subseteq \mathrm{CATIME}(2\,\mathrm{Id})$, it would be enough to find a language that can be recognized in time f^2 (for some time-constructible function f such that $\forall x, f(x) \geq x$) on a one-tape Turing machine but not in time $2f$ on a cellular automaton. However, no such example is known as of today.

5 Conclusion

We have shown in this paper how padding techniques that are broadly known to work on Turing machines can also work on cellular automata, by transforming the space-time diagram so that the automaton has enough time during the computation to "recreate" the information that was held by the "padded" cells. Using this technique we have shown the cellular equivalent to the transfer theorem.

This theorem enabled us to establish an important link between problems of separation of space and time complexity classes and problems of parallel computation (if $\mathrm{CASPACE(Id)} \subseteq \mathrm{CATIME}(2\,\mathrm{Id})$ then every sequential algorithm can be parallelized with a quadratic speed-up).

Even though both questions remain open, the implications that we have seem to indicate that the inclusions do not hold, and open new possibilities to prove it.

References

1. Choffrut, C., Čulik II, K.: On real-time cellular automata and trellis automata. Acta Informatica **21** (1984) 393–407 pages 10
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 pages 10
3. Čulik II, K., Gruska, J., Salomaa, A.: Systolic automata for VLSI on balanced trees. Acta Inf. **18** (1982) 335–344 pages 10
4. Čulik II, K.: Variations of the firing squad problem and applications. Inf. Process. Lett. **30** (1989) 153–157 pages 1
5. Fischer, P.C.: Generation of primes by one-dimensional real-time iterative array. Journal of the Assoc. Comput. Mach. **12** (1965) 388–394 pages 1
6. Ibarra, O., Jiang, I.: Relating the power of cellular arrays to their closure properties. Theoretical Computer Science **57** (1988) 225–238 pages 10
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 pages 5
8. Paterson, M.S.: Tape bounds for time bounded Turing machines. JCSS **6** (1972) 116–124 pages 11
9. Smith III, A.R.: Simple computation-universal cellular spaces. J. ACM **18** (1971) 339–353 pages 5
10. Smith III, A.R.: Real-time language recognition by one-dimensional cellular automata. Journal of the Assoc. Comput. Mach. **6** (1972) 233–253 pages 10