INFORMATIQUE THÉORIQUE ET APPLICATIONS

MEENA MAHAJAN
KAMALA KRITHIVASAN

Language classes defined by time-bounded relativised cellular automata

Informatique théorique et applications, tome 27, n° 5 (1993), p. 403-432

<http://www.numdam.org/item?id=ITA_1993__27_5_403_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 27, n° 5, 1993, p. 403 à 432)

LANGUAGE CLASSES DEFINED BY TIME-BOUNDED RELATIVISED CELLULAR AUTOMATA (*)

by Meena MAHAJAN (1) and Kamala KRITHIVASAN (2)

Communicated by C. CHOFFRUT

Abstract. – *Some of the fundamental problems concerning cellular automata (CA) are as follows:*
- *Are linear-time CA (l CA) more powerful than real-time CA (r CA)?*
- *Are nonlinear-time CA more powerful than linear-time CA?*
- *Does one-way communication reduce the power of a CA?*

These question have been open for a long time. In this paper, we address these questions with respect to tally languages in relativised worlds, interpreting timevarying CA (TVCA) as oracle machines. We construct
- *oracles which separate r CA from l CA and l CA from CA,*
- *oracle classes under which the CA classes coincide, and*
- *oracles which leave the CA classes unchanged.*

Further, with r CA and l CA at the base, we build a hierarchy of relativised CA complexity classes between r CA and CA, and study the dependencies between the classes in this hierarchy.

1. INTRODUCTION

Cellular automata (*CA*) as language recognisers have been the object of study for several years [BC84, CC84, IJ88, IPK85, Smi71, Smi72]. A *CA* consists of a 1-dimensional array of identical finite-state machines called cells, one for each letter of the input. The cells operate synchronously at discrete

(*) Received October 29, 1991; accepted March 5, 1993.
(1) The Institute of Mathematical Sciences, Madras 600 113, India.
(2) Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600 036, India
A preliminary version of this paper was presented at the 11th FST and TCS Conference [MK 91].
This work was done when the second author was at the Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600 036. This research was partially supported by a grant from the Department of Science and Technology, Government of India.

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/93/05/$4.00/

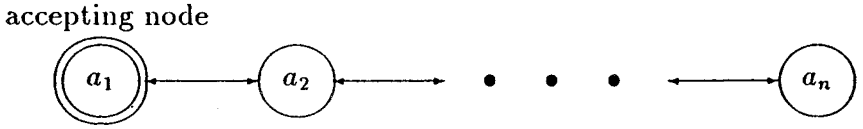


Figure 1. – A cellular automaton

time steps. See *fig.* 1. Let $c(i, t)$ denote the state of the ith cell at time t. The input word $a_1 a_2 \ldots a_n$ is fed to the *CA* by setting $c(i, 0)$ to a_i. The state of the ith cell at time $t+1$ is a function of its states and the states of its left and right neighbours at time t; thus $c(i, t+1)=\delta(c(i-1, t), c(i, t), c(i+1, t))$. (If a neighbouring cell is missing, *i.e.* at the boundaries of the array, a special state # is taken to be the missing argument.) The *CA* accepts the input $a_1 a_2 \ldots a_n$ if its leftmost cell eventually enters an accepting state.

Formally, a *CA* is defined as follows:

DEFINITION 1.1: *A cellular automaton is a* 4*-tuple* $C=(Q, \#, \delta, A)$ *where*

- Q *is a finite set of states*
- $\# \in Q$ *is the boundary state*
- $\delta: Q \times Q \times Q \to Q$ *is the local transition function satisfying*

$$\delta(a, b, c)=\# \quad \textit{if and only if} \quad b=\#$$

- $A \subseteq Q$ *is the set of accepting states.*

If inputs of length n are accepted within time $T(n)$, the *CA* is said to have time complexity $T(n)$. Clearly, $T(n) \geq n$ for non-trivial recognition. Of special interest are the cases when $T(n)=n$, given "real-time" *CA* (*rCA*), and $T(n)=cn$ for some constant c, giving "linear-time" *CA* (*lCA*). *CA* with no time restriction are equivalent to deterministic linear-space-bounded Turing machines [Smi72], and thus accept exactly the class of deterministic context-sensitive languages (*DCSLs*). It is not known whether *lCA* are more powerful than *rCA* or whether nonlinear-time *CA* are more powerful than *lCA*.

A restricted version of a *CA* is the one-way *CA* (*OCA*), where the communication between cells is from left to right only. Here, $c(i, t+1)$ is a function of $c(i-1, t)$ and $c(i, t)$ only. The rightmost cell must enter an accepting state for the input to be accepted. Like *CA*, *OCA* have also been studied in depth [BC84, CC84, CIV88, Dye80, IJ88, IJ87], because despite their simplicity, they are remarkably powerful. For instance, they can accept some *PSPACE*-complete languages, as well as all languages in $NSPACE(\sqrt{n})$ [CIV88, IJ87].

It is not known whether CA are more powerful than OCA. The containments and equalities known are as follows:

$$rOCA \subset rCA = lOCA \subseteq lCA \subseteq OCA \subseteq CA.$$

It is easy to see that lCA languages are contained in $DTIME(n^2) \subset P$. (In general, a $T(n)$-time CA can be simulated by a Turing machine in $O(nT(n))$ time, using only linear space.) Thus any proof that OCA are only as powerful as lCA (e.g. $lCA = CA$, $lOCA = OCA$, etc.) will immediately imply that $P = PSPACE$. On the other hand, a proof to the contrary seems extremely difficult to obtain.

The motivation for our work here has come from the field of structural complexity theory, specifically from the use of relativisation techniques. A relativised Turing machine has a separate tape for writing oracle queries, and three special states $q_?$, q_y, q_n. The state $q_?$ is used to ask whether the string on the query tape belongs to the oracle set. If this is so, then at the next time step the machine enters state q_y; else it enters state q_n. The computation then proceeds normally, until a fresh query is made by entering the state $q_?$ again. Relativisation of Turing machines and the study of the polynomial hierarchy (PH) have not answered the $P \stackrel{?}{=} NP$ question, but they have provided a lot of insight into the structural properties of these complexity classes. We hope that similar useful insights into the structure of the CA complexity classes rCA, lCA and CA can be obtained through relativisation. However, the notion of relativisation in the context of CA is difficult to formalise, because the control of the computation is not centralised but is distributed over all the cells. Besides, there is no tape on which to write out queries to the oracle. Even if queries were to be written in a separate component of the state of each cell, synchronising the query procedure would become contrived and cumbersome. We adopt the approach of using implicit oracle querying as provided by a time-varying CA ($TVCA$). $TVCA$ have been defined in [MK92], and several interesting properties have been studied. A $TVCA$ is similar to a CA except in one respect: the transition rule which specifies the next state of a cell in terms of the states of cells in its neighbourhood is not fixed, but depends on how many time steps have elapsed since the $TVCA$ operation began. Thus an oracle is implicitly queried on inputs $t = 1, 2, \ldots$, and its replies tell the CA which transition rule is to be used.

The querying mechanism as described above is represented most naturally using tally languages (languages over a unary alphabet); therefore, throughout the rest of this paper, only tally languages are considered as oracles. While

tally sets are often inadequate in capturing the complexity of various classes, they sometimes suffice to express strong interdependencies [Boo74]. For instance, tally sets are present in $NP-P$ if and only if $DEXT \neq NEXT$. Even when only tally sets are considered, the problem $r\mathrm{CA} \stackrel{?}{=} l\,CA$ is open [IJ88]. Though many conjecture that the classes (of tally sets) are distinct, no answer is forthcoming. Book's results (theorem 2, [Boo74]) imply that if, for tally languages, $l\,CA = CA$, then $EXPSPACE = EXPTIME$ and every tally language in $PSPACE$ belongs to P. Our work here is based on the conjecture that if the classes $r\,CA$, $l\,CA$ and CA are distinct, then there are tally sets in the difference.

In section 2, we define $TVCA$ and formalise the interpretation of $TVCA$ as oracle CA. In section 3 we construct oracles separating $r\,CA$, $l\,CA$ and unrestricted-time CA. We also present oracle sets relative to which these classes coincide, and relative to which they remain unchanged. The CA hierarchy is defined in section 4 and shown to be contained in the class of CA languages. This hierarchy is defined with the tally languages in $r\,CA$ and $l\,CA$ at the base, by iteratively relativising the classes obtained via oracles from the preceding level. Some interesting properties of this hierarchy are presented in section 5.

2. *TVCA* AS RELATIVISED *CA*

Time-varying cellular automata ($TVCA$) have been defined in [MK92] and several properties of these automata are investigated there. We rephrase the definition here in a form most convenient for this work. Informally, a $TVCA$ differs from a CA in the following way: the transition function δ depends on time. Thus different transition rules may be used at different times. We consider the simplest case where there are only two possible rules to be used, δ_1 and δ_2, and one of them is used depending on the current time. In the notation of [MK92], such $TVCA$ are referred to as 2-$TVCA$; since we will consider only such $TVCA$ here, we will drop the prefix 2-. The usage of δ_1 and δ_2 is controlled by a tally set L. If $0^i \in L$, then at time i rule δ_1 is used, else rule δ_2 is used. Thus a $TVCA$ is completely described by specifying Q, $\#$, δ_1, δ_2, L and A. Formally,

DEFINITION 2.1: *A TVCA is a* 6*-tuple* $C = (Q, \#, \delta_1, \delta_2, L, A)$ *where*

- Q *is a finite set of states*
- $\# \in Q$ *is the boundary state*

- *L is a tally set over* $\{0\}$
- δ_1 *and* δ_2 *are transition rules* $\delta_i : Q \times Q \times Q \to Q$ *satisfying* $\delta_i(a, b, c) = \#$ *if and only if* $b = \#$
- $A \subseteq Q$ *is the set of accepting states*

The composite transition function δ of the *TVCA* is then defined as follows:

$$\delta(a, b, c, i) = \begin{cases} \delta_1(a, b, c) & \text{if } 0^i \in L \\ \delta_2(a, b, c) & \text{otherwise} \end{cases}$$

From this definition it is clear that the *TVCA* acts as if it has an oracle answering queries about the membership of strings in L. The queries are quite restricted; the oracle must be queried at each time step, and it must be queried on strings $0^1, 0^2, \ldots$ in that order. Thus such a machine may not be equivalent to a $DSPACE(n)$ Turing machine with an oracle for L, even if they are the same without oracles. However, even this restrictive notion of querying appears to be quite non-trivial, and is used throughout this paper.

With this interpretation of 2-*TVCA* as oracle *CA*, we will henceforth specify a *TVCA* as $C(L)$, where $C = (Q, \#, \delta_1, \delta_2, A)$ and L is the oracle. When the oracle is an empty set, this denotes the $CA(Q, \#, \delta_2, A)$. Classes of *CA* $\mathcal{S}$ with a particular oracle L are denoted $\mathcal{S}(L)$; e.g. $rCA(L)$, $CA(L)$, etc. Classes of *CA* $\mathcal{S}$ relative to a set of tally oracles $\mathcal{L}$ are denoted $\mathcal{S}(\mathcal{L})$; e.g. $CA(rCA)$, etc. By $\mathcal{S}(\mathcal{L})$ we will mean both the class of machines in $\mathcal{S}$ using an oracle from $\mathcal{L}$ and the class of languages accepted by such machines. For a class of languages $\mathcal{L}$, we use $\mathcal{L}|_t$ to denote the restriction of $\mathcal{L}$ to tally languages.

For some of the proofs in the next section, we need to fix an enumeration of rCA and lCA. To completely specify a *CA*, two notions must be specified: the *CA* itself (*i.e.* its state set and its transition function), and the set of accepting states. In addition, if an lCA is being specified, then the maximal time at which the accepting cell can be checked should also be indicated. These notions are encoded by integers as follows. We assume that the state set of a *CA* is $\{0, 1, \ldots, k\}$ for some finite k, and that # is the state 0. The number of distinct transition rules is $m = k^{(k+1)^2 k}$ (since # always maps to # and no other symbol maps to #), and the number of possible sets of accepting states is 2^k (since # cannot be an accepting state). Thus a *TVCA* can be specified, without the oracle language, as a 4-tuple (k, i_1, i_2, j) where i_1, i_2 are integers between 1 and m specifying the transition rules, and j is an integer between 1 and 2^k specifying A. Let ϕ_i, $i = 1, 2, \ldots$ be an ordering of such 4-tuples. This serves as an enumeration of *CA* as well as rCA. To

enumerate lCA, we order pairs (ϕ_i, c_j) where ϕ_i is the machine and c_j specifies the constant for linear-time acceptance. Let this ordering by ψ_i, an enumeration of lCA. Since we allow the set of accepting states to be empty, CA s accepting the empty set will occur infinitely often in both these enumerations.

3. RELATIVISED *CA* CLASSES

In this section we show how different oracle sets differently affect the containment $rCA \subseteq lCA \subseteq CA$. We first construct oracles separating these classes. We also show how to effect strong separations via immune sets. A set X is said to be immune to a class $\mathscr{L}$ ($\mathscr{L}$-immune) if X is infinite and contains no infinite subset belonging to $\mathscr{L}$. An oracle L strongly separates classes $\mathscr{L}_1$ and $\mathscr{L}_2$, where $\mathscr{L}_1 \subseteq \mathscr{L}_2$, if $\mathscr{L}_2(L)$ contains a set that is $\mathscr{L}_1(L)$-immune.

We will construct oracle sets A and B such that $lCA(A) \neq CA(A)$ and $rCA(B) \neq lCA(B)$. We will then generalise the construction to obtain sets C and D such that C (respectively, D) strongly separates lCA from CA (respectively, rCA from lCA). All these separations hinge around the fact that in our model of relativisation, a time bound imposes a stringent bound on the potential query space. Before doing so we will show some intermediate results.

DEFINITION 3.1: *A function $T(n): \Sigma^+ \to \mathbf{N}^+$ is said to be CA-time-constructible if there is a CA which, on any input of length n, puts its accepting cell into a special state after exactly $T(n)$ time steps.*

PROPOSITION 3.2: *Let $f: \Sigma^+ \to \mathbf{N}$ be a CA-time-constructible function, and let A be some set acting as an oracle. If A is tally, then the set $L_{ft}(A)$, given by*

$$L_{ft}(A) = \{ x \in \Sigma^+ \mid 0^m \in \mathrm{A}, \text{ where } m = f(x) \}$$

can be accepted by a CA, with tally oracle A, in time $f(x)$. Thus if $f_1: \mathbf{N} \to \mathbf{N}$ is the function defined as $f_1(n) = \max_{|x|=n} f(x)$, then $L_{ft}(A)$ can be accepted by a CA, wich oracle A, in time $f_1(n)$.

Proof: Since f is CA-time-constructible, we can design a relativised CA where both δ_1 and δ_2 compute $m = f(x)$. At time instant m, only δ_1 puts the CA into an accepting state. Thus $L_{ft}(A)$ is accepted. ■

LEMMA 3.3: 1. *$f: \{0\}^+ \to \mathbf{N}$, where $f(0^n) = 2n$, is CA-time-constructible.*

2. *$g: \{0\}^+ \to \mathbf{N}$, where $g(0^n) = n^2$, is CA-time-constructible.*

Proof: (*a*) This is straightforward – the *CA* just has to send a signal from right to left at half speed.

(*b*) This is achieved as follows. At $t=1$, the rightmost cell enters a special bounding state **b** and also sends a signal \$ left. \$ travels upto a cell marked **b** and then returns to the rightmost cell before setting out leftwards again. Every time \$ reaches a **b** cell, the **b** marker moves one unit left. Thus the \$ goes through excursions of length $2\times 1,\ 2\times 2,\ \ldots,\ 2\times i,\ \ldots$ When the \$ reaches the leftmost cell, the number of steps elapsed is $\left[\sum_{i=1}^{n-1}(2\times i)\right]+n=n^2$. An example is shown in figure 2. ■

For tally sets A and B, let

$$L_1(\mathrm{A})=\{0^s \mid 0^{s^2}\in \mathrm{A}\}=\sqrt{A},$$

and

$$L_2(B)=\{0^s \mid 0^{2s}\in B\}=\frac{1}{2}B.$$

Then, in the above notation, $\sqrt{A}=L_{gt}(A)$ and $1/2\,B=L_{ft}(B)$, where f and g are as in Lemma 3.3. Clearly, $f_1(n)=2n$ and $g_1(n)=n^2$. The next lemma now follows from the preceding two results.

LEMMA 3.4

$$\forall A,\quad \sqrt{A}\in CA(A).$$

$$\forall B,\quad \frac{1}{2}B\in lCA(B).$$

By direct diagonalisation we can now construct sets A and B such that $\sqrt{A}\notin lCA(A)$ and $(1/2)\,B\notin rCA(B)$, giving the following result.

THEOREM 3.5: (*a*) *There exists an oracle A such that* $lCA(A)\neq CA(A)$.

(*b*) *There exists an oracle B such that* $rCA(B)\neq lCA(B)$.

Proof: (*a*) Let $\psi_1, \psi_2, \ldots$ be an enumeration of relativised lCA, with constants $c_1, c_2, \ldots$ For any tally set X, $\sqrt{X}$ can be accepted by a *CA* with oracle X. We will incrementally construct a tally set A such that for any relativised lCA ψ_i, the language accepted by ψ_i with oracle A differs from $\sqrt{A}$. This will prove the theorem's first assertion.

Stage 0: $A_0=\varnothing$, $m_0=0$.

Stage i: Choose the smallest integer m_i satisfying

t					
0	0	0	0		#
1	.	.	.	b $\uparrow\$$	#
2	.	.	b	$\downarrow\$$	#
3	.	.	b	$\uparrow\$$	#
4	.	.	b $\uparrow\$$	.	#
5	.	b	$\downarrow\$$	.	#
6	.	b	.	$\downarrow\$$	#
7	.	b	.	$\uparrow\$$	#
8	.	b	$\uparrow\$$	.	#
9	.	b $\uparrow\$$	.	.	#
10	b	$\downarrow\$$	.	.	#
11	b	.	$\downarrow\$$	.	#
12	b	.	.	$\downarrow\$$	#
13	b	.	.	$\uparrow\$$	#
14	b	.	$\uparrow\$$	.	#
15	b	$\uparrow\$$	.	.	#
16	b $\uparrow\$$	.	.	.	#

Figure 2. – Computing n^2 on a *CA*.

1. $c_i m_i < m_i^2$
2. $\forall j < i, c_j m_j < m_i^2$
3. $\forall j < i,\ m_i \neq m_j$.

The first condition ensures that with input 0^{m_i}, ψ_i does not get a chance to query A on $0^{m_i^2}$. The second condition ensures that the inclusion/exclusion of $0^{m_i^2}$ in/from A does not change the behaviour of the *TVCA* already considered. The third condition ensures that a string once excluded from A cannot subsequently by included in A.

Simulate ψ_i on 0^{m_i} for $c_i m_i$ steps, using A_{i-1} as oracle. If 0^{m_i} is accepted, then $A_i = A_{i-1}$, else $A_i = A_{i-1} \cup \{0^{m_i^2}\}$.

$$A = \lim_{n \to \infty} A_n = \{w \mid w \text{ belongs to all but finitely many } A_n\}.$$

Since our construction never deletes strings from any A_n, $A = \bigcup_{n>0} A_n$.

Claim: The set A so constructed satisfies "$\sqrt{A}$ cannot be accepted by any relativised lCA with oracle A".

Proof of claim: By contradiction. Assume that for some i, $\psi_i(A)$ accepts $\sqrt{A}$. On input 0^{m_i}, ψ_i queries A on strings of length upto $c_i m_i$. By our construction,

$$A \cap \{0^j \mid j \leqq c_i m_i\} = A_{i-1} \cap \{0^j \mid j \leqq c_i m_i\}$$

Thus on input 0^{m_i}, $\psi_i(A)$ behaves as $\psi_i(A_{i-1})$. Suppose $\psi_i(A_{i-1})$ accepts 0^{m_i}. Then our construction excludes $0^{m_i^2}$ from A_i and A. Thus 0^{m_i} is in $L(\psi_i(A_{i-1})) - \sqrt{A}$. On the other hand, if $\psi_i(A_{i-1})$ does not accept 0^{m_i}, then $0^{m_i^2}$ is included in A_i. Since words are never deleted from A, it remains in A, and hence 0^{m_i} is in $\sqrt{A}$. Thus 0^{m_i} is in $\sqrt{A} - L(\psi_i(A_{i-1}))$. In either case, $\psi_i(A)$ cannot be correctly accepting $\sqrt{A}$.

Essentially, our construction ensures that $\forall i$, $0^{m_i} \in L(\psi_i(A)) \Delta \sqrt{A}$. This proves the claim.

(*b*) Let ϕ_i, $i = 1, 2, \ldots$ be an enumeration of relativised rCA. We will incrementally construct a tally set B such that $(1/2)\,B$ is not accepted by any $\phi_i(B)$. Since $(1/2)\,B$ can be accepted by an lCA with oracle B, the assertion will be proved.

Stage 0: $B_0 = \varnothing$.

Stage i: Simulate ϕ_i on the string 0^i for i steps, using oracle B_{i-1}. If ϕ_i accepts the input, then $B_i = B_{i-1}$, else $B_i = B_{i-1} \cup \{0^{2i}\}$.

$$B = \lim_{n \to \infty} B_n = \{w \mid w \text{ belongs to all but finitely many } B_n\}$$

Claim: The set B so constructed satisfies "$(1/2)\,B$ cannot be accepted by any relativised rCA with oracle B".

This proof is similar to that in (*a*). Here we ensure that $\forall i$, $0^i \in L(\phi_i(B)) \Delta (1/2)\,B$. ■

COROLLARY 3.6: *There exists an oracle X such that*

$$rCA(X) \neq lCA(X) \neq CA(X).$$

Proof: In the above theorem, we have seen how to construct oracles A and B separating CA from lCA and lCA from rCA respectively. If the construction of A is modified so that only odd length strings are chosen (choose odd m_i), the separation is still valid. Now the even length strings can be used to separate lCA from rCA, as in the construction of B. ϕ_i will now be simulated on the string 0^{2i} instead of 0^i, and according to the outcome 0^{4i} may or may not be added to the oracle. It is easy to see that the oracle so constructed simultaeneously separates CA from lCA and lCA from rCA. ■

These results can be strengthened to strong separations; using delayed diagonalisation in a similar fashion as above, we can show the following (*see* [Mah92]).

THEOREM 3.7: (*a*) *There is a tally oracle C such that $CA(C)$ contains an $lCA(C)$-immune set.*

(*b*) *There is a tally oracle D such that $lCA(D)$ contains an $rCA(D)$-immune set.*

The proofs of both Theorem 3.5 and Theorem 3.7 are identical in nature to the corresponding oracle constructions for separating P and NP; refer [BDG90].

In [MK92] we have shown the following:

THEOREM 3.8: *Let $C = CA(X)$ be a TVCA with oracle X. If X is ultimately periodic (this includes regular languages), then C can be simulated by a non-time-varying CA with no loss of time.*

The proof is included in the appendix.

The following proposition is easily verified (for completeness, this is also proved in the appendix).

PROPOSITION 3.9: *$rOCA|_t$ equals the class of tally regular sets.*

Thus as an oracle class, the class $rOCA|_t$ has no effect on the classes rCA, lCA and CA. Hence the problem of whether rCA are properly contained in lCA remains unchanged in this relativised world.

THEOREM 3.10

$$rCA(rOCA|_t) = rCA$$

$$lCA(rOCA|_t) = lCA$$

$$CA(rOCA|_t) = CA$$

At the other extreme, when the class $CA|_t$ is used as oracles, $rCA|_t$ become as powerful as $lCA|_t$, as seen below.

THEOREM 3.11: $rCA(CA|_t)|_t = lCA(CA|_t)|_t = CA|_t$.

Proof: Clearly, $rCA(CA|_t)|_t \subseteq lCA(CA|_t)|_t$. Observe that for any tally set A, A can be accepted in real time by some CA using oracle A. Thus $CA|_t \subseteq rCA(CA|_t)|_t \subseteq lCA(CA|_t)|_t$. To show that $lCA(CA|_t)|_t \subseteq CA|_t$, let $\psi(A)$ be an lCA using oracle A, where A can be accepted by $CA\,\phi$. Let the constant for ψ be c. Then we can construct a $CA\,\phi'$ which, for the ith step of ψ(A), first simulates ϕ on the i length input 0^i, and then uses the outcome, after re-synchronising the array, to simulate one step of ψ. Since ψ is an lCA, ϕ has to be simulated on inputs upto length cn. This requires a cn length array, which can be compacted onto the n length array of ϕ'. ϕ' thus simulates ψ(A). Thus $CA|_t \subseteq rCA(CA|_t)|_t \subseteq lCA(CA|_t)|_t \subseteq CA|_t$, implying the theorem. ■

Thus relativisation with respect to the class $CA|_t$ merges the classes $rCA|_t$ and $lCA|_t$ and makes them equal to the class $CA|_t$. However relativisation with respect to the same class $CA|_t$ increases the power of the class CA much further. There is a CA controlled by a CA oracle which accepts languages provably not acceptable by CA; this result follows from the following theorem and the space hierarchy theorem [BDG88] which strictly separates DSPACE(n) from DSPACE(2^n). Before stating the theorem we prove a simple proposition. For any positive natural number n, if $1x$ is the unique binary representation of n, then x, denoted $\langle n \rangle$, is the standard representation for the number n.

PROPOSITION 3.12: *There is a CA which, given input* $\langle n \rangle$, *puts its rightmost cell into a special state* **S** *after exactly n steps.*

Proof: The CA with input $\langle n \rangle$ counts n time steps by subtracting 1, at each time step, from the number in its array. The leftmost "active" cell has two bits of the number. Initially the leftmost cell of the CA is the leftmost "active" cell and behaves as if it has $1\,b_k$, where b_k is the leftmost bit of $\langle n \rangle$. Subtraction occurs at the rightmost end, with "borrow" signals travelling left as and when generated, and the leftmost "active" cell gradually shifting right.

An example for $\langle n \rangle = 011$, is shown in figure 3. Since the input is 011, the number in question is the binary number 1011 (the leading 1 is implicit), *i.e.* 11 in decimal notation. The leftmost cell initially sets its state to 10 and is the left-most "active" cell. The rightmost cell subtracts 1 from its contents at each step. When it has only 0, it sets the 0 to 1 and also sets a **b** flag in

$t = 0$	#	0	1	1	#
1	#	10	1	0	#
2	#	10	1	1_b	#
3	#	10	0	0	#
4	#	10	0	1_b	#
5	#	10	11	0	#
6	#	X	11	1_b	#
7	#	X	10	0	#
8	#	X	10	11	#
9	#	X	X	10	#
10	#	X	X	01	#
11	#	X	X	S	#

Figure 3. – Computing n from $\langle n \rangle$ on a *CA*

its state, indicating that it has "borrowed" a 1 from its left neighbour. A cell which sees this **b** flag in its right neighbour's state subtracts one from its own contents – if necessary, borrowing a 1 from its left neighbour in a similar fashion. The exception is when a cell needs to borrow a 1 and finds that its left neighbour is the leftmost "active" cell and contains 10. Clearly, this cell will become the leftmost "active" cell after the borrowing, so it directly sets itself to 11. A cell with a 10, on seeing its right neighbour set itself to 11, knows that it is no longer required to be "active", and sets itself to some special state X. The computation ends when the rightmost cell becomes the leftmost "active" cell and decrements its contents to 00. ■

THEOREM 3.13: $DSPACE\ (2^n) \subseteq CA\ (CA|_t)$

Proof: Let L be any language in DSPACE (2^n). Consider the tally version TALLY(L) defined as $\{ 0^n \mid \langle n \rangle \in L \}$. Since the length of 0^n is exponential in the length of $\langle n \rangle$, it is easy to see that for any language L in DSPACE (2^n), TALLY(L) is in DSPACE$(n)|_t = CA|_t$. Now it is easy to construct a *TVCA* which has δ_1 and δ_2 alike everywhere except in the presence of S. The input to the *TVCA* is $\langle n \rangle$, and both δ_1 and δ_2 compute n in time, as in the above

proposition. In the presence of S, δ_1 puts the *TVCA* in an accepting state and δ_2 puts it in a rejecting state. Such a *TVCA*, with a controlling language TALLY(L) from $CA|_t$, can thus accept any language L in DSPACE(2^n). ■

COROLLARY 3.14: $CA \subset CA(CA|_t)$

The next theorem strengthens Theorem 3.11.

THEOREM 3.15: $rCA(OCA|_t)|_t = lCA(OCA|_t)|_t = OCA|_t$.

Proof: Clearly, $OCA|_t \subseteq rCA(OCA|_t)|_t \subseteq lCA(OCA|_t)|_t$. To show that $lCA(OCA|_t)|_t \subseteq OCA|_t$, we will use the sequential machine characterisation for *OCA*. It has been shown [CIV88, IJ87, IPK85] that *OCA* are equivalent to a restricted form of an online single-tape Turing machine, called a Sweeping Automaton (*SA*). An *SA* consists of a semi-infinite worktape (bounded at the left by a special boundary marker ¢) and a finite-state control with an input terminal at which it receives the serial input $a_1 a_2 \ldots a_n$. The symbol \$ is used as endmarker. The *SA* operates in left-to-right sweeps as follows:

Initially, all cells of the worktape to the right of ¢ contain the blank symbol λ. A sweep begins with the read-write-head (*RWH*) scanning ¢ and the machine in a distinguished state q_0. In the ith sweep, the machine reads a_i and moves right of ¢ into a non-q_0 state. It continues moving right, rewriting non-λ symbols by non-λ symbols and changing states except into q_0. When the RWH reads a λ, it rewrites it by a non-λ symbol and resets to the leftmost cell in state q_0 to begin the next sweep. When \$ is first read, the machine completes the $(n+1)$th sweep, writes a \$ in the $(n+1)$th tape cell, and resets to ¢ in state q_0. Subsequent sweeps are performed between ¢ and \$ without expanding the workspace. \$ is assumed to be always available for reading after the input is exhausted. The input is accepted if the machine eventually enters an accepting state at the end of a sweep.

Several techniques for programming an *SA* have been described in [CIV88]. For a full description of how the techniques are implemented on an *SA*, the reader is referred to [CIV88].

We will show that any language in the class $lCA(OCA|_t)|_t$ can be accepted by an *SA*. This will prove the theorem.

Let L be a language accepted by an $lTVCA\psi$ controlled by the *OCA* language A. A is accepted by an *OCA* φ. Choose constant c sufficiently large so that L is accepted by ψ in $T(n) < cn$ time steps. Let the input be $a_1 a_2 \ldots a_n$. Construct *SA* M accepting L as follows:

The *SA* operates in sweeps, reading a_i at the beginning of the ith sweep. In the first sweep, M creates $2c$ subcells in the first worktape cell, and puts

a boundary marker **b** on the cth subcell. It also puts markers [and] on the $(c+1)$th subcell, and writes a_1 on the $(c+1)$th subcell (along with the [and]). In subsequent sweeps while reading the input, it creates $2c$ new subcells per sweep (in the first λ cell read). It also moves **b** and [c subcells right, and] $c+1$ subcells right. The characters a_1 to a_{i-1} are shifted c subcells right, and a_i is written, with], beyond them. Thus the worktape is partitioned by **b** into two parts such that after the ith sweep, each part has ci subcells. In the second part the first i subcells are marked off between [and], and hold the input read so far, one character per subcell. Each subcell in the left part holds the unary character 0 (appart from possibly **b**).

When M starts getting \$ as input, it begins the actual simulation. M places a $*$ on subcell 1 to indicate that membership of input 0^1 in A is to be determined. φ is simulated on input 0^{cn} in the left part. Since an *OCA* has only two arguments in its transition function, the left cell's state and the current cell's state, the state information can be updated in a left-to-right sweep. Let $c(i, t)$ denote the state of the ith cell of φ at time t. Because of one-way communication, $c(i, t)$ is the same for all input lengths $n \geqq i$. So simulating φ on input 0^{cn} also gives simulations of φ on input 0^i, $i \leqq cn$. If in any sweep the ith subcell in the left part enters an accepting state of φ, the subcell is marked with a **Y** indicating that input 0^i belongs to A; whenever it enters a rejecting state, the subcell is marked **N**. (Since *OCA* are closed under complementation, accepting and rejecting states can be defined.)

In any sweep, if M encounters a **Y** (**N**) in a subcell marked $*$, then the current query to A has been answered, so M moves the $*$ one subcell right to query A on the next input. It then simulates one transition of ψ in the region between and including the subcells marked [and] in the right part, using transition δ_1 (δ_2). However, since ψ is a two-way *CA*, a simulation in a left-to-right sweep will shift its configuration one unit right. The [and] markers are also correspondingly shifted. Since ψ operates within time cn, the right part is provided with cn subcells to allow for the shifting configuration. In any sweep if an accept state of ψ is written on the subcell marked [, then this means that the *lCA* ψ has accepted the input. So M completes this sweep by moving right in a final state.

In sweeps where neither **Y** nor **N** are found on the $*$ subcell, the $*$ is kept where it is and the right part is left unchanged.

Thus the membership of strings in the controlling language is answered in the cn left subcells. As and when an answer to the next query is available, the corresponding transition step of the *TVCA* is simulated in n subcells in the right part.

When M attempts to move $*$ beyond **b**, all cn queries to A have been answered and this sweep will complete the operation of ψ. So by this time if M has not found an accept state in the [subcell, it moves right in a rejecting state.

	a_1	a_2	a_3
δ_1	a	b	c
δ_2	d	e	f
δ_2	g	h	i
δ_1	j	k	
δ_2	l		

(a) lCA ψ on input $a_1a_2a_3$

0	0	0	0	0
z_1^1	z_2^1	z_3^1	z_4^1	z_5^1
$_Y z_1^2$	z_2^2	z_3^2	z_4^2	z_5^2
z_1^3	z_2^3	$_N z_3^3$	z_4^3	z_5^3
z_1^4	$_N z_2^4$	z_3^4	z_4^4	z_5^4
z_1^5	z_2^5	z_3^5	z_4^5	z_5^5
z_1^6	z_2^6	z_3^6	$_Y z_4^6$	z_5^6
z_1^7	z_2^7	z_3^7	z_4^7	z_5^7
z_1^8	z_2^8	z_3^8	z_4^8	z_5^8
z_1^9	z_2^9	z_3^9	z_4^9	$_N z_5^9$

(b) Computation of OCA φ on 0^5

(Accepting states are marked **Y**, rejecting states **N**)

Figure 4.

input	cell 1				cell 2				cell 3			
a_1	0	b	$[a_1]$	.	.	.	.	.	.	.	.	.
a_2	0	0	0	b	$[a_1$	$a_2]$	.	.	.	.	.	.
a_3	0	0	0	0	0	b	$[a_1$	a_2	$a_3]$	.	.	.
\$	$\dagger_* z_1^1$	z_2^1	z_3^1	z_4^1	z_5^1	b	$[a_1$	a_2	$a_3]$	.	.	.
\$	$\dagger_Y z_1^2$	$*z_2^2$	z_3^2	z_4^2	z_5^2	b	.	$[a$	b	$c]$	.	.
\$	$\dagger_Y z_1^3$	$*z_2^3$	$_N z_3^3$	z_4^3	z_5^3	b	.	$[a$	b	$c]$	.	.
\$	$\dagger_Y z_1^4$	$_N z_2^4$	$*_N z_3^4$	z_4^4	z_5^4	b	.	.	$[d$	e	$f]$	.
\$	$\dagger_Y z_1^5$	$_N z_2^5$	$_N z_3^5$	$*z_4^5$	z_5^5	b	.	.	.	$[g$	h	$i]$
\$	$\dagger_Y z_1^6$	$_N z_2^6$	$_N z_3^6$	$_Y z_4^6$	$*z_5^6$	b	.	.	.	.	$[j$	k
\$	$\dagger_Y z_1^7$	$_N z_2^7$	$_N z_3^7$	$_Y z_4^7$	$*z_5^7$	b	.	.	.	.	$[j$	k
\$	$\dagger_Y z_1^8$	$_N z_2^8$	$_N z_3^8$	$_Y z_4^8$	$*z_5^8$	b	.	.	.	.	$[j$	k
\$	$\dagger_Y z_1^9$	$_N z_2^9$	$_N z_3^9$	$_Y z_4^9$	$_N z_5^9$	$*b$	.	.	.	.	.	$[l$

(c) SA simulating $\psi(\varphi)$

Figure 4. – An *SA* simulating an *lCA (OCA)* computation

One such simulation is depicted in figure 4. For the lCA operation shown in figure 4 (a), with the oracle OCA operating as in figure 4 (b), the worktape profile for the corresponding SA is shown in figure 4 (c). (The † symbol is printed by the SA on the first subcell in the $(n+1)$th sweep to allow the SA to tell this sweep apart from subsequent sweeps. This is crucial because only in this sweep should the SA print a $*$ on the first subcell) ■

In theorems 3.11 and 3.15, showing the containments from left to right requires only the oracle to be tally, not the accepted languages. Thus, with minor modifications, we can also show that

THEOREM 3.16

$$rCA(CA|_t) \subseteq lCA(CA|_t) \subseteq CA \subset CA(CA|_t)$$
$$rCA(OCA|_t) \subseteq lCA(OCA|_t) \subseteq OCA$$

Thus for the oracle class below rCA, *i.e.* $rOCA$, relativisation does not alter the $rCA \overset{?}{=} lCA$ question. For the oracle classes above lCA, *i. e.* OCA and CA, relativisation merges $rCA|_t$ and $lCA|_t$. The question naturally occurring at this point is: What happens under relativisation with respect to classes between $rOCA$ and OCA? This motivated us to construct the cellular automata hierarchy, which is the subject of the next two sections. This hierarchy is obtained by repeatedly relativising the classes $rCA|_t$ and $lCA|_t$, using the previously obtained classes as oracles.

4. THE *CA* HIERARCHY

The CA hierarchy is formally defined as follows:

DEFINITION 4.1: *The cellular automata hierarchy* $(CAH|_t)$ *of tally languages is the structure formed by the classes* $rrCA_k$, $lrCA_k$, $llCA_k$ *and* $rlCA_k$, *for each* $k \geqq 0$, *where*

1. $rrCA_0 = rlCA_0 = rCA|_t$
2. $llCA_0 = lrCA_0 = lCA|_t$
3. $rrCA_{k+1} = rCA(rrCA_k)|_t$
4. $lrCA_{k+1} = lCA(rrCA_k)|_t$
5. $llCA_{k+1} = lCA(llCA_k)|_t$
6. $rlCA_{k+1} = rCA(llCA_k)|_t$

Also, $CAH|_t = \bigcup_{k \geqq 0} (rrCA_k \cup rlCA_k \cup lrCA_k \cup llCA_k)$.

Some elementary properties of the cellular automata hierarchy are given below.

PROPOSITION 4.2: (*a*) $llCA_0 \subseteqq lrCA_1$.

(*b*) $\forall k \geqq 0$, $llCA_k \subseteqq rlCA_{k+1}$.

(*c*) $\forall k \geqq 0$, $rrCA_k \subseteqq rrCA_{k+1}$; $lrCA_k \subseteqq lrCA_{k+1}$; $rlCA_k \subseteqq rlCA_{k+1}$; $llCA_k \subseteqq llCA_{k+1}$

(*d*) $\forall k \geqq 0$, $rrCA_k \subseteqq lrCA_k \subseteqq llCA_k$; $rrCA_k \subseteqq rlCA_k \subseteqq llCA_k$

Proof: (*a*), (*b*), (*c*): Obvious, because the empty set belongs to all these classes, and because with oracle A, A can be accepted in real time.

(*d*): This is proved by induction. The assertion is obviously true for $k=0$. Assume it is true upto $k-1$. Now $rr\,CA_k$ and $rl\,CA_k$ are both real time CA, but the oracle set of $rl\,CA_k$, by the induction hypothesis, contains the oracle set of $rr\,CA_k$. So $rr\,CA_k \subseteq rl\,CA_k$. $rl\,CA_k$ and $ll\,CA_k$ both have the oracle set $ll\,CA_{k-1}$, but the CA from the class $rl\,CA_k$ can use only real time, while CA from the class $ll\,CA_k$ can use linear time. So $rl\,CA_k \subseteq ll\,CA_k$. The other inclusions are similarly shown. Thus the assertions are true for all k. ■

THEOREM 4.3: $CAH|_t \subseteq (OCA \cap P)|_t$.

Proof: By statement (*d*) of the previous proposition, it suffices to show that $\forall k$, $ll\,CA_k \subseteq (OCA \cap P)|_t$. This is shown by induction. $ll\,CA_0 = l\,CA|_t$ is clearly in the class $(OCA \cap P)|_t$. Let $ll\,CA_{k-1}$ be in $(OCA \cap P)|_t$. Then $ll\,CA_k$ is contained in the class $l\,CA\,(OCA|_t)|_t$, which by Theorem 3.16 is contained in $OCA|_t$. Also, $ll\,CA_k$ is contained in the class $l\,CA\,(P|_t)|_t$, which can be easily seen to be contained in $P(P)|_t = P|_t$. ■

This result, along with Book's results [Boo74], immediately yields the following corollary:

COROLLARY 4.4: *If* $CAH|_t = CA|_t$, *then* $EXPSPACE = EXPTIME$ *and every tally language in* $PSPACE$ *belongs to* P.

This suggests that while the power of the class $l\,CA$ may be increased some-what due to repeated relativisations with respect to previously obtained classes, it is unlikely to increase sufficiently to equal the class $CA|_t$, or even $OCA|_t$.

The following theorem is mentioned in this section essentially for completeness; the actual proof is provided only in the next section.

THEOREM 4.5: *If* $rr\,CA_0 = ll\,CA_0$, *then* $\forall k$,

$$rr\,CA_k = rl\,CA_k = lr\,CA_k = ll\,CA_k = r\,CA|_t.$$

Consequently, $CAH|_t = r\,CA|_t$.

This theorem says that if the classes $r\,CA|_t$ and $l\,CA|_t$ are equal, then for tally sets, linear time can be brought down to real time even in the presence of any oracle from $CAH|_t$. Consequently, the entire hierarchy collapses.

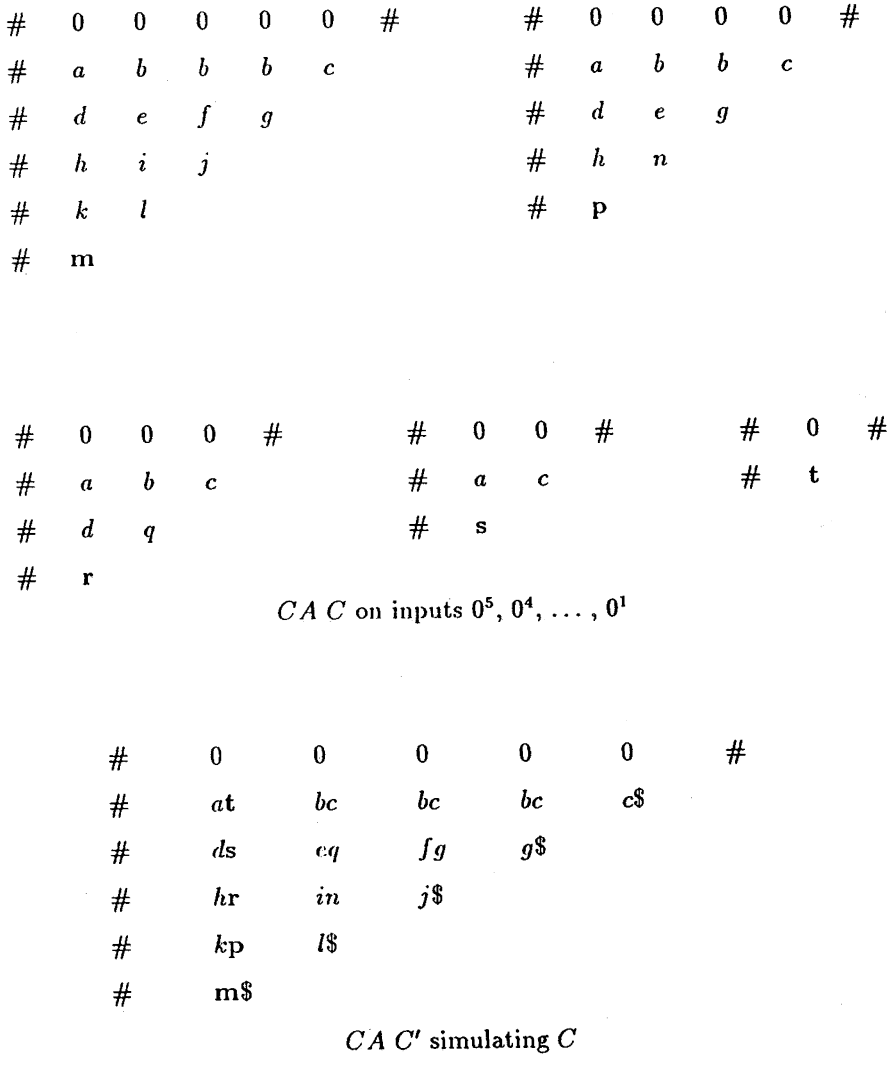


Figure 5. – Simulating an *r CA* on all prefixes of the input in real time

5. THE STRUCTURE OF THE CELLULAR AUTOMATA HIERARCHY

In this section we show some interesting inclusions in the cellular automata hierarchy. We first need some preliminary results.

LEMMA 5.1: *Let L be a (tally) language accepted by rCA C. We can effectively construct CAs C' and C'' which, on input 0^n, do the following:*

(a) At time step i, the accepting cell of C' specifies whether or not $0^i \in L$.

(b) At time step $2i-1$, the accepting cell of C'' specifies whether or not $0^i \in L$.

Proof: (a) Consider the time-space unrolling of C. This is an array where the topmost row has the input configuration, and successive configurations appear in successive rows beneath it. Thus the ith row gives the configuration of the CA after i time steps, and the jth column gives the sequence of states entered by the jth cell of the CA. (Figures 2 and 3 are such examples.) In this diagram, the unrollings of C on inputs 0^i and 0^{i+1} differ only in the ith diagonal from right to left. So we can construct C' so that each cell stores the corresponding values in the unrollings of two input lengths. This allows C to be simulated on all input lengths. An example is shown in figure 5.

More specifically, let $c^n(i, t)(\bar{c}^n(i, t))$ denote the state of the ith cell of C (C'), on input 0^n, at time t. Then $\bar{c}^n(i, t)$ contains both $c^n(i, t)$ and $c^{i+t-1}(i, t)$. $\bar{c}^n(1, t)$ will now contain $c^t(1, t)$ as the second component of its state for $t<n$, denoting membership of 0^t in L, and at $t=n$ it will contain $[c^n(1, n), \$]$. To achieve this, let δ be the transition function of C. Then h, the transition function of C', is given by the following rules. The first four rules give the transitions at $t=1$ and the other rules are used at subsequent steps.

$$
\begin{aligned}
h(\#, 0, \#) &= [\delta(\#, 0, \#), \$] \\
h(\#, 0, 0) &= [\delta(\#, 0, 0), \delta(\#, 0, \#)] \\
h(0, 0, 0) &= [\delta(0, 0, 0), \delta(0, 0, \#)] \\
h(0, 0, \#) &= [\delta(0, 0, \#), \$] \\
h(\#, [c, d], [e, f]) &= [\delta(\#, c, e), \delta(\#, c, f] \\
h(\#, [c, d], [e, \$]) &= [\delta(\#, c, e), \$] \\
h([a, b], [c, d], [e, f]) &= [\delta(a, c, e), \delta(a, c, f)] \\
h([a, b], [c, d], [e, \$]) &= [\delta(a, c, e), \$]
\end{aligned}
$$

For arguments where h is not specified above, h maps to some don't-care state D.

(b) C'' is merely a half-speed version of C'. ■

This lemma states that in a sense, rCA languages are closed under "doubling"; for $L \in rCA|_t$, the language $2L-1 = \{0^{2i-1} \mid 0^i \in L\}$ is also in $rCA|_t$.

THEOREM 5.2: $rrCA_1 \subseteqq llCA_0$.

Proof: Let L be an $rrCA_1$ language accepted by an $rCAC_1$ with oracle L', where L' is accepted by an $rCAC_2$. The machines of Lemma 5.1 can be used to find responses to all the oracle queries made by C_1. These responses must then be propagated down the array. This involves a delay; so C_2'' rather than C_2' is used. Thus at time $2i-1$, the response to the ith oracle query is available at the leftmost cell. So the ith transition of the leftmost cell of C_1 is also implemented at this cell now. Simultaneously, the oracle response is sent right at unit speed, so that the jth cell implements the ith transition step of C_1 at time $2i-1+j-1$. It is easily verified that at this time, if each cell stores the current and the previous value of the corresponding cell in the simulation of C_1, the arguments to the transition function are indeed available in the cell and its neighbours. An example is shown in figure 6. The oracle queries are answered by C_2'', a half-speed version of the CA C' shown in figure 5. For the behaviour of C_1 as in figure 6 (a), the simulating CA functions as in figure 6 (b), recognising the input in time $2n-1$.

Formally, the transition function of such a CA can be specified in terms of those of C_1 and C_2'' as follows.

Let

$$C_1=(Q_1, \#, \delta_1, \delta_2, L', F_1) \qquad \text{and} \qquad C_2''=(Q'', \#, \delta, F'').$$

Define a CA $C=(Q, \#, h, F)$ where

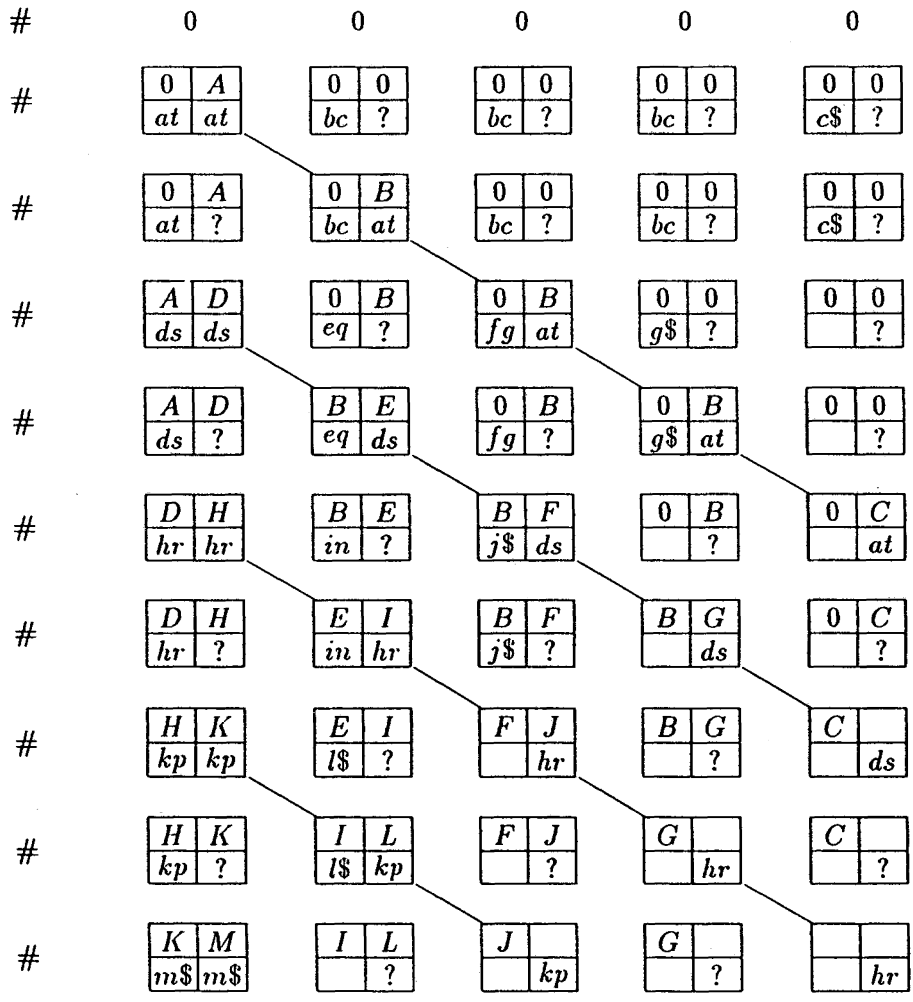
$$Q=\{[u, v, w, x] \mid u, v \in Q_1,\ w \in Q'' \text{ and } x \in Q'' \cup \{?\}\}$$

u and v hold the old and current states in the simulation of C_1. w holds the state in the simulation of C_2'' and is updated at each time step. The value of

#	0	0	0	0	0	#
#	A	B	B	B	C	
#	D	E	F	G		
#	H	I	J			
#	K	L				
#	M					

(a) $rrCA_1$ C on input length 5, with the oracle from Figure 5.

Figure 6.



(b) The simulating CA

Figure 6. —Simulating an $rrCA_1$ by an $llCA_0$

w in the leftmost cell is the response to the oracle query, and is propagated right at unit speed in x. When x holds a ?, u and v are not updated. When it holds a state from Q'', then v is stored in u and v is updated as in C_1, using the previous states u from the left neighbour and the current states v from the cell and its right neighbour. In figure 6 (b), a state $[u, v, w, x]$ is represented

as $\begin{array}{|c|c|}\hline u & v \\ \hline w & x \\ \hline \end{array}$

$$F=\{[u, v, w, x] \mid u \in Q_1,\ v \in F_1,\ w \in Q'' \text{ and } x \in Q'' \cup \{?\}\}$$

$h(\#, 0, z)=[0, A, B, C]$ for $z=0$ or $\#$, where
$B=\delta(\#, 0, z)$, $C=B$, and if $C \in F''$
then $A=\delta_1(\#, 0, z)$
else $A=\delta_2(\#, 0, z)$.

$h0, 0, z)=[0, 0, \delta(0, 0, z), ?]$ for $z=0$ or $\#$.

$h(\#, b, c)=[A, B, C, D]$ for 4-tuples b and c, where
$C=\delta(\#, b_3, c_3)$, and if $b_4 \neq ?$
then $A=b_1$, $B=b_2$, $D=?$
else $A=b_2$, $D=C$, and
if $C \in F''$
then $B=\delta_1(\#, b_2, c_2)$
else $B=\delta_2(\#, b_2, c_2)$.

$h(a, b, c)=[A, B, C, D]$ for 4-tuples a, b, c, where
$C=\delta(a_3, b_3, c_3)$, and if $a_4=?$
then $A=b_1$, $B=b_2$, $D=?$
else $A=b_2$, $D=a_4$, and
if $D \in F''$
then $B=\delta_1(a_1, b_2, c_2)$
else $B=\delta_2(a_1, b_2, c_2)$.

(If $c=\#$, then we take c_i to be $\#$ for $i=1$ to 4.) ■

THEOREM 5.3: $lr\,CA_1 = ll\,CA_0$.

Proof: $ll\,CA_0 \subseteq lr\,CA_1$ follows from Proposition 4.2 (*a*). $lr\,CA_1 \subseteq ll\,CA_0$ can be shown as above, packing c cells of C_2'' together to simulate C_2'' on input 0^{cn} within an n length array. ■

Note that in the above proofs, the crucial point is that the oracle classes contain only tally sets. The accepted language itself need not be tally; thus we can also conclude that $r\,CA\,(r\,CA\,|_t)$ is contained in $l\,CA$ which is equal to $l\,CA\,(r\,CA\,|_t)$. In other words, $r\,CA\,|_t$ is useless as an oracle class if the CA is allowed even as much as linear time.

Theorem 4.5 of the previous section now follows from the above two theorems, by a simple inductive argument.

Proof of Theorem 4.5: We know that $rr\,CA_0 \subseteq rr\,CA_1 \subseteq ll\,CA_0$. So if $rr\,CA_0 = ll\,CA_0$, then $rr\,CA_0 = rr\,CA_1$. Let $rr\,CA_k = rr\,CA_0$. $rr\,CA_{k+1}$ is the

class of languages accepted by $r\,CA$ using oracles from $rr\,CA_k$, *i.e.* oracles from $rr\,CA_0$, and so equals the class $rr\,CA_1$. But this is equal to $rr\,CA_0$, under our assumption. So $rr\,CA_{k+1} = rr\,CA_0$. Thus by induction, $\forall k$, $rr\,CA_k = rr\,CA_0$. From the definition, it then follows that $\forall k$, $lr\,CA_k = lr\,CA_1$ which equals $rr\,CA_0$ by assumption. The other classes are similarly shown to be equal to $rr\,CA_0$. Thus if $rr\,CA_0 = ll\,CA_0$, then the CAH collapses to the smallest class $rr\,CA_0 = r\,CA\,|_t$. ■

We now show that the results of Lemma 5.1 and Theorems 5.2 and 5.3 "translate upwards"; they also hold at higher levels of the CA hierarchy. The following lemma essentially states that Lemma 5.1 (*b*) relativises if the oracle classes are $rr\,CA_k$ classes. Thus all $rr\,CA_k$ classes are closed under doubling.

LEMMA 5.4: *If* $L \in rr\,CA_k$, *then* $2L-1 = \{0^{2i-1} \mid 0^i \in L\} \in rr\,CA_k$.

Proof: Consider $L \in rr\,CA_0$. Let L be accepted by $r\,CA\,C$. On input 0^{2m-1}, simulate machine C' described in Lemma 5.1, and also send a signal **S** at unit speed from the rightmost cell to the left. **S** reaches the accept cell when it is specifying membership of 0^m in L. So C'' will accept (reject) 0^{2m-1} if $0^m \in L$ ($0^m \notin L$), in time $2m-1$, *i.e.* in real time. So C'' is an $r\,CA$ accepting $2L-1$.

Assume that the statement of the lemma is true for k. Let $L \in rr\,CA_{k+1}$. L is accepted by an $r\,CA\,C$ using oracle $L' \in rr\,CA_k$. By our assumption, $2L'-1 \in rr\,CA_k$. Construct C'' as above, using oracle $2L'-1$. The resulting CA accepts $2L-1$ in real time; hence $2L-1 \in rr\,CA_{k+1}$. So the statement of the lemma is also true for $k+1$.

Thus by induction, the statement is true for all k. ■

Since $ll\,CA_0 = lr\,CA_0$, in Theorem 5.2 we are essentially proving that $rr\,CA_1 \subseteq lr\,CA_0$. Using the above lemma, this generalises as follows.

THEOREM 5.5: *For* $k \geq 0$, $rr\,CA_{k+1} \subseteq lr\,CA_k$.

Proof: For $k=0$, this is proved in Theorem 5.2. Consider $k>0$. Let C be an $rr\,CA_{k+1}$ CA using oracle L. $L \in rr\,CA_k$, so L is accepted by an $r\,CA$ using oracle $L' \in rr\,CA_{k-1}$. Now $2L'-1$ is also in $rr\,CA_{k-1}$. A CA using oracle $2L'-1$ can accept the same language as C, in linear time, as described in Theorem 5.2. But this $l\,CA$ uses an oracle from $rr\,CA_{k-1}$, and hence will belong to $lr\,CA_k$. Hence the theorem. ■

Similarly, reading Theorem 5.3 as $lr\,CA_1 = lr\,CA_0$ and translating it upwards in an identical fashion, we get

THEOREM 5.6: *For* $k \geq 0$, $lr\,CA_{k+1} \subseteq lr\,CA_k$.

This, along with Proposition 4.2, immediately yields

COROLLARY 5.7: $\forall k \geqq 0$, $lr\,CA_k = l\,CA|_t$.

This corollary clearly generalises Theorem 5.3; not only are $r\,CA|_t$ (*i.e.* $rr\,CA_0$) languages useless as oracles for the class $l\,CA$, but so are all languages in the classes $rr\,CA_k$, for any k. Thus the $rr\,CA_k$ languages seem to be quite limited in their power.

Now we can combine all these known results to obtain an overall picture of the CA hierarchy. The structure of the cellular automata hierarchy is as shown in figure 7.

The following series of propositions shows how this structure changes under various assumptions of equality of certain classes.

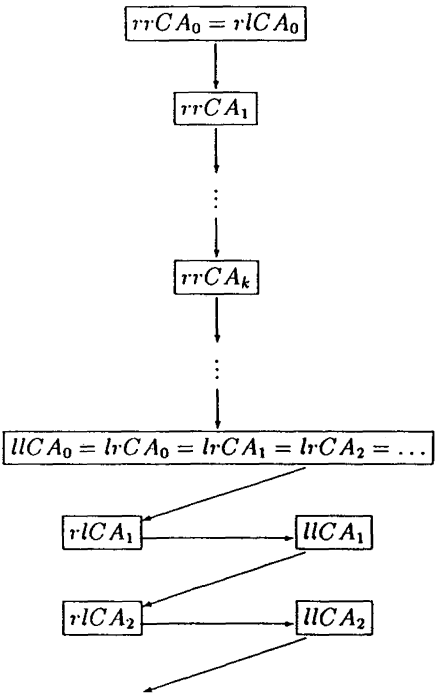


Figure 7. – The structure of the CAH

PROPOSITION 5.8: *If* $rr\,CA_0 = rr\,CA_1$, *then all the* $rr\,CA$ *classes are equal.*

Proof: Obvious, as seen in proof of Theorem 4.5. ■

PROPOSITION 5.9: *$rr\,CA_1 = lr\,CA_1$ if and only if $rr\,CA_1 = rl\,CA_1$. In this case, the structure in figure 8 results.*

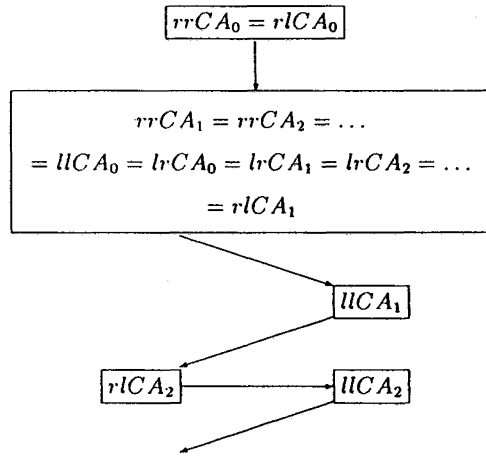


Figure 8. – The CAH, assuming $rr\,CA_1 = lr\,CA_1$

Proof: From figure 7, it is obvious that $rr\,CA_1 = rl\,CA_1$ implies $rr\,CA_1 = lr\,CA_1$. Assume that $rr\,CA_1 = lr\,CA_1$. From figure 7, we see that this implies $rr\,CA_1 = rr\,CA_2 = ll\,CA_0$. So

$$rl\,CA_1 = r\,CA\,(ll\,CA_0)\,|_t = r\,CA\,(rr\,CA_1)\,|_t = rr\,CA_2 = rr\,CA_1.$$

Further, since under this assumption we have $rr\,CA_2 = rr\,CA_1$, it is clear that $\forall k > 0, rr\,CA_k = rr\,CA_1$. ■

$rr\,CA_1 = rl\,CA_1$ means that $r\,CA\,|t$ and $l\,CA\,|_t$ as oracles add equally to the class $r\,CA\,|_t$. $rr\,CA_1 = lr\,CA_1$ means that $\mathrm{r\,CA}\,|_t$ and $l\,CA\,|_t$ coincide relative to the class of oracles $r\,CA\,|_t$. Equivalently, since $l\,CA\,(r\,CA\,|_t) = l\,CA$, this also means that with an $r\,CA$ oracle, the class $r\,CA\,|_t$ rises up to equal $l\,CA\,|_t$. These equalities imply each other and also imply that the $rr\,CA_k$ classes are not distinct for $k > 0$.

PROPOSITION 5.10: *If $rr\,CA_1 = ll\,CA_1$, then the cellular automata hierachy has only two distinct classes: $r\,CA\,|_t = rr\,CA_0 = rl\,CA_0$, and $l\,CA\,|_t$, which is equal to all the remaining classes.*

Proof: $rr\,CA_1 = ll\,CA_1$ clearly implies $rr\,CA_1 = lr\,CA_1$. So from the above proposition we immediately conclude that $\forall k \geqq 1$, $rr\,CA_k = rr\,CA_1$. Further, since $ll\,CA_1 = ll\,CA_0$, $\forall k \geqq 0$ $ll\,CA_k = ll\,CA_0$. This also implies that $\forall k \geqq 1$,

$rl\,CA_k = ll\,CA_0$. Thus $rr\,CA_0$ and $rl\,CA_0$ are identical, and all other classes are identical; the CAH has at most two distinct classes. ■

PROPOSITION 5.11: *$lr\,CA_1 = ll\,CA_1$ if and only if $ll\,CA_0 = ll\,CA_1$. In this case, the CAH has the structure shown in figure* 9. ■

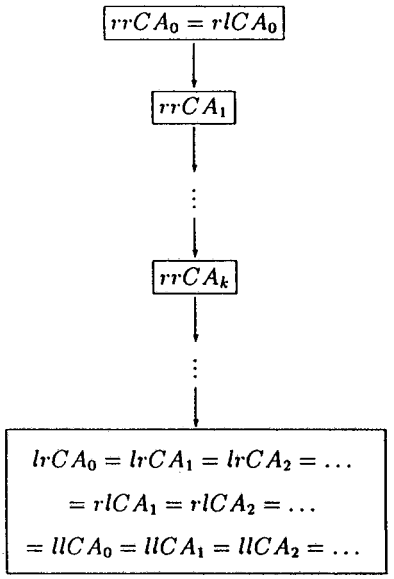


Figure 9. – The CAH, assuming $lr\,CA_1 = ll\,CA_1$

Proof: Obvious.

A point worth examining is whether proper containments translate upwards. Equalities do; we have, by a straightforward argument,

$$rr\,CA_k = rr\,CA_{k+1} \Leftrightarrow \forall m > k, \quad rr\,CA_m = rr\,CA_k$$
$$ll\,CA_k = ll\,CA_{k+1} \Leftrightarrow \forall m > k, \quad ll\,CA_m = ll\,CA_k$$

6. CONCLUSIONS

This work attempts to study the structure of the tally languages, if any, separating CA, $l\,CA$ and $r\,CA$. We have also restricted the language classes CA, $l\,CA$ and $r\,CA$ to tally sets. A similar hierarchy of CA language classes can be constructed if non-tally languages are considered for acceptance and as oracles, within the framework of TVCA (*see* [Mah92]). It is easily

verified that Proposition 4.2 (*a*), (*c*), (*d*) continue to hold. (*b*) does not appear to, because once we consider non-tally oracles, the containment $A \in l\,CA\,(A)$ does not necessarily hold. [$A \in CA\,(A)$ does hold, from Proposition 3.2, since on input $x = \langle m \rangle$, m is CA-time-constructible; refer Proposition 3.12.] A straightforward algorithm for recognising an $ll\,CA_1$ language (non-tally oracle) by a CA requires $O(n \log n)$ time, while for tally oracles such an algorithm runs in $O(n^2)$ time. However, even for $rr\,CA_1$ languages with non-tally oracles we have been unable to improve the $O(n \log n)$ upper bound, whereas $rr\,CA_1$ languages with tally oracles can be accepted by $l\,CA$; refer Theorem 5.2. Of course, such differences are to be expected, since tally sets are very low in information content.

An unanswered question is whether or not $r\,CA$ and $l\,CA$ coincide over unary alphabets. If this is the case, then Theorem 4.5 states that the CA hierarchy collapses. In [IJ88] it is conjectured that these classes do not coincide. Our work is motivated by a weaker conjecture – namely, that if the classes $r\,CA$ and $l\,CA$ are distinct, then there are tally sets in the difference.

Another aspect which deserves more study is finding languages complete for CA and $l\,CA$, where completeness will have to be suitably defined. Such complete languages may admit a relativisation under which the classes $r\,CA$, $l\,CA$ and CA coincide. This, along with Theorem 3.5, will provide a contradictory relativisation of these problems, but may also provide more information about the nature of the CA complexity classes.

REFERENCES

[BC84] W. Bucher and K. Culik II, On real-time and linear-time cellular automata, *R.A.I.R.O. Informatique théorique*, 1984, *18*, pp. 307-325.

[BDG88] J. L. Balcázar, J. Díaz and J. Gabarró, Structural Complexity I, volume 11 of *EATCS Monograph Series*, Springer-Verlag, Berlin, 1988.

[BDG90] J. L. Balcázar, J. Díaz and J. Gabarró, Structural Complexity II, volume 22 of *EATCS Monograph Series*, Springer-Verlag, Berlin, 1990.

[Boo74] R. V. Book, Tally languages and complexity classes, *Information and Control*, 1974, *26*, pp. 186-193.

[CC84] C. Choffrut and K. Culik II, On real-time cellular automata and trellis automata, *Acta Informatica*, 1984, *21*, pp. 393-409.

[CGS84] K. Culik II, J. Gruska and A. Salomaa, Systolic trellis automata Part I. *International J. of Computer Mathematics*, 1984, *15*, pp. 195-212.

[CIV88] J. H. Chang, O. H. Ibarra and A. Vergis, On the power of one-way communication. *J. of the ACM*, 1988, *35*, pp. 697-726.

[Dye80] C. Dyer, One-way bounded cellular automata, *Information and Control*, 1980, *44*, pp. 261-281.

[IJ87] O. H. Ibarra and T. Jiang, On one-way cellular arrays, *SIAM J. of Computing*, 1987, *16* pp. 1135-1154.

[IJ88] O. H. IBARRA and T. JIANG, Relating the power of cellular arrays to their closure properties, *Theoretical Computer Science*, 1988, *57*, p. 225-238.

[IPK85] O. H. IBARRA, M. PALIS and S. M. KIM, Some results concerning linear iterative (systolic) arrays, *J. of Parallel and Distributed Computing*, 1985, *2*, pp. 182-218.

[Mah92] M. MAHAJAN, Studies in Language Classes Defined by Different Types of Time-Varying Cellular Automata, *Ph. D. Thesis*, Indian Institute of Technology, Madras, India, 1992.

[MK91] M. MAHAJAN and K. KRITHIVASAN, Relativised cellular automata and complexity classes. In *Proceedings of the 11th International FST&TCS Conference*, New Delhi, December 1991, LNCS 560, pp. 172-185.

[MK92] M. MAHAJAN and K. KRITHIVASAN, Some results on time-varying and relativised cellular automata, *International J. of Computer Mathematics*, 1992, *43*, pp. 21-38.

[Smi71] A. R. SMITH III, Cellular automata complexity trade-offs, *Information and Control*, 1971, *18*, pp. 466-482.

[Smi72] A. R. SMITH III, Real-time language recognition by one-dimensional cellular automata, *J. of Computer and System Sciences*, 1972, *6*, pp. 233-253.

APPENDIX

Proof of Theorem 3.8

A language $L \subseteq \{0\}^*$ is said to be ultimately periodic if there exist natural numbers $n_0 \geq 0$ and $p \geq 1$ such that

$$\forall n \geq n_0, \quad 0^{n+p} \in L \quad \textit{if and only if} \quad 0^n \in L$$

Clearly, a *CA* which is not time-varying can be considered as an ultimately periodic *TVCA* with $n_0 = 0$ and $p = 1$.

Let $C = (Q, \#, \delta_1, \delta_2, X, A)$ be an ultimately periodic *TVCA*. n_0 and p are as in the definition above.

Construct $C' = (Q', \#, \delta', A')$ as follows.

Define sets Q_i, $i = 2$ to $n_0 + p - 1$, to be disjoint copies of Q; thus $Q_i = \{q_i \mid q \in Q - \{\#\}\}$. Then $Q' = \left(\bigcup_{i=2}^{n_0+p-1} Q_i \right) \cup Q$. Let A_i be the restriction of Q_i to copies of states in A; then $A' = \left(\bigcup_{i=2}^{n_0+p-1} A_i \right) \cup A$. δ' is defined as follows:

$\delta'(a, b, c) = q_2$ where $q = \delta(a, b, c, 1)$ and $b \neq \#$.

$\delta'(a_i, b_i, c_i) = q_{i+1}$ where $q = \delta(a, b, c, i)$ and $i < n_0 + p - 1$.

$\delta'(a_i, b_i, c_i) = q_j$ where $i = n_0 + p - 1$ and $q = \delta(a, b, c, i)$ and $j = n_0$.

$\delta'(a, \#, b) = \#$ for any $a, b \in Q'$.

It is easy to see that C' so defined simulates C and that the simulation is step-for-step, *i.e.*, with no loss of time.

Essentially, $n_0 + p$ copies of the state set are created, and the transition function depends on which copy of the state is an argument. Copies of the boundary state need not be created. Accepting states can be suitably defined.

Proof of Proposition 3.9

In [CGS84, CC84] it has been shown that all regular languages can be accepted by *r OCA*. To see the converse when restricted to tally sets, let $C = (Q, \#, \delta, A)$ be an *r OCA* accepting a tally language L. We can construct a finite-state machine M accepting L as follows: $M = (q_0 \cup (Q \times Q), \{0\}, \delta', q_0, F)$ where

$$\delta'(q_0, 0) = [\delta(\#, 0), \delta(0, 0)],$$
$$\delta'([a, b], 0) = [\delta(a, b), \delta(b, b)],$$

and

$$F = \{[a, b] \mid a \in A\}.$$

For instance, if the cells of C change states as shown in Figure 10, then M goes through states q_0, AB, CD, EF, GH, ...

#	O	O	O	O	O
	A	B	B	B	B
		C	D	D	D
			E	F	F
				G	H
					I

Figure 10. – The unrolling of an *r OCA*