A Time Hierarchy Theorem for Nondeterministic Cellular Automata

Chuzo Iwamoto[⋆], Harumasa Yoneda, Kenichi Morita, and Katsunobu Imai

Hiroshima University, Graduate School of Engineering
Higashi-Hiroshima, 739-8527 Japan
chuzo@hiroshima-u.ac.jp

Abstract. We present a tight time-hierarchy theorem for nondeterministic cellular automata by using a recursive padding argument. It is shown that, if $t_2(n)$ is a time-constructible function and $t_2(n)$ grows faster than $t_1(n+1)$, then there exists a language which can be accepted by a $t_2(n)$-time nondeterministic cellular automaton but not by any $t_1(n)$-time nondeterministic cellular automaton.

1 Introduction

One of the basic problems in complexity theory is to find the slightest enlarging of the complexity bound which allows new languages to be accepted. There is a huge amount of literature on time hierarchy theorems for various models of computation, such as Turing machines (TMs) [3,4,6,8,14,16], random access machines (RAMs) [2], parallel RAMs [7,13], and uniform circuit families [10].

In this paper, we investigate time-hierarchies of cellular automata (CA). The first result on CA-based hierarchies was given in [11]; it was shown that there is a language which can be accepted by a one-dimensional deterministic CA (1-DCA) in $t_2(n)$ time but not by any 1-DCA in $t_1(n)$ time. Here, $t_1(n)$ and $t_2(n)$ are arbitrary time-constructible functions such that $t_2(n)$ is not bounded by $O(t_1(n))$. (Time constructible functions on 1-DCA were also discussed in [11].)

Another result on CA-hierarchies is in the hyperbolic space [9]; it was shown that there is a language which can be accepted by two-dimensional hyperbolic CA (2-HCA) in $(t_2(n))^3$ time but not by any 2-HCA in $t_1(n)$ time. When $t_1(n) = n^r$, this hierarchy result can be improved as follows: For any rational constants $r \geq 1$ and $\epsilon > 0$, there is a language which can be accepted by an $n^{r+\epsilon}$-time 2-HCA but not by any n^r-time 2-HCA [12]. Interestingly, these time-hierarchy results in the hyperbolic space hold for both deterministic and nondeterministic cases.

On the other hand, no attempt has been made to present time-hierarchy results on one-dimensional nondeterministic CA (1-NCA). In this paper, it is shown that, if $t_2(n)$ is a time-constructible function and $t_2(n)$ grows faster than $t_1(n+1)$, then there exists a language which can be accepted by a $t_2(n)$-time 1-NCA but not by any $t_1(n)$-time 1-NCA.

⋆ Corresponding author. This research was supported in part by Scientific Research Grant, Ministry of Japan.

A lot of techniques have been known for separating complexity classes. For example, (i) the crossing-sequence argument for one-tape TMs [5], (ii) diagonal arguments for deterministic TMs [3,4,8], 1-DCA [11], PRAMs and DLOGTIME-uniform circuits [7], and (non)deterministic HCA [9], and (iii) padding arguments for nondeterministic TMs [6], alternating TMs and PRAMs [13], and (non)deterministic HCA [12]. In this paper, we will show a hierarchy theorem for one-dimensional NCA by using a recursive padding argument, which was firstly used in [1,19,20] for multi-tape nondeterministic TMs.

In Section 2, we give the definition of 1-NCA. The main result is also given in that section. The proof is given in Section 3.

2 Nondeterministic Cellular Automata

A cellular automaton (CA) is a synchronous parallel string acceptor, consisting of a semi-infinite one-dimensional array of identical finite-state automata, called *cells*, which are uniformly interconnected (see Fig. 1). Every cell operates synchronously at discrete time steps and changes its state depending on the previous states of itself and its neighbours.

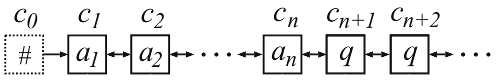

Fig. 1. Cellular automaton

A nondeterministic CA is a 7-tuple $M = (Q, \Sigma, \#, \delta, q, Q_A, Q_R)$, where

1. Q is the finite nonempty set of states,
2. $\Sigma \subseteq Q$ is the finite input alphabet,
3. # is the special boundary symbol not in Q,
4. $\delta : (Q \cup \{\#\}) \times Q \times Q \to 2^Q$ is the local transition function,
5. $q \in Q$ is the quiescent state such that $\delta(q, q, q) = \{q\}$, and if $\delta(p_1, p_2, p_3) \ni q$ then $(p_1, p_2, p_3) \in Q \times \{q\} \times \{q\}$ (i.e., any non-quiescent state does not become the quiescent state),
6. $Q_A \subseteq Q$ is the set of accepting states such that if $(p_1, p_2, p_3) \in \{\#\} \times Q_A \times Q$ then $\delta(p_1, p_2, p_3) \subseteq Q_A$,
7. $Q_R \subseteq Q$ is the set of rejecting states such that if $(p_1, p_2, p_3) \in \{\#\} \times Q_R \times Q$ then $\delta(p_1, p_2, p_3) \subseteq Q_R$.

The cell assigned to the integer $i \geq 1$ is denoted by c_i. The input string $a_1 a_2 \cdots a_n$ is applied to the array in parallel at step 0 by setting the states of cells $c_1, c_2, \ldots, c_n$ to $a_1, a_2, \ldots, a_n$, respectively. The remaining cells $c_{n+1}, c_{n+2}, \ldots$ are in the quiescent state. (Our CA is the so-called parallel-input model; however, the same hierarchy result holds for the sequential-input model, in which the input is fed serially to the leftmost cell.)

A configuration of a CA is represented by a string in $\{\#\}(Q - \{q\})^*$. A particular configuration is said to be *accepting* (resp. *rejecting*) if the first cell c_1 is in an accepting (resp. rejecting) state. The definition of a computation tree is mostly from [17]. For an input, computations of a nondeterministic CA are described as a tree T: All nodes are configurations, the root is the initial configuration of the CA for the given input, and the children of a configuration C are exactly those configurations reachable from C in one step allowed by the transition function. Leaves of T are final configurations, which may be accepting or rejecting. Certain paths in T may be infinite.

An interior node is defined to be accepting if at least one of its children is accepting. The CA accepts the input iff the root is accepting. A nondeterministic CA M is defined to be $t(n)$-*time bounded* if, for every accepted input w of length n, the computation tree T of M started with w stays accepting if it is pruned at depth $t(n)$. A CA is said to be *deterministic* if its transition function satisfies $|\delta(p_1, p_2, p_3)| = 1$ for every $(p_1, p_2, p_3) \in (Q \cup \{\#\}) \times Q \times Q$.

Let $M_1, M_2, \dots, M_r$ be CA with state sets $Q_1, Q_2, \dots, Q_r$, respectively. Consider a CA M such that the state set Q is a Cartesian product $Q = Q_1 \times Q_2 \times \cdots \times Q_r$ and each cell is partitioned into r *sub-cells*. The array of specific sub-cells in all cells is called a *track*. Then M can simulate $M_1, M_2, \dots, M_r$ in r tracks simultaneously.

The language accepted by a CA M is denoted by $L(M)$. A function $t(n)$ is said to be *time-constructible* if, for each n, there is a deterministic CA such that the first cell c_1 enters an accepting state at step $t(n)$ for the first time on all inputs of length n. It is known that if a function $t(n)$ is computable by an $O(t(n) - n)$-time TM (to which value n is given as a binary string of length $\lfloor \log_2 n \rfloor + 1$), then $t(n)$ is also time-constructible by a CA [11]. If two functions are time-constructible by CA, then the sum, product, and exponential functions of them are also time-constructible by CA. If $t(n)$ is time-constructible, then $t(n)$ is also *space-constructible*, i.e., there is a one-dimensional deterministic CA which places a "marker" at the $t(n)$th cell in $O(t(n))$ time (by emitting 1/2-speed and unit-speed pulses at steps 0 and $t(n)$, respectively).

Now we are ready to present our main theorem.

Theorem 1. *Suppose that* $t_2(n)$ *is an arbitrary time-constructible function and* $\lim_{n\to\infty} \frac{t_1(n+1)}{t_2(n)} = 0$. *There exists a language which can be accepted by a* $t_2(n)$*-time nondeterministic CA but not by any* $t_1(n)$*-time nondeterministic CA.*

Since $t_1(n+1) = O(t_1(n))$ when $t_1(n)$ is a polynomial in n, we can say, for example, n^r-time nondeterministic CA are stronger than $(n^r/\log\log n)$-time nondeterministic CA, where $r \geq 1$ is an arbitrary rational constant. However, it is not known whether $(t(n))^2$-time nondeterministic CA are stronger than $t(n)$-time nondeterministic CA when $t(n) = 2^{2^n}$.

3 Time Hierarchy for Nondeterministic CA

In this section, we will prove Theorem 1. In the rest of this paper, all CA are nondeterministic CA unless stated otherwise. We use the recursive padding method [20]. The outline of the proof is as follows: We first define a universal CA in Section 3.1. In Section 3.2, we construct a recursively padding CA which lengthens the input string as many times as we want. In Section 3.3, we observe the relationship between our recursively padding CA and its language. In Section 3.4, we assume for contradiction that any $t_2(n)$-time computation is sped-up to $t_1(n)$ time. Under this assumption, we will show that any recursive language can be accepted by $t_1(n)$-time CA (i.e., any computation for recursive languages (with no time restriction) can be sped-up to $t_1(n)$ time), a contradiction.

3.1 Universal CA

All languages in this section are over $\{0,1\}$. We first define the encoding rule of CA, which is essentially from [11]. We denote the states of CA by $q_1, q_2, \ldots, q_m$, where q_1 is the special boundary state #, and q_2 is the quiescent state q. For simplicity, we assume that q_3 (resp. q_4) is the unique accepting (resp. rejecting) state. State q_i is encoded into string 10^i of length $i+1$. We encode each transition rule $\delta(q_i, q_j, q_k) = \{q_{l_1}, q_{l_2}, \ldots, q_{l_r}\}$ into string $1110^i10^j10^k110^{l_1}10^{l_2}\cdots10^{l_r}$ for every $(q_i, q_j, q_k) \in (Q \cup \{\#\}) \times Q \times Q$. The encoding sequence e of a CA is the concatenation of all transition rules in the lexicographical order, called the *encoding part*, followed by $1111\cdots10$ of length $2^l - l$, called the *padding part*, where l is the length of the encoding part. Let M_e denote the CA whose encoding sequence is e.

Let $L_U = \{ex \in \{0,1\}^* |\ e$ is the encoding sequence of some CA M_e, and M_e accepts $x\}$. We construct a universal CA U accepting L_U such that U accepts ex in $c_e t(|x|)$ time if M_e accepts x in $t(|x|)$ time, where c_e is a constant depending only on e.

It is not difficult to verify whether the syntax of the encoding sequence e is proper in time proportional to $|e|$. Note that any polynomial in l is much smaller than $|e|$ because $|e| = 2^l$. In order to verify whether M_e accepts x, U simulates M_e on input x as follows. We denote the ith cell of M_e by s_i. Each cell s_i of M_e is simulated by $|e|$ cells of U. Namely, the cellular space of U is divided into *blocks*, $B_1, B_2, \ldots$, of the same length $|e|$, and the ith block B_i corresponds to M_e's cell s_i. Each block is divided into two tracks in order to store e and the state of s_i. Therefore, every block has all transition rules of M_e.

We can generate blocks of the same length as follows (see Fig. 2). The first and $|e|$th cells emit unit-speed and 1/2-speed pulses p_0, p_1 at step 0 to the right, respectively. Then the $|e|$th cell emits a 1/2-speed pulse p_2 at step $|e|$. Similarly, at steps $3|e|, 5|e|, 7|e|, \cdots$, cells at positions $2|e|, 3|e|, 4|e|, \cdots$ emit unit-speed pulses p_0. Markers are placed at positions where a pulse p_0 catches up with p_1. The encoding sequence e in a block can be copied into the next block in time proportional to $|e|$ (by using the firing squad synchronization (FSS) algorithm [15]). The ith symbol x_i of M_e's input x is moved to the ith block in $2|e| \cdot |x|$ time (the detail of this procedure is left to the reader).

After the above procedures, U starts to simulate M_e on input x. In order that every cell starts the simulation simultaneously, we use the FSS algorithm. Since each block has length $|e|$, a single step of M_e's cell s_i can be simulated by U's

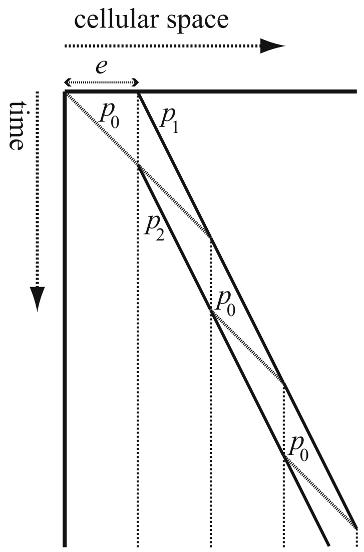

Fig. 2. Time-space diagram of CA. Markers are placed at regular intervals of length $|e|$.

block B_i in c'_e steps, where c'_e is a constant depending only on e. Therefore, there is a constant c_e such that U accepts ex in $c_e t(|x|)$ time if M_e accepts x in $t(|x|)$ time.

3.2 Recursive Padding

For a fixed encoding e, let $f(e)$ be the encoding of a CA $M_{f(e)}$ such that (i) $M_{f(e)}$ with the input string x changes x into ex and then (ii) $M_{f(e)}$ makes the computation of M_e on the input ex. (This definition is based on recursive function theory [18,21].)

For a fixed function h and a fixed CA M_1, let $g(h, M_1)$ be the encoding of a CA $M_{g(h,M_1)}$ which changes its input wx into $h(w)x$ and then makes the computation of M_1 on input $h(w)x$, where w is a valid encoding of some CA.

Now, consider CA $M_{f(g(f,M_1))}$ on input x. According to the definitions of f and g, (i) $M_{f(g(f,M_1))}$ on input x changes x into $g(f, M_1)x$, then (ii) $M_{f(g(f,M_1))}$ makes the computation of $M_{g(f,M_1)}$ on the input $g(f, M_1)x$ (i.e., $M_{g(f,M_1)}$ changes $g(f, M_1)x$ into $f(g(f, M_1))x$, and makes the computation of M_1 on input $f(g(f, M_1))x$). Therefore, $M_{f(g(f,M_1))}$ accepts x iff M_1 accepts $f(g(f, M_1))x$.

We analyse the relationship between the time complexities of $M_{f(g(f,M_1))}$ and M_1 as follows: We can convert the input x into $g(f, M_1)x$ and then into $f(g(f, M_1))x$ in time proportional to $|f(g(f, M_1))x|$. In order to start the computation of M_1 in every cell simultaneously, $M_{f(g(f,M_1))}$ performs the FSS algorithm, which can also be done in linear time. Hence, if M_1 accepts $f(g(f, M_1))x$ in $t(|f(g(f, M_1))x|)$ time, then $M_{f(g(f,M_1))}$ accepts x in $ct(|f(g(f, M_1))x|)$ time for some constant c.

Let $L \subseteq \{1\}^*$ be any recursive language, and let M be the deterministic CA which accepts L in $t(n)$ time. Now, we define (nondeterministic) CA M' which recursively pads the input string until the length becomes larger than $t(|x|)$.

The CA M' first verifies whether the input string is of the form $ex0^k$, where $x \in \{1\}^*$ and e is the encoding sequence of some CA M_e. So, M' verifies whether the input is an encoding sequence of a CA followed by an arbitrary number of 1's, which are further followed by an arbitrary number of 0's. Then, CA M' compares the values of $t(|x|)$ and $|x0^k|$ by (i) emitting a unit-speed pulse to the left from the rightmost 0 in $ex0^k$ and (ii) making the deterministic $t(|x|)$-step computation of M on x (see Fig. 3). If the computation of M halts before the pulse reaches the position of the first symbol of $x0^k$ (i.e., $t(|x|) < |x0^k|$), then M' halts with an accepting state iff M accepts x. If $t(|x|) \geq |x0^k|$, then M' pads $ex0^k$ to $ex0^{k'}$, where $k' > k$ is a nondeterministically chosen integer; M' performs the FSS algorithm in order to start the computation of the universal CA U on input $ex0^{k'}$.

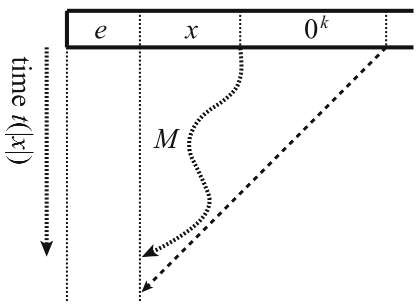

Fig. 3. M' makes the computation of M on x, and emits a unit-speed pulse

In the following, we analyse the properties of the above padding procedure. Consider the computation of M' on input $f(g(f,M'))x$. According to the definition, M' pads $f(g(f,M'))x$ to $f(g(f,M'))x0^k$ for nondeterministically chosen $k > 0$, and starts the computation of U on $f(g(f,M'))x0^k$. Since U is a universal CA, the computation of U on $f(g(f,M'))x0^k$ implies the computation of $M_{f(g(f,M'))}$ on input $x0^k$. According to the definition, $M_{f(g(f,M'))}$ first changes the input $x0^k$ to $g(f,M')x0^k$. The next task is to change $g(f,M')x0^k$ into $f(g(f,M'))x0^k$ and then to make the computation of M' on input $f(g(f,M'))x0^k$. According to the definition, M' compares the values of $t(|x|)$ and $|x0^k|$. If $t(|x|) < |x0^k|$, then M' halts with an accepting state iff M accepts x. If $|x0^k|$ has not yet been larger than $t(|x|)$, M' pads $x0^k$ to $x0^{k'}$ for nondeterministically chosen $k' > k$, and restarts the computation of U on $f(g(f,M'))x0^{k'}$. The CA M' recursively performs this procedure until the padding sequence becomes so long that $t(|x|) < |x0^{k'}|$.

3.3 Recursively Padding CA and Their Languages

Recall that $L \subseteq \{1\}^*$ is any recursive language, and M is the deterministic CA recognizing L in $t(n)$ time. We will prove

$$L(M_{f(g(f,M'))}) = \{x0^k \mid x \in L(M), k \geq 0\} \tag{1}$$

by induction on k running down from a sufficiently large k' to 0.

Let k' be a sufficiently large integer such that $t(|x|) < |x0^{k'}|$. As we mentioned in the last paragraph of Section 3.2, a computation of $M_{f(g(f,M'))}$ on $x0^{k'}$ implies a computation of M' on $f(g(f,M'))x0^{k'}$, and M' accepts it iff M accepts x. Therefore,

$$\begin{aligned} L(M_{f(g(f,M'))}) &= \{x0^{k'} \mid f(g(f,M'))x0^{k'} \in L(M')\} \\ &= \{x0^{k'} \mid x \in L(M)\}. \end{aligned} \tag{2}$$

From equation (2), one can see that $M_{f(g(f,M'))}$ accepts $x0^{k'}$ iff M accepts x, for any such large k'.

Consider an integer k such that $t(|x|) \geq |x0^k|$. Assume for induction that, for every $k' > k$, equation (2) holds. We observe the computation of $M_{f(g(f,M'))}$ on $x0^k$ (which implies the computation of M' on $f(g(f,M'))x0^k$.) Recall that the first task for M' was to pad $f(g(f,M'))x0^k$ to $f(g(f,M'))x0^{k'}$ for nondeterministically chosen $k' > k$, and the second task was to make the computation of U on $f(g(f,M'))x0^{k'}$. Thus,

$$L(M_{f(g(f,M'))}) = \{x0^k \mid f(g(f,M'))x0^{k'} \in L(U) \text{ for some } k' > k\}. \tag{3}$$

Since U is a universal CA, the computation of U on input $f(g(f,M'))x0^{k'}$ implies the computation of $M_{f(g(f,M'))}$ on input $x0^{k'}$. Therefore, the right-hand side of (3) can be rewritten as

$$L(M_{f(g(f,M'))}) = \{x0^k \mid x0^{k'} \in L(M_{f(g(f,M'))}) \text{ for some } k' > k\}. \tag{4}$$

From the induction hypothesis (see equation (2)), $M_{f(g(f,M'))}$ accepts $x0^{k'}$ iff M accepts x. Hence, the right-hand side of (4) is further rewritten as

$$L(M_{f(g(f,M'))}) = \{x0^k \mid x \in L(M)\}.$$

Thus, if equation (2) holds for every $k' > k$, then the same equation holds also for k. Hence, equation (1) holds for every $k \geq 0$.

3.4 Proof of Theorem 1

Let $t_2(n)$ be an arbitrary time-constructible function which is not bounded by $O(t_1(n+1))$. Assume for contradiction that there does not exist any language which can be accepted by $t_2(n)$-time CA but not by any $t_1(n)$-time CA. This assumption implies that any $t_2(n)$-time computation in CA can be sped-up to $t_1(n)$ time.

Recall that $L \subseteq \{1\}^*$ is any recursive language, and M is the deterministic CA recognizing L in $t(n)$ time. Note that $t(n)$ can be taken as rapidly as we want by choice of L. In the following paragraphs, we will prove by induction on n that, *for each sufficiently long x accepted by M, U accepts $f(g(f,M'))x0^k$ of length n in $t_1(n)$ time for every $n \geq |f(g(f,M'))x|$*. Here, the computation of U on $f(g(f,M'))x0^k$ in $t_1(n)$ time implies the computation of $M_{f(g(f,M'))}$ on $x0^k$ in $O(t_1(n))$ time, since U is a universal CA. It is not difficult to show that there is an $O(t_1(n))$-time CA, say, $M'_{f(g(f,M'))}$, such that

$$\begin{aligned} L(M'_{f(g(f,M'))}) &= \{x \,|\, x0^k \in L(M_{f(g(f,M'))})\} \\ &= \{x \,|\, x \in L(M)\} = L(M) = L. \end{aligned} \tag{5}$$

The second equation holds because of equation (1). Equation (5) implies that any language L (recognized by the deterministic CA M with no time restriction) can be accepted by an $O(t_1(n))$-time CA $M'_{f(g(f,M'))}$, a contradiction.

It remains to show that U accepts $f(g(f,M'))x0^k$ of length n in $t_1(n)$ time. Consider a sufficiently large n such that $n > |f(g(f,M'))x| + t(|x|)$. In this case, since the padding sequence is sufficiently long, the computation of $M_{f(g(f,M'))}$ on $x0^k$ can be done in linear time (which is less than $t_2(n)$). From the assumption, any $t_2(n)$-time computation (of $M_{f(g(f,M'))}$ on $x0^k$) can be sped-up to $t_1(n)$ time. Therefore, U can accept $f(g(f,M'))x0^k$ in $t_1(n)$ time.

Consider an integer n such that $|f(g(f,M'))x| \leq n \leq |f(g(f,M'))x| + t(|x|)$. Assume for induction that, for each sufficiently long x accepted by M, U accepts $f(g(f,M'))x0^{k+1}$ of length $n' = n+1$ in $t_1(n')$ time. Under this assumption, we will show that U accepts $f(g(f,M'))x0^k$ of length n in $t_1(n)$ time as follows.

Consider the nondeterministic computation tree of U on input $f(g(f,M'))x0^k$. The computation of U on $f(g(f,M'))x0^k$ implies the computation of $M_{f(g(f,M'))}$ on $x0^k$, which further implies the computation of M' on $f(g(f,M'))x0^k$ (see the last paragraph of Section 3.2). According to the definition, M' pads $f(g(f,M'))x0^k$ to $f(g(f,M'))x0^{k'}$ for nondeterministically chosen $k' > k$.

Consider the nondeterministic choice which pads $f(g(f,M'))x0^k$ to $f(g(f,M'))x0^{k+1}$. (We will analyse the height of the subtree rooted at $f(g(f,M'))x0^{k+1}$.) The time complexity for lengthening the padding sequence is $O(n)$. After lengthening the padding sequence, M' makes the computation of U on the padded input $f(g(f,M'))x0^{k+1}$. The computation of U on $f(g(f,M'))x0^{k+1}$ of length $n+1$ can be done in $t_1(n+1)$ time (because of the induction hypothesis). The total complexity is $t_1(n+1) + O(n)$. From the assumption of the theorem, $t_2(n)$ is not bounded by $O(t_1(n+1))$. Thus, U can accepts $f(g(f,M'))x0^k$ of length n in $t_2(n)$ time. From the assumption (of this section), any $t_2(n)$-time computation can be sped-up to $t_1(n)$ time. Therefore, U accepts $f(g(f,M'))x0^k$ of length n in $t_1(n)$ time. This completes the proof of Theorem 1.

4 Conclusion and Final Observations

In this paper, we presented a tight time-hierarchy theorem for nondeterministic cellular automata (NCA). It was shown that, for any time-constructible function $t_2(n)$ not bounded by $O(t_1(n+1))$, $t_2(n)$-time NCA are stronger than $t_1(n)$-time NCA.

Finally, we intuitively observe a recursive padding for the separation between nondeterministic classes of $t_1(N) = N^2/\log\log N$ and $t_2(N) = N^2$. Let L be any recursive language recognized by a deterministic CA (DCA) M in time $t(n) = 2^{2^n}$. Assume for contradiction that any $t_2(N)$-time (nondeterministic) computation is sped-up to $t_1(N)$ time.

We recursively pad the input x of length n until the length of $x00\cdots0$ becomes $N = 2^{2^n}$. Let $L_N = \{x0^{N-n} \mid x \in L\}$. Then, there is a CA accepting L_N in $t_2(N)$ time, since $t_2(N) = 2^{2^{n+1}}$ is larger than $t(n) = 2^{2^n}$. From the assumption, the $t_2(N)$-time computation for L_N is sped-up to $t_1(N)$ time. If such a $t_1(N)$-time computation exists, then there is a $t_2(N-1)$-time computation for L_{N-1} (see Section 3.4). Again, the $t_2(N-1)$-time computation is sped-up to $t_1(N-1)$ time, and thus there is a $t_2(N-2)$-time computation for L_{N-2}. By continuing this observation, one can see that there is a $t_2(n)$-time computation for $L_n = L$. Therefore, any recursive language L, which can be accepted by a 2^{2^n}-time DCA, can also be accepted by an n^2-time NCA. This contradicts the following simulation and separation results: (i) every n^2-time NCA can be simulated by a $2^{2^{n-1}}$-time DCA, and (ii) there is a language which can be accepted by a 2^{2^n}-time DCA but not by any $2^{2^{n-1}}$-time DCA [11].

References

1. S.A. Cook, A hierarchy for nondeterministic time complexity, *J. Comput. System Sci.* **7** (1973) 343–353.
2. S.A. Cook and R.A. Reckhow, Time bounded random access machines, *J. Comput. System Sci.* **7** (1973) 354–375.
3. M. Fürer, The tight deterministic time hierarchy, Proc. ACM Symp. on Theory of Computing, 8–16, 1982.
4. J. Hartmanis and R.E. Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965) 285–306.
5. F.C. Hennie, One-tape off-line Turing machine computations, *Inform. Contr.* **8** (1965) 553–578.
6. O.H. Ibarra, A note concerning nondeterministic tape complexity, *J. Assoc. Comput. Mach.*, **19** (1972) 608–612.
7. K. Iwama and C. Iwamoto, Parallel complexity hierarchies based on PRAMs and DLOGTIME-uniform circuits, Proc. 11th IEEE Conf. on Computational Complexity, 1996, 24–32.
8. K. Iwama and C. Iwamoto, Improved time and space hierarchies of one-tape off-line TMs, Proc. 23rd Int'l Symp. on Mathematical Foundations of Computer Science, LNCS 1450, Springer, 1998, 580–588.
9. C. Iwamoto, T. Andou, K. Morita, and K. Imai, Computational complexity in the hyperbolic plane, Proc. 27th Int'l Symp. on Mathematical Foundations of Computer Science, LNCS 2420, Springer, 2002, 365–374.

10. C. Iwamoto, N. Hatayama, K. Morita, K. Imai, and D. Wakamatsu, Hierarchies of DLOGTIME-uniform circuits, in M. Margenstern (ed.): Machines, Computations and Universality (Proc. MCU 2004, Saint-Petersburg, Sep. 21–26, 2004), LNCS 3354, Springer, 2005, 211–222.
11. C. Iwamoto, T. Hatsuyama, K. Morita, and K. Imai, Constructible functions in cellular automata and their applications to hierarchy results, *Theoret. Comput. Sci.* **270** (2002) 797–809.
12. C. Iwamoto and M. Margenstern, Time and space complexity classes of hyperbolic cellular automata, *IEICE Trans. on Information and Systems*, **E87-D** 3 (2004) 700–707.
13. C. Iwamoto, Y. Nakashiba, K. Morita, and K. Imai, Translational lemmas for alternating TMs and PRAMs, Proc. 15th Int'l Symp. on Fundamentals of Computation Theory, LNCS 3623, Springer, 2005, 126–137.
14. K. Loryś, New time hierarchy results for deterministic TMs, Proc. 9th Symp. on Theoretical Aspects of Computer Science, LNCS 577, Springer, 1992, 329–336.
15. J. Mazoyer, A 6-state minimal time solution to the firing squad synchronization problem, *Theoret. Comput. Sci.*, **50** (1987) 183–238.
16. W.J. Paul, On time hierarchies, *J. Comput. System Sci.* **19** (1979) 197–202.
17. W.J. Paul, E.J. Prauß, and R. Reischuk, On alternation, *Acta Inform.* **14** (1980) 243–255.
18. H. Rogers Jr., Theory of recursive functions and effective computability, McGraw-Hill, New York, 1967.
19. J.I. Seiferas, Nondeterministic time and space complexity classes, MIT-LCS-TR-137, Proj. MAC, MIT, Cambridge, Mass., Sept. 1974.
20. J.I. Seiferas, M.J. Fischer, and A.R. Meyer, Separating nondeterministic time complexity classes, *J. Assoc. Comput. Mach.*, **25** 1 (1978) 146–167.
21. M. Sipser, Introduction to the theory of computation, PWS Publishing, Boston, Mass., 1997.