I F I G
RESEARCH
REPORT

INSTITUT FÜR INFORMATIK

ON TALLY LANGUAGES AND GENERALIZED INTERACTING AUTOMATA

Thomas Buchholz Andreas Klein
Martin Kutrib

IFIG RESEARCH REPORT 9902
FEBRUARY 1999

Institut für Informatik
JLU Gießen
Arndtstraße 2
D-35392 Giessen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

IFIG Research Report
IFIG Research Report 9902, February 1999

On tally languages and generalized interacting automata

Thomas Buchholz[1] Andreas Klein
Martin Kutrib[2]

Institute of Informatics, University of Giessen
Arndtstr. 2, D-35392 Giessen, Germany

Abstract. Devices of interconnected parallel acting sequential automata are investigated from a language theoretic point of view. Starting with the well-known result that each tally language acceptable by a classical one-way cellular automaton (OCA) in real-time has to be a regular language we will answer the three natural questions 'How much time do we have to provide?' 'How much power do we have to plug in the single cells (i.e., how complex has a single cell to be)?' and 'How can we modify the mode of operation (i.e., how much nondeterminism do we have to add)?' in order to accept non-regular tally languages.
We show the surprising result that for some classes of generalized interacting automata parallelism does not lead to more accepting power than obtained by a single sequential cell.
Adding a wee bit of nondeterminism an infinite hierarchy of unary language families can be shown by allowing more and more nondeterminism.

CR Subject Classification (1998): F.1, F.4.3, B.6.1, E.4

[1]E-mail: buchholz@informatik.uni-giessen.de
[2]E-mail: kutrib@informatik.uni-giessen.de

1 Introduction

Devices of interconnected parallel acting automata have extensively been investigated from a language theoretic point of view. The specification of such a system includes the type and specification of the single automata, the interconnection scheme (which sometimes implies a dimension to the system), a local and/or global transformation and the input and output modes. One-dimensional devices with nearest neighbor connections whose cells are deterministic finite automata are commonly called cellular automata (CA) resp. iterative arrays (IA) in case of parallel resp. sequential input mode. One-way information flow is indicated by the notion OCA resp. OIA. The family of languages acceptable by e.g., CAs in real-time (linear-time) is denoted by $\mathscr{L}_{rt}(\text{CA})$ ($\mathscr{L}_{lt}(\text{CA})$) and the corresponding subfamily of tally (unary) languages by $\mathscr{L}_{rt}^{u}(\text{CA})$ ($\mathscr{L}_{lt}^{u}(\text{CA})$).

There are several important open problems concerning the relations between the various families. The tally languages are playing an important role in investigations on that field.

Under unary restriction several essentially different families are identical (e.g., $\mathscr{L}_{rt}(\text{IA}) \subset \mathscr{L}_{rt}(\text{CA})$ [6, 20] but $\mathscr{L}_{rt}^{u}(\text{IA}) = \mathscr{L}_{rt}^{u}(\text{CA})$ [18] or $\mathscr{L}_3 \subset \mathscr{L}_{rt}(\text{OCA})$ [15] but $\mathscr{L}_3^u = \mathscr{L}_{rt}^{u}(\text{OCA})$ [19], where $\mathscr{L}_3$ denotes the regular languages), from which follows that the capabilities of some devices become noticable not for tally languages.

There are incomparable families that become comparable (e.g., $\mathscr{L}_{rt}(\text{OCA})$ is not comparable to $\mathscr{L}_{rt}(\text{IA})$ [7] but $\mathscr{L}_{rt}^{u}(\text{OCA}) \subset \mathscr{L}_{rt}^{u}(\text{IA})$ [4]), which means that the restriction affects the families differently.

Moreover, for unary families some general open properties are known (e.g., $\mathscr{L}_{rt}^{u}(\text{CA})$ is closed under concatenation [12]) and some others are still open (e.g., whether or not $\mathscr{L}_{rt}^{u}(\text{CA}) = \mathscr{L}_{lt}^{u}(\text{CA})$).

From these examples one can obtain that there is no general rule for what happens if we go from arbitrary to unary languages.

Though any language can be unarily encoded there are differences in time and space complexity. The tally languages have been investigated in several works on the field of sequential automata and complexity theory (e.g., [1, 2, 5, 8, 11]).

Tally languages serve in many proofs as (counter-)examples. E.g., in the past the only languages known not to belong to $\mathscr{L}_{rt}(\text{OCA})$ had been the non-regular unary ones. If one intend to accept non-regular tally languages by parallel one-way devices at least three natural questions arise: How much time do we have to provide for OCAs? How much power do we have to plug in the single cells? and How can we modify the mode of operation?

The paper is organized as follows: In section 2 we define the basic notions and the model in question. Thereby the parallel model is derived from the definition of general sequential machines that are actually the single cells.

In order to answer the second question in section 3 we generalize the result $\mathscr{L}_3^u = \mathscr{L}_{rt}^{u}(\text{OCA})$ to OCAs whose cells are much more complex, and draw a stripline to cells that give OCAs the power to accept non-contextfree unary languages. A consequence is that for these specific devices parallelism does not lead to more powerful acceptance compared to just one single sequential cell.

Section 4 is devoted to the first question. We show that there exists a complexity class between the real-time and linear-time OCA languages: $\mathscr{L}_{rt}(\mathrm{OCA}) \subset \mathscr{L}_{rt+\log}(\mathrm{OCA}) \subseteq \mathscr{L}_{(1+\epsilon)\cdot rt}(\mathrm{OCA})$. Moreover, there is a gap between $\mathscr{L}^u_{rt}(\mathrm{OCA})$ and $\mathscr{L}^u_{rt+\log}(\mathrm{OCA})$. Thus (in terms of tally languages) that at least an amount of a logarithmic number of time steps have to be added to increase the capabilities of real-time one-way cellular arrays.

Allowing some kind of restricted nondeterminism we prove in section 5 that a slight increase of nondeterminism as well as adding two-way communication reduces the time complexity from linear-time to real-time. These results might be used to show an infinite hierarchy of unary language families allowing more and more nondeterminism from which an answer to the third question follows.

2 Models and Definitions

We denote the integers by $\mathbb{Z}$, the positive integers $\{1, 2, \dots\}$ by $\mathbb{N}$ and the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$. $X_1 \times \dots \times X_d$ denotes the Cartesian product of the sets $X_1, \dots, X_d$. If $X_1 = \dots = X_d$ we also write X_1^d. If $(x_1, \dots, x_d)$ is a d-tuple $\pi_i(x_1, \dots, x_d) = x_i$ is the projection to the ith component.

The empty word is denoted by ε and the reversal of a word w by w^R. Similarly the notions L^R and $\mathscr{L}^R$ for languages and families of languages are used. Two languages L and L' are considered to be equal if they differ at most by the empty word, i.e., $L \setminus \{\varepsilon\} = L' \setminus \{\varepsilon\}$. If $\mathscr{L}$ is a family of languages we denote by $\mathscr{L}^u$ the subfamily of tally (or unary) languages, i.e., the languages over a singleton.

2.1 Sequential machines

A sequential machine is basically a memory augmented nondeterministic finite automaton. Its state transition depends on the actual state, actual input symbol, and additionally on the actual memory content. The memory access is limited such that the finite control gets only a partial (finite) memory information although the memory itself has an infinite capacity. On the other hand the memory content is updated whereby no memory access limitation is made.

Before defining a sequential machine formally, we introduce a fixed ordered decidable infinite set $V := \{m_1, m_2, \dots\}$ from which a finite number of memory symbols have to be chosen. Certainly, this is not a real restriction. However since later we are interested in special subfamilies of sequential machines where the memory updating and memory access is restricted we avoid by this way technical difficulties.

Definition 1 A *sequential machine* (SM) is a system (S, M, D, s_0, A, F), where

a) S is the finite, nonempty set of *states*,
b) $M \subset V$ is the finite set of *memory symbols*,
c) $s_0 \in S$ is the *initial state*,
d) A is the finite, nonempty set of *input symbols*,
e) $F \subseteq S$ is the set of *accepting states*, and
f) D is the finite, nonempty set of *local transformations*. Each of which is a triple (δ, μ, α) where
   - $\delta : S \times (M \cup \{\varepsilon\}) \times A \to S$ is a *state transition function*,
   - $\mu : S \times M^* \times A \to M^*$ is a computable *memory update function*, and
   - $\alpha : M^* \to (M \cup \{\varepsilon\})$ is a computable *memory access function*.

Observe that in case of $M = \emptyset$ the memory access function is uniquely determined.

The operation of a SM is now formalized in terms of configurations and a global transformation function. A configuration of a SM is a description of its global state which is actually an element of $S \times M^* \times A^*$. During its course of computation a SM nondeterministically steps through a sequence of configurations. The initial configuration is defined by (s_0, ε, w) where w is an input word, while subsequent configurations are chosen according to the *global transformation* Δ: Let $c = (s, u, w)$ and $c' = (s', u', w')$ be two configurations such that $w \neq \varepsilon$. Then

$$c' \in \Delta(c) \iff \exists\, (\delta, \mu, \alpha) \in D : s' = \delta(s, \alpha(u), a) \wedge u' = \mu(s, u, a) \wedge w' = x$$

where $w = ax$ for some $a \in A$ and $x \in A^*$.

The i-fold composition of Δ is defined as follows:

$$\begin{aligned} \Delta^0(c) &:= \{c\} \\ \Delta^{i+1}(c) &:= \bigcup_{c' \in \Delta^i(c)} \Delta(c') \end{aligned}$$

where $0 \leq i < |w|$.

Definition 2 Let $\mathcal{M}$ be a SM. Then

$$L(\mathcal{M}) := \{w \in A^* \mid \exists\, c \in \Delta^{|w|}(s_0, \varepsilon, w) : \pi_1(c) \in F\}$$

is the language *accepted* by $\mathcal{M}$.

A SM is *deterministic* if $|D| = 1$. Deterministic sequential machines are denoted by DSM.

Since especially the memory update function is not subject to any restrictions except for being computable the family of languages acceptable in a SM (as well as in a DSM) is the well-known family of recursive languages. However sequential machines should rather serve as a general framework to consider various (more familiar) sequential models simultaneously. Therefore we define subfamilies of the sequential machines depending on two predicates restricting the corresponding memory functions.

Definition 3 Let P and Q be two decidable predicates each of which relates two words over V where $P(u,u)$ holds for all $u \in V^*$. The family of all SM resp. DSM fulfilling

$$\forall\,(\delta,\mu,\alpha) \in D, \forall\, s \in S, \forall\, u \in M^*, \forall\, a \in A : P(u,\mu(s,u,a)) \wedge Q(u,\alpha(u))$$

is denoted by $SEQ(P,Q)$ resp. $DSEQ(P,Q)$.

Observe that the assertion on P is a very natural one: sequential machines in $SEQ(P,Q)$ resp. $DSEQ(P,Q)$ need not to change their memory content.

By supplying suitable predicates we may obtain well known subfamilies of SM.

Example 4 Let $P(u,v)$ iff $u = v$ and let $Q(u,v)$ iff $v = \varepsilon$, then $NFA := SEQ(P,Q)$ resp. $DFA := DSEQ(P,Q)$ denotes the family of nondeterministic resp. deterministic finite automata.

Example 5 Let $P(u,v)$ iff $u = \varepsilon$ or $u = ay$ and $v = xy$ where $a \in V$ and $x,y \in V^*$ and let $Q(u,v)$ iff $u = v = \varepsilon$ or $u = ay$ and $v = a$ where $a \in V$ and $y \in V^*$. Then $PDA := SEQ(P,Q)$ resp. $DPDA := DSEQ(P,Q)$ is the family of pushdown automata resp. deterministic pushdown automata without ε-moves.

In the sequel SEQ always denotes a subfamily of SM resp. DSM which is induced by some two predicates P and Q, i.e., $SEQ = SEQ(P,Q)$ resp. $SEQ = DSEQ(P,Q)$. The family of all languages which can be accepted by a SEQ (i.e., by some sequential machine in SEQ) is denoted by $\mathscr{L}(SEQ)$.

Observe that the sequential machines have been designed to meet the requirements a model has to fulfill such that one can easily construct an interconnected and interacting array of such machines. Therefore we did not introduce the capability to perform transitions without consuming input symbols. Further it is adequate to allow real-time computations only (i.e., the length of the input determines the number of transitions) although the model can easily be extended to operate beyond real-time.

2.2 Cellular machines

A cellular machine is now an infinite linear array of identical SM, sometimes called cells, each of them is connected to its both immediate neighbors to the left and to the right. For our convenience we identify the cells by integers. The state transition as well as the memory updating of each cell depends on its actual state and memory content and the actual states of its neighbors. More precisely at discrete time steps *one* local transformation is nondeterministically choosen and applied to all the cells synchronously. Formally:

Definition 6 A *cellular machine* (CM) is a system $(S, M, D, s_0, \#, F, A)$, where

a) $(S, M, D, s_0, S \times S, F)$ is a sequential machine,

b) $\# \in S$ is the *border state* satisfying

$$\forall\, s \in S, \forall\, m \in M \cup \{\varepsilon\}, \forall\, p \in S \times S : \quad s = \# \iff \delta(s, m, p) = \#$$

and

$$\forall\, s \in S, \forall\, u \in M^*, \forall\, p \in S \times S : \quad s = \# \longrightarrow \mu(s, u, p) = u,$$

c) $A \subseteq S$ is the finite, nonempty set of *input symbols*.

A configuration of a cellular machine at some time $t \geq 0$ is a mapping $c_t : \mathbb{Z} \to S \times M^*$ giving the actual state and memory content of the single cells. The initial configuration $c_{w,0}$ at time 0 is defined by the input word $w = a_1 \cdots a_n \in A^+$:

$$c_{w,0}(i) = (a_i, \varepsilon) \text{ if } 1 \leq i \leq n \quad \text{and} \quad c_{w,0}(i) = (\#, \varepsilon) \text{ otherwise.}$$

Similarly, subsequent configurations are choosen accordingly to the *global transformation* Δ:

Let c and c' be two configurations with $c(i) = (s_i, u_i)$ and $c'(i) = (s'_i, u'_i)$. Then

$$\begin{aligned} c' \in \Delta(c) \iff \exists\, (\delta, \mu, \alpha) \in D : s'_i &= \delta(s_i, \alpha(u_i), (\pi_1(c(i-1)), \pi_1(c(i+1)))) \wedge \\ u'_i &= \mu(s_i, u_i, (\pi_1(c(i-1)), \pi_1(c(i+1)))) \end{aligned}$$

for all $i \in \mathbb{Z}$.

If the state set is a Cartesian product of some smaller sets $S = S_1 \times \ldots \times S_r$, we will use the notion *register* for the single parts of a state. The concatenation of a specific register of all cells forms a *track*.

Definition 7 Let $\mathcal{M}$ be a CM.

a) A word w is *accepted* by $\mathcal{M}$ iff there exists a (smallest) time step $t_w \geq 1$ such that there exists a $c \in \Delta^{t_w}(c_{w,0})$ with $\pi_1(c(1)) \in F$.

b) $L(\mathcal{M}) = \{w \mid w \text{ is accepted by } \mathcal{M}\}$ is the language *accepted* by $\mathcal{M}$.

c) Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping. If all $w \in L(\mathcal{M})$ are accepted within $t_w \leq t(|w|)$ time steps, then $L(\mathcal{M})$ is said to be of *time complexity* t.

If SEQ is a subfamily of the sequential machines then C SEQ denotes the corresponding family of cellular machines. The family of all languages which can be accepted by a CSEQ with time complexity t is denoted by $\mathscr{L}_{t(n)}(\mathrm{C}\,SEQ)$. If t equals the *identity function* $id(n) := n$ acceptance is said to be in real-time and we write $\mathscr{L}_{rt}(\mathrm{C}\,SEQ)$. The linear-time languages $\mathscr{L}_{lt}(\mathrm{C}\,SEQ)$ are defined according to

$$\mathscr{L}_{lt}(\mathrm{C}\,SEQ) := \bigcup_{k \in \mathbb{N}} \mathscr{L}_{k \cdot n}(\mathrm{C}\,SEQ)$$

If the flow of information in a cellular machine is restricted only to leftwards the resulting device is a *one-way cellular machine* (OCM). I.e., the behavior of a cell is independent of its left immediate neighbor.

Usually we write CA resp. OCA for C DFA resp. OC DFA and denote the corresponding models *cellular automata* resp. *one-way cellular automata*. In connection with CA resp. OCA we often omit the memory augmentation such that they are considered to be merely systems $(S, \delta, \#, F, A)$ where δ maps from $S \times S \times S$ resp. from $S \times S$ to S.

3 Cells beyond finite automata

It is known that $\mathscr{L}(DFA) \subset \mathscr{L}_{rt}(\text{OCA})$ [15] but $\mathscr{L}^u(DFA) = \mathscr{L}^u_{rt}(\text{OCA})$ [19]. I.e., although the cells in an OCA are an instance of one finite automaton their interaction yield capabilities which exceeds those of a single finite state machine. On the other side the restriction to tally languages collapses that advantage.

A similar result is known for pushdown cellular automata: translated in our notion it has been shown in [17] that $\mathscr{L}(DPDA) \subset \mathscr{L}_{rt}(\text{OC}\ DPDA)$ but $\mathscr{L}^u(DPDA) = \mathscr{L}^u_{rt}(\text{OC}\,DPDA)$.

The aim of the present section is to show that it is an intrinsic property of real-time one-way cellular arrays that their computational power becomes effective for non unary languages only.

First, we will show that $\mathscr{L}(SEQ)$ is contained in $\mathscr{L}_{rt}(\text{OC}\,SEQ)$. Clearly, this inclusion is not a proper one in general since trivially $\mathscr{L}(\text{SM}) = \mathscr{L}_{rt}(\text{OCM})$.

Theorem 8 Let SEQ be a subfamily of SM. Then it holds

$$\mathscr{L}(SEQ) \subseteq \mathscr{L}_{rt}(\text{OC}\,SEQ)$$

Proof. Let P and Q be two predicates such that $SEQ = SEQ(P, Q)$. Further let $\mathcal{M} = (S, M, D, s_0, F, A)$ be some SEQ where $D = \{(\delta_i, \mu_i, \alpha_i) \mid 1 \leq i \leq k\}$ for a constant number $k \in \mathbb{N}$.

The construction of an OC SEQ $\mathcal{M}' = (S', M', D', \#, F', A')$ which accepts $L := L(\mathcal{M})$ in real-time follows a simple idea:

The input word is symbol-wise shifted to the left through the cells. Each cell fetching such a symbol simulates a corresponding transition of $\mathcal{M}$. Since $\mathcal{M}$ works in real-time by definition a mechanism has to be provided to rule out acceptance of words beyond real-time in $\mathcal{M}'$. Hence a cell simply stops the simulation if it fetches information from the rightmost cell which can identify itself. Altogether the leftmost cell simulates $\mathcal{M}$ on its whole input and thus $\mathcal{M}'$ can be designed to be a real-time acceptor of L.

However, since SEQ is some abstract subfamily of SM it might be possible that the SM we have to plug into the cells of $\mathcal{M}'$ in order to achieve the described tasks does not belong to SEQ. Fortunately, the assertion (on P) that a cell need not to change its memory content resolves this problem.

Formally $\mathcal{M}'$ can be constructed as follows.

$\#$ is a new symbol which belongs neither to A nor to $S \times A$.
$S' := S \times (A \cup \{\#\}) \cup A \cup \{\#\}, \quad M' := M, \quad A' := A, \quad F' := \{(s, \#) \mid s \in F\}$
$D' := \{(\delta'_i, \mu'_i, \alpha'_i) \mid 1 \leq i \leq k\}$ where $\forall\, i \in \{1, \ldots, k\}$:

$$\alpha'_i \;:=\; \alpha_i$$

$\forall\, s' \in A, \forall\, t' \in A \cup \{\#\}, \forall\, m' \in M \cup \{\varepsilon\}$:

$$\begin{aligned} \delta'_i(\#, m', t') &:= \# \\ \delta'_i(s', m', t') &:= (\delta_i(s_0, m', s'), t') \end{aligned}$$

$\forall\, s'_1 \in S, \forall\, s'_2 \in A, \forall\, (t'_1, t'_2) \in S \times (A' \cup \{\#\}), \forall\, m' \in M \cup \{\varepsilon\}$:

$$\begin{aligned} \delta'_i((s'_1, \#), m', (t'_1, t'_2)) &:= (s'_1, \#) \\ \delta'_i((s'_1, s'_2), m', (t'_1, t'_2)) &:= (\delta_i(s'_1, m, s'_2), t'_2) \end{aligned}$$

$\forall\, s' \in A, \forall\, t \in A \cup \{\#\}, \forall\, u' \in M^*$:

$$\begin{aligned} \mu'_i(\#, u', t') &:= u' \\ \mu'_i(s', u', t') &:= \mu_i(s_0, u', s') \end{aligned}$$

$\forall\, s'_1 \in S, \forall\, s'_2 \in A, \forall\, (t'_1, t'_2) \in S \times (A' \cup \{\#\}), \forall\, u' \in M^*$:

$$\begin{aligned} \mu'_i((s'_1, \#), u', (t'_1, t'_2)) &:= u' \\ \mu'_i((s'_1, s'_2), u', (t'_1, t'_2)) &:= \mu_i(s'_1, u', s'_2) \end{aligned}$$

□

For the converse we have to restrict to tally languages:

Lemma 9 Let SEQ be a subfamily of SM. Then it holds

$$\mathscr{L}^u_{rt}(\mathrm{OC}\,SEQ) \subseteq \mathscr{L}^u(SEQ)$$

Proof. Let L be some tally language in $\mathscr{L}^u_{rt}(\mathrm{OC}\,SEQ)$ over the alphabet $A = \{\mathtt{a}\}$. Denote by $\mathcal{M} = (S, M, D, \#, F, A)$ an OC SEQ which is a real-time acceptor of L where $D = \{(\delta_i, \mu_i, \alpha_i) \mid 1 \leq i \leq k\}$ for some constant number $k \in \mathbb{N}$. W.l.o.g. we may assume that a cell of $\mathcal{M}$ which once enters an accepting state remains in an accepting state (eventually we have to introduce some new states).

On input $w = \mathtt{a}^n$ we make two observations (cf. figure 1):

At every time step one of the local transformations is applied to all cells. Thus two cells having the same state and memory content act equally if they obtain the same state information from their corresponding right neighbors. Therefore $c_1(1) = \ldots = c_1(n-1)$ since $c_{w,0}(1) = \ldots = c_{w,0}(n) = (\mathtt{a}, \varepsilon)$. And similarly we have for all time steps t with $0 \leq t \leq n-1$: $c_t(1) = \ldots = c_t(n-t)$.

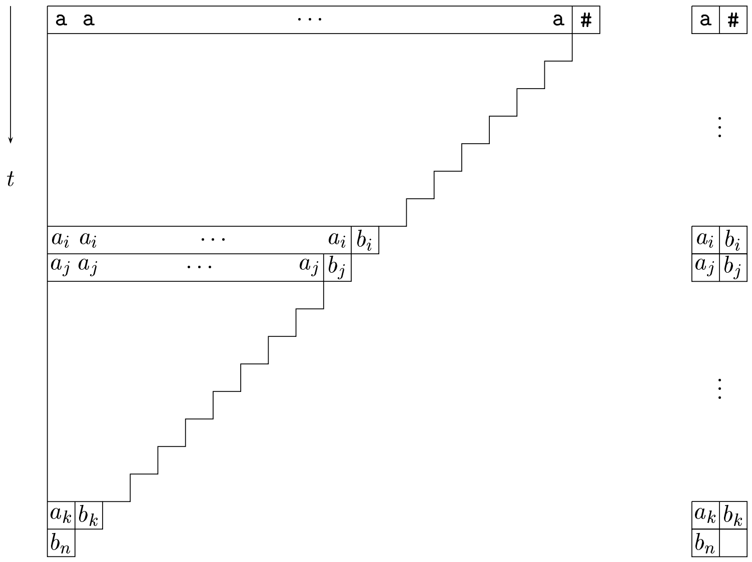

Figure 1: Example to the proof of lemma 9. Memory contents are omitted.

Since $\mathcal{M}'$ is a real-time acceptor the behavior of the leftmost cell cannot be influenced by the states of a cell i $(1 \leq i \leq n)$ at time t with $t > n - i + 1$. Furthermore a cell is not able to access the memory of its neighbor directly. Hence together with the first observation it suffices to know the state of the cells i at time $n - i + 1$ to simulate the state and memory transitions of the leftmost cell of $\mathcal{M}$ up to time n.

A corresponding SEQ $\mathcal{M}' = (S', M', D', s'_0, F', A')$ with $L = L(\mathcal{M}')$ uses two registers. In the first one it simulates the behavior of the cells $1, \ldots, i$ at time $n-i$ and in the second the behavior of the cell i at time $n-i+1$ for $1 \leq i \leq n$.

Now the construction of $\mathcal{M}'$ is straight forward:

$S' := S \times S, \quad M' := M, \quad A' := \{\mathtt{a}\}, \quad F' := \{(s,t) \mid t \in F\}, \quad s_0 := (\mathtt{a}, \#)$

$D' := \{(\delta'_i, \mu'_i, \alpha'_i) \mid 1 \leq i \leq k\}$ where $\forall\, i \in \{1, \ldots, k\}$:

$$\alpha'_i \;:=\; \alpha_i$$

$\forall\,(s,t) \in S', \forall\, m \in M \cup \{\varepsilon\}$:

$$\delta'_i((s,t), m, \mathtt{a}) \;:=\; (\delta_i(s,m,s), \delta_i(s,m,t))$$

$\forall\,(s,t) \in S', \forall\, u \in M^*$:

$$\mu'_i((s,t), u, \mathtt{a}) \;:=\; \mu_i(s,u,s)$$

□

The previous lemma together with theorem 8 yields our main result in this section:

Theorem 10 Let SEQ be a subfamily of SM. Then it holds

$$\mathscr{L}^u(SEQ) = \mathscr{L}^u_{rt}(\mathrm{OC}\,SEQ)$$

In [16] the so-called nondeterministic time-varying cellular automata (which actually are C NFA) have been investigated and related to NCA. An NCA is basically a C NFA where the mode of operation is significantly modified. Not only one nondeterministically determined local transition is applied to all the cells during each time step but for each cell a local transformation may individually nondeterministically be chosen. It has been left an open problem whether or not the inclusion $\mathscr{L}_{rt}(\mathrm{OC}\,NFA) \subseteq \mathscr{L}_{rt}(\mathrm{NOCA})$ is a proper one. Now we can answer it positively.

Corollary 11 $\qquad \mathscr{L}_{rt}(\mathrm{OC}\,NFA) \subset \mathscr{L}_{rt}(\mathrm{NOCA})$

Proof. By Theorem 10 we obtain $\mathscr{L}^u_{rt}(\mathrm{OC}\,NFA) = \mathscr{L}^u(NFA)$ is the family of regular tally languages. On the other side results in [3] imply $\mathscr{L}_{lt}(\mathrm{CA}) \subseteq \mathscr{L}_{rt}(\mathrm{NOCA})$ which in turn means that $\mathscr{L}_{rt}(\mathrm{NOCA})$ contains non-regular tally languages [4]. $\square$

An one-way stack automaton is a push-down automaton with an additional feature (cf. figure 2). In addition to push and pop at the top of the stack a stack head can enter the stack in read-only mode, traveling up and down the stack without rewriting any symbol. Hence an one-way stack automaton may intuitively be seen to be only slightly more complicated than a push-down automaton.

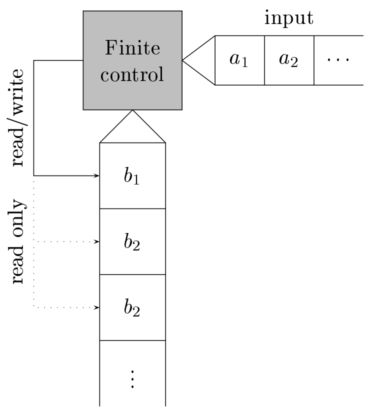

Figure 2: An one-way stack automaton.

In terms of sequential machines stack automata may be defined as follows. Let $V_0 := \{m_2, m_4, m_6, \ldots\}$. Moreover, let $P(u,v)$ iff $u = \varepsilon$ and $v \in \{\varepsilon, m_1 y\}$ or $u = m_1$ and $v = m_1 y$ or $u = m_1 ay$ and $v = m_1 xy$ or $u = xam_1 by$ and $v \in \{xm_1 aby, xam_1 by, xabm_1 y\}$ where $a, b, c \in V_0$ and $x, y \in V_0^*$. Let $Q(u,v)$ iff $u \in \{\varepsilon, xm_1\}$ and $v = \varepsilon$ or $u = m_1 ay$ and $v = a$ or $u = xam_1 m_i y$ and $v = m_{i+1}$ where $a, m_i \in V_0$ and $x, y \in V_0^*$. Then $OSA := SEQ(P,Q)$ resp. $DOSA := DSEQ(P,Q)$ denotes the family of one-way stack automata resp. deterministic one-way stack automata.

Lemma 12 $\mathscr{L}_{rt}^{u}(\mathrm{OCA}) \subset \mathscr{L}_{rt}^{u}(\mathrm{OC}\,DOSA)$

Proof. It is known that $\mathscr{L}^{u}(DOSA)$ contains non-regular languages [9, 10]. □

4 One-way arrays beyond real-time

The previous section showed that especially the capabilities of finite state machines coincides with the capabilities of real-time one-way cellular automata as far as tally languages are considered. Now we are going to prove that this relation remains valid even if we allow an amount of $o(\log)$ more computation time.

Theorem 13 Let $t : \mathbb{N} \to N$, $t \in o(log)$ be a mapping. Then it holds

$$\mathscr{L}_{rt+t}^{u}(\mathrm{OCA}) = \mathscr{L}_{rt}^{u}(\mathrm{OCA})\,.$$

Proof. Let L (say $\subseteq \{\mathtt{a}\}^{+}$) be a tally language such that L belongs to $\mathscr{L}_{rt+t}(\mathrm{OCA})$, and let $\mathcal{M} = (S, \delta, \#, F, A)$ be an OCA which is a witness for that fact. W.l.o.g. we may assume $|L|$ is infinite.

Let $N := \{n \in \mathbb{N} \mid \mathtt{a}^n \in L\}$ be the image of L under the length homomorphism. Further denote for all $n \in N$ by t_n the smallest nonnegative integer for which $(\Delta^{t_n+n}(w_{0,\mathtt{a}^n}))(1) \in F$ holds, i.e. the number of time steps more than real-time that the acceptance of $\mathtt{a}^n$ by $\mathcal{M}$ requires. $T := \{t_n \mid n \in N\}$ is the set of all such numbers.

We are going to prove that T is bounded by some $k \in \mathbb{N}_0$, which allows us to conclude that L belongs to $\mathscr{L}_{rt+k}(\mathrm{OCA})$. Since $\mathscr{L}_{rt+k}(\mathrm{OCA}) = \mathscr{L}_{rt}(\mathrm{OCA})$ [13] and all tally languages in $\mathscr{L}_{rt}(\mathrm{OCA})$ are regular [19] L must be regular, too.

It suffices to show the boundness of T. Contrarily, we assume that T is unbounded. Since $t \in o(\log)$ we can find (especially) for $b := (|S| + 1)^3$ a positive integer n_0 such that $t(n) < \lfloor \log_b(n) \rfloor$ for all $n \geq n_0$. The infinity of L and the unboundness of T imply the existence of an $n \in N$ with $n \geq n_0$ such that $t_n > t_m$ for all $m \in N$ with $n > m$ (cf. figure 3).

Now let $w^{(0)} := \mathtt{a}^n$ and define $w^{(i)} := \Delta(w^{(i-1)})$ for all $i \in \mathbb{N}$. The symbols of $w^{(i)}$ are denoted by $w_n^{(i)}, \ldots, w_1^{(i)}$ and numbered from right to left. We consider for $1 \leq i \leq n$ the words $u^{(i)} := w_i^{(i-1)} w_i^{(i)} \cdots w_i^{(i-1+\lfloor l \rfloor)}$ of length $\lfloor l \rfloor + 1$ where $l := \log^2_{|S|}(n)$ describing the evolution of the ith cell (from right) between the time steps $i - 1$ and $i - 1 + \lfloor l \rfloor$ inclusively.

Using the equality $l = \log_{|S|}(n)/2$ the number $|S|^{\lfloor l \rfloor + 1}$ of all words of length $\lfloor l \rfloor + 1$ over S can be approximated by $|S|^{\lfloor l \rfloor + 1} = |S|^{\lfloor l \rfloor} + |S| \leq \lfloor |S|^{l} \rfloor + |S| \leq \lfloor \sqrt{n} \rfloor + |S|$. By our choice of b we further have $1 \leq \lfloor \log_b(n) \rfloor$ and therefore $|S|^3 < n$ and thus $|S| < \lfloor \sqrt{n} \rfloor$ since $|S| \geq 2$. So we are done with $|S|^{\lfloor l \rfloor + 1} < \lfloor \sqrt{n} \rfloor^2 \leq n$.

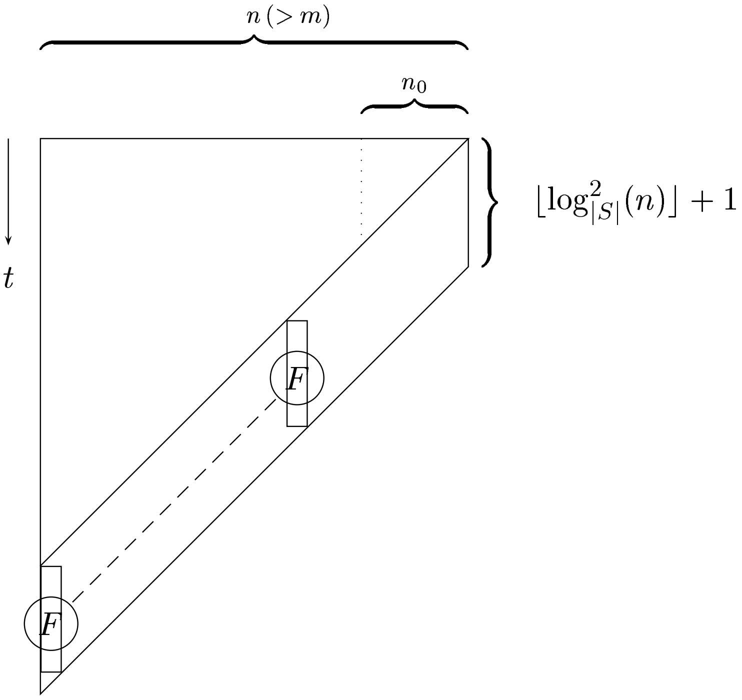

Figure 3: Example to the proof of theorem 13.

Now the pidgenhole principle ensures that at least two of the words $u_1, \ldots, u_n$ are equal, say u_i and u_j with $1 \leq i < j \leq n$. Since $\delta(w_k^{(k-1)}, w_k^{(k-1)}) = w_{k+1}^{(k)}$ for all $1 \leq k < n$, and because of the determinism of the local transformation we observe that u_{k+1} is uniquely determined by u_k. Therefore the equality of u_i and u_j implies especially the equality of u_{n+i-j} and u_n.

Since $\lfloor l \rfloor + 1 > \lfloor \log_b(n) \rfloor > t(n) \geq t_n$ at least one symbol of $u_n = u_{n+i-j}$ belongs to F. Due to the one-way information flow the evolution of a cell is independent of the behavior of the cells on the left from it. So we can conclude that $\mathtt{a}^{n+i-j} \in L$ and moreover $t_n = t_{n+i-j}$ which contradicts our choice of n. $\square$

The theorem has shown that at least for tally languages there is a gap between real-time and real-time plus logarithmic time in OCA. On the other hand, the following lemma shows that an amount of log more time strictly increases the family of accepted languages, which answers our first question.

Lemma 14 It holds

$$\{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\} \in \mathscr{L}^u_{rt+\log}(\mathrm{OCA})\,.$$

Proof. The following OCA $\mathcal{M} = (S, \delta, \#, \{\mathtt{a}\}, F)$ accepts $L := \{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\}$ with time complexity $rt + \log$:

$S := \{\mathtt{a}, \mathtt{e}, \mathtt{1}, \mathtt{+}, \mathtt{0}, \overset{\bullet}{\mathtt{0}}, \overset{\mathtt{+}}{\mathtt{1}}, \#\}$
$A := \{\mathtt{a}\}$
$F := \{\mathtt{+}\}$

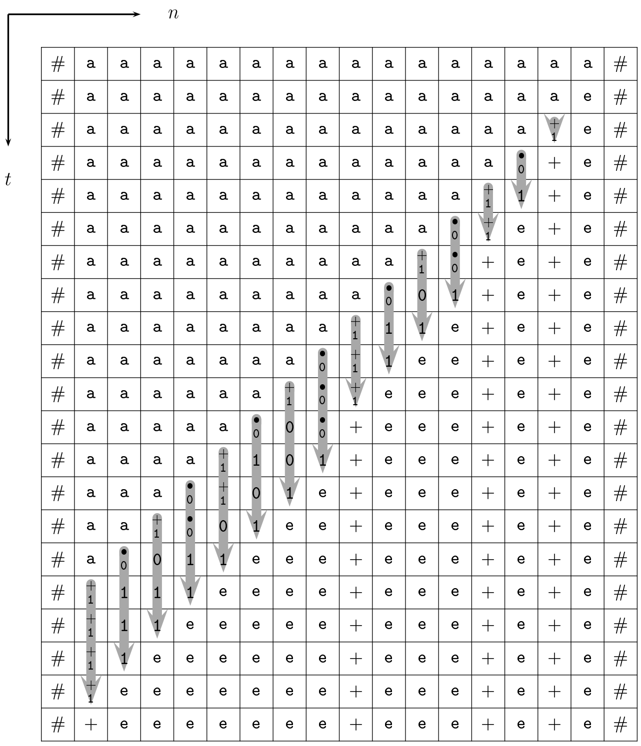

Figure 4: Example to the proof of lemma 14.

$\forall\, s,t \in S$:

$$\delta(s,t) := \begin{cases} \mathtt{e} & \text{if} \quad (s \notin \{\overset{\bullet}{\mathtt{0}}, \overset{+}{\mathtt{1}}, \mathtt{+}, \mathtt{a}\} \wedge t \in \{\mathtt{e}, \mathtt{+}\}) \vee (s = \mathtt{a} \wedge t = \mathtt{\#}) \\ \mathtt{+} & \text{if} \quad (s = \overset{+}{\mathtt{1}} \wedge t = \mathtt{e}) \\ \overset{+}{\mathtt{1}} & \text{if} \quad (s = \mathtt{a} \wedge t \in \{\mathtt{e}, \overset{\bullet}{\mathtt{0}}\}) \vee (s = \overset{+}{\mathtt{1}} \wedge t = \mathtt{1}) \\ \overset{\bullet}{\mathtt{0}} & \text{if} \quad (s = \mathtt{a} \wedge t = \overset{+}{\mathtt{1}}) \vee (s = \overset{\bullet}{\mathtt{0}} \wedge t \in \{\overset{+}{\mathtt{1}}, \mathtt{1}\}) \\ \mathtt{0} & \text{if} \quad (s \notin \{\mathtt{a}, \overset{\bullet}{\mathtt{0}}\} \wedge t \in \{\overset{\bullet}{\mathtt{0}}, \mathtt{0}\}) \\ \mathtt{1} & \text{if} \quad (s = \overset{\bullet}{\mathtt{0}} \wedge t \in \{\mathtt{0}, \mathtt{e}, \mathtt{+}\}) \vee (s \neq \overset{+}{\mathtt{1}} \wedge t = \mathtt{1}) \\ s & \text{otherwise} \end{cases}$$

On input $\mathtt{a}^n$, $n \in \mathbb{N}$, in the first time step a (vertical) binary counter (initialized by zero) is generated in the rightmost cell which moves leftwards with maximal speed. Subsequently, the counter is incremented by one in every time step. If a carry over takes place the counter is enlarged by one digit.

A cell switches into the accepting state iff the counter which moves digit by digit through that cell consists of ones only. Thus, the counter represents the number $2^k - 1$ for some $k \in \mathbb{N}$.

Hence, the counter reaches the leftmost cell at time step n. After that it additionally takes $\log(n)$ time steps to inspect the counter such that the time complexity is $rt + \log$.

Apart from the self-explanatory states $\overset{\bullet}{0}$ represents the digit zero with a carry over flag and $\overset{+}{1}$ the digit one with the information that all of the former digits has been ones, too. An accepting computation for $\mathtt{a}^{16}$ is shown in figure 4. $\square$

Corollary 15 Let $t : \mathbb{N} \to \mathbb{N}$, $t \in o(\log)$, be a mapping. Then it holds

$$\mathscr{L}^u(DFA) = \mathscr{L}^u_{rt}(\mathrm{OCA}) = \mathscr{L}^u_{rt+t}(\mathrm{OCA}) \subset \mathscr{L}^u_{rt+\log}(\mathrm{OCA}) \subseteq \mathscr{L}^u_{lt}(\mathrm{OCA})\,.$$

5 A wee bit nondeterminism

Now we are going to restrict the nondeterminism of an NCA in the following way. The set of local transformations is divided into two parts $D = \{\delta_d\} \cup D_{nd}$ giving a distinguished so-called deterministic local transformation δ_d and a possibly empty set of so-called nondeterministic local transformations D_{nd}. During the first time step a nondeterministically chosen transformation from D_{nd} is applied to at most k cells where $k \in \mathbb{N}_0$ is a fixed constant number. The other cells have to work according to δ_d. The cells performing a local transformation from D_{nd} are called nondeterministic and the others deterministic with respect to the actual computation. In subsequent time steps δ_d is applied to all the cells. We denote such a model by kC 1G-CA (k cells one guess CA).

The aim of the present section is to investigate kC 1G-CA and their one-way variants the kC 1G-OCA with respect to tally languages. Observe that in case of $k = 0$ these models are CA resp. OCA.

5.1 Time versus nondeterminism

The following theorem states that it is in some sense possible to gain time by allowing additional nondeterminism.

Theorem 16 Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping and $c \in \mathbb{N}$ and $k \in \mathbb{N}_0$ be constant numbers. Then it holds

$$\mathscr{L}^u_{c\cdot t(n)}(k\mathrm{C}\,1\mathrm{G\text{-}OCA}) \subseteq \mathscr{L}^u_{t(n)}((k+1)\mathrm{C}\,1\mathrm{G\text{-}OCA})\,.$$

Proof. Let $\mathcal{M}$ be a kC 1G-OCA which accepts a tally language L with time complexity $c \cdot t(n)$. Say L is a language over $\{\mathtt{a}\}$.

To achieve the desired speed-up an $(k+1)$C 1G-OCA acceptor $\mathcal{M}'$ of L has to simulate $\mathcal{M}$ c times faster. Therefore, in principle each cell of $\mathcal{M}'$ simulates c consecutive cells of $\mathcal{M}$. So at every time step a cell of $\mathcal{M}'$ can simulate c time steps of $\mathcal{M}$. Further, since L is a tally language each cell 'knows' the initial state of the cells it has to simulate from the very beginning (namely $\mathtt{a}$).

Actually $\mathcal{M}'$ works as follows on an input of length n. Its additional nondeterministic power is used to guess which might be cell $\lceil n/c \rceil$ by marking some of the cell(s). Additionally it forces the marked cell(s) to simulate only some $r \leq c$ cells of $\mathcal{M}$. Subsequently a signal which is generated during the first time step in the rightmost cell and which moves with maximal speed to the left is used to verify that exactly one such cell has been marked. Namely it prohibits each cell it enters to switch into an accepting state unless it has already passed through exactly one marked cell.

We may assume that exactly one cell i has been marked. It behaves as if it would manage the rightmost r cells of $\mathcal{M}$ in the simulation. Subsequently using two leftward moving signals it can be verified that $r = n - c(i-1)$ (and hence $c = \lceil n/c \rceil$), i.e., that $\mathcal{M}'$ simulates exactly n cells of $\mathcal{M}$. Namely one signal is generated during the first time step in the marked cell i which is delayed r time steps in cell i and c time steps in each cell left from it. The other signal is generated in the rightmost cell during the first time step and moves with maximal speed. Subsequently a cell is allowed to switch into an accepting state iff the two signals meet in that cell. So, the leftmost cell of $\mathcal{M}'$ is allowed to enter an accepting state only if $c(i-1) + r = n$ holds.

Finally, the remaining k nondeterministic transformations of $\mathcal{M}'$ can be brought in in such a way that at most k of the leftmost $\lceil n/c \rceil$ cells are able to simulate some cells of $\mathcal{M}$ which perform a nondeterminitic transformation during the first time step. Obviously, since k is finite within the first n time steps it can be ensured that at most k nondeterministic cells of $\mathcal{M}$ have been simulated. □

Corollary 17 Let $k \in \mathbb{N}_0$ be a constant number. Then it holds

$$\mathscr{L}_{lt}^{u}(k\mathrm{C}\,1\mathrm{G\text{-}OCA}) \subseteq \mathscr{L}_{rt}^{u}((k+1)\mathrm{C}\,1\mathrm{G\text{-}OCA})\,.$$

Theorem 16 can directly be transferred to kC 1G-CA. However for our purposes it suffices to formulate the corresponding corollary.

Corollary 18 Let $k \in \mathbb{N}_0$ be a constant number. Then it holds

$$\mathscr{L}_{lt}^{u}(k\mathrm{C}\,1\mathrm{G\text{-}CA}) \subseteq \mathscr{L}_{rt}^{u}((k+1)\mathrm{C}\,1\mathrm{G\text{-}CA})\,.$$

The next result shows that corollary 17 can actually be improved: not only inlusion but equality holds. Hence we have in this clearly defined situation a trade-off between time and nondeterminism.

Theorem 19 Let $k \in \mathbb{N}_0$ be a constant number. Then it holds

$$\mathscr{L}_{lt}^{u}(k\mathrm{C}\,1\mathrm{G\text{-}OCA}) = \mathscr{L}_{rt}^{u}((k+1)\mathrm{C}\,1\mathrm{G\text{-}OCA})\,.$$

Proof. It suffices to prove $\mathscr{L}_{rt}^{u}((k+1)\mathrm{C}\,1\mathrm{G\text{-}OCA}) \subseteq \mathscr{L}_{lt}^{u}(k\mathrm{C}\,1\mathrm{G\text{-}OCA})$. We transform a real-time $(k+1)$C 1G-OCA acceptor $\mathcal{M}$ for some tally language in five steps into a desired linear-time acceptor with the reduced amount of nondeterminism. Suppose $L := L(\mathcal{M})$ is a language over the singleton $\{\mathtt{a}\}$.

Step 1 At first $\mathcal{M}$ is modified such that its leftmost cell switches into an accepting state at time t if the already processed prefix of length $n - t$ of the input $\mathtt{a}^n$ belongs to L. More precisely, if the actual distribution and action of the nondeterministic cells within the leftmost $n-t$ cells would lead to acceptance of $\mathtt{a}^{n-t}$.

Therefore each cell of the resulting $(k+1)$C 1G-OCA $\mathcal{M}_1$ consists of two registers. In the first register the normal computation of $\mathcal{M}$ takes place. The second register of a cell i $(1 \leq i \leq n)$ holds at time t, $1 \leq t \leq i$, the corresponding state of cell i of $\mathcal{M}$ on input $\mathtt{a}^{t-n}$. Thus, during the first time step each cell additionally computes its state under the assumption that its neighbor is in the border state and stores the result in its second register. Subsequently the content of the second register of a cell results from applying the deterministic local transformation (of $\mathcal{M}$) to the state stored in its first register and the content of the second register of its neighbor.

Since $\mathcal{M}_1$ is an interim construction we better say that it accepts virtually if its leftmost cell contains an accepting state in its second register.

Step 2 Now $\mathcal{M}_1$ is modified such that its rightmost cell behaves as if all cells right from it would have been initialized by $\mathtt{a}$ and perform subsequently deterministic transformations.

Therefore the rightmost cell additionally to its own work simulates another (virtual) cell which is located to its right and is initialized by $\mathtt{a}$. The state transition of that virtual cell simply results from applying its actual state s to the local deterministic transformation $\delta_{d,1}$ of $\mathcal{M}_1$ under the assumption that its right neighbor would be in the same actual state: $\delta_{d,1}(s, s)$. Since the rightmost cell can identify itself this behavior can be achieved.

Now the leftmost cell of the newly obtained $(k+1)$C 1G-OCA $\mathcal{M}_2$ (which is no longer real-time bounded) may accept virtually at time t iff $\mathtt{a}^t \in L$ and, additionally, there exists an accepting computation of $\mathcal{M}$ for which the (at most) $(k+1)$ nondeterministic cells are contained in the leftmost n cells.

Step 3 Consider a computation of a $(k+1)$C 1G-OCA which requires $k+1$ nondeterministic cells and in which the rightmost cell is one of the nondeterministic ones. Such a computation can be simulated by a kC 1G-OCA as follows (cf. figure 5 (a) and (b)).

The k nondeterministic cells apart from the rightmost cell are located as in the considered computation. The rightmost cell which can identify itself deterministically simulates all possible transitions it might nondeterministically have been performed. Therefore it (and all the other cells) consists of $|D_{nd}| + 1$ registers where D_{nd} is the set of nondeterministic transitions of the $(k+1)$C 1G-OCA into which the corresponding states are stored. So, after the first time step all possible configurations depending on the rightmost cell can be simulated on different tracks in parallel. Finally, the computation leads to acceptance if at least on one of the tracks acceptance is simulated.

Now denote by $\mathcal{M}_3$ the corresponding kC 1G-OCA which is build from $\mathcal{M}_2$. Assume that $\mathcal{M}_2$ virtually accepts at time t whereby cell i is the rightmost nondeterministic cell (if any). Let $d = n - i$. Then cell $d+1$ of $\mathcal{M}_3$ would be able to accept virtually at time t if the nondeterministic cells are shifted by d to the right.

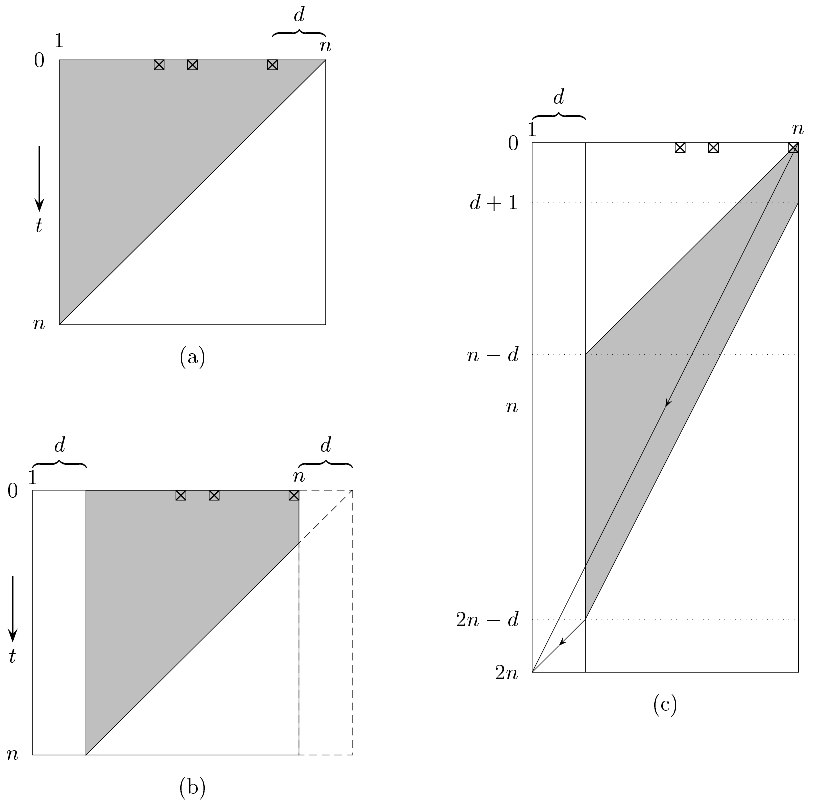

Figure 5: Overview of the construction in the proof of theorem 19. (a) Real-time $(k+1)$C 1G-OCA, (b) right shift of the computation in step 3, (c) delayed computation in step 5. Nondeterministic cells are marked by ⊠.

On the other side virtual acceptance in cell j of $\mathcal{M}_3$ at time t where the cells $1, \ldots, j-1$ are deterministic ones implies that the leftmost cell of $\mathcal{M}_2$ would virtually accept at time t if the nondeterministic cells are located correspondingly.

Step 4 Now the computation of $\mathcal{M}_3$ is delayed by one time step for each cell from right to left. I.e., a cell i of the resulting kC 1G-OCA $\mathcal{M}_4$ simulates at time $t+(n-i+1)$ the state of cell i at time t. Therefore especially the leftmost cell of $\mathcal{M}_4$ simulates at time $2n$ the state of $\mathcal{M}_3$ at time n.

The cells of $\mathcal{M}_4$ consist of two registers. During the first time step each cell simulates a transformation of $\mathcal{M}_3$ and stores the corresponding state in its first register. Subsequently the second register is used to store the previously simulated state. If a cell is allowed to proceed with the simulation only if the second register of its neighbor is not empty then the required delay can be reached. Observe that the rightmost cell which, again, can identify itself is excluded from that restriction.

Step 5 Finally $\mathcal{M}_4$ is modified in the following way what results in $\mathcal{M}_5$ (cf. figure 5 (c)): In the first time step a leftward moving signal s is generated in the rightmost cell which moves with speed $1/2$. Further, if a cell accepts virtually it generates a signal c which is sent to the left with maximal speed. If the signal s meets some signal c in some cell the corresponding cell enters a (real) accepting state.

Obviously, the leftmost cell is reached by s at time $2n$. Hence $\mathcal{M}_5$ is a linear-time kC 1G-OCA acceptor.

Now let $\mathtt{a}^n \in L$. Then there exists some cell i of $\mathcal{M}_4$ which virtually accepts at time $n+(n-i+1)$. Its signal c arrives in the leftmost cell at time $n+(n-i+1)+i-1=2n$. Hence $\mathtt{a}^n \in L(\mathcal{M}_5)$.

If on the other side $\mathtt{a}^n \in L(\mathcal{M}_5)$ then the leftmost cell of $L(\mathcal{M}_5)$ has been reached by a signal c which was generated in some cell i at time $2n-i+1 = n+(n-i+1)$. This holds also for cell i of $L(\mathcal{M}_4)$ and hence $\mathtt{a}^n \in L$, i.e., $L = L(\mathcal{M}_5)$. □

5.2 Information flow versus nondeterminism

In the sequel the relation between kC 1G-OCA and kC 1G-CA will be investigated whereby the amount of nondeterminism is taken into account.

The following result is already known for the deterministic case $k=0$. In [21] $\mathscr{L}_{rt}(\text{CA}) \subseteq \mathscr{L}^R_{2\cdot id}(\text{OCA})$ has been shown and in [4] the converse was proved. Using speed-up theorems [14] one obtains the corresponding result for linear-time. Observe, that for tally languages it clearly holds $\mathscr{L}^u_{rt}(\text{CA}) = \mathscr{L}^u_{lt}(\text{OCA})$.

Theorem 20 Let $k \in \mathbb{N}_0$ be a constant number. Then it holds

$$\mathscr{L}^u_{lt}(k\text{C 1G-OCA}) = \mathscr{L}^u_{rt}(k\text{C 1G-CA})\,.$$

Proof. The proof is analogous to the previously cited deterministic case.

However the adaption might require a cell of a kC 1G-OCA to perform a nondeterministic transformation during the second time step. Therefore observe that a local transformation can be represented as a finite array of $S^2 \times S$ states. So a cell may guess such an array (and hence a local transformation) during the first time step which is applied to the cell in the second one. □

Corollary 21 Let $k \in \mathbb{N}_0$ be a constant number. Then it holds

$$\mathscr{L}^u_{rt}(k\text{C 1G-CA}) = \mathscr{L}^u_{rt}((k+1)\text{C 1G-OCA})\,.$$

Proof. The assertion follows by theorem 19 and theorem 20. Namely, it holds $\mathscr{L}^u_{rt}(k\text{C 1G-CA}) = \mathscr{L}^u_{lt}(k\text{C 1G-OCA})$ and $\mathscr{L}^u_{lt}(k\text{C 1G-OCA}) = \mathscr{L}^u_{rt}((k+1)\text{C 1G-CA})$. □

From corollary 21 it follows $\mathscr{L}^u_{lt}(k\text{C 1G-CA}) \supseteq \mathscr{L}^u_{rt}((k+1)\text{C 1G-OCA})$ and $\mathscr{L}^u_{rt}(k\text{C 1G-CA}) \subseteq \mathscr{L}^u_{lt}((k+1)\text{C 1G-OCA})$ such that we obtain the relations depicted in figure 6.

Observe further that corollary 21 also covers the deterministic case $k=0$: $\mathscr{L}^u_{rt}(\text{CA}) = \mathscr{L}^u_{rt}(1\text{C 1G-OCA})$.

$$
\begin{array}{ccc}
\vdots & & \vdots \\
\rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
\mathscr{L}^u_{lt}(\text{2C 1G-CA}) & & \mathscr{L}^u_{rt}(\text{3C 1G-OCA}) \\
\rotatebox{90}{$\subseteq$} & & \parallel \\
\mathscr{L}^u_{rt}(\text{2C 1G-CA}) & = & \mathscr{L}^u_{lt}(\text{2C 1G-OCA}) \\
\rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
\mathscr{L}^u_{lt}(\text{1C 1G-CA}) & & \mathscr{L}^u_{rt}(\text{2C 1G-OCA}) \\
\rotatebox{90}{$\subseteq$} & & \parallel \\
\mathscr{L}^u_{rt}(\text{1C 1G-CA}) & = & \mathscr{L}^u_{lt}(\text{1C 1G-OCA}) \\
\rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
\mathscr{L}^u_{lt}(\text{CA}) & & \mathscr{L}^u_{rt}(\text{1C 1G-OCA}) \\
\rotatebox{90}{$\subseteq$} & & \parallel \\
\mathscr{L}^u_{rt}(\text{CA}) & = & \mathscr{L}^u_{lt}(\text{OCA}) \\
 & & \rotatebox{90}{$\subset$} \\
 & & \mathscr{L}^u_{rt}(\text{OCA}) \\
 & & \parallel \\
 & & \mathscr{L}^u(DFA)
\end{array}
$$

Figure 6: A hierarchy of tally families.

References

[1] Book, R. V. *Tally languages and complexity classes.* Information and Control 26 (1974), 186–193.

[2] Book, R. V. *Context-sensitive tally languages.* Bulletin of the European Association for Theoretical Computer Science 15 (1981), 31–34. Technical Contributions.

[3] Buchholz, Th., Klein, A., and Kutrib, M. *One guess one-way cellular arrays.* Mathematical Foundations in Computer Science 1998, LNCS 1450, 1998, pp. 807–815.

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

[5] Chrobak, M. *Finite automata and unary languages.* Theoretical Computer Science 47 (1986), 149–158.

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

[7] Dyer, C. R. *One-way bounded cellular automata.* Information and Control 44 (1980), 261–281.

[8] Geidmanis, D., Kaneps, J., Apsitis, K., and Taimina, D. *Tally languages accepted by alternating multitape finite automata*. Lecture Notes in Computer Science 1276 (1997), 422.

[9] Ginsburg, S., Greibach, S. A., and Harrison, M. A. *One-way stack automata*. Journal of the ACM 14 (1967), 389–418.

[10] Ginsburg, S., Greibach, S. A., and Harrison, M. A. *Stack automata and compiling*. Journal of the ACM 14 (1967), 172–201.

[11] Hemachandra, L. A. and Rubinstein, R. S. *Separating complexity classes with tally oracles*. Theoretical Computer Science 92 (1992), 309–318.

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

[13] Ibarra, O. H., Kim, S. M. and Moran, S. *Sequential machine characterizations of trellis and cellular automata and applications*. SIAM Journal on Computing 14 (1985), 426–447.

[14] Ibarra, O. H. and Palis, M. A. *Some results concerning linear iterative (systolic) arrays*. Journal of Parallel and Distributed Computing 2 (1985), 182–218.

[15] Kasami, T. and Fuji, M. *Some results on capabilities of one-dimensional iterative logical networks*. Electronics and Communications in Japan 51-C (1968), 167–176.

[16] Krithivasan, K. and Mahajan, M. *Nondeterministic, probabilistic and alternating computations on cellular array models*. Theoretical Computer Science 143 (1995), 23–49.

[17] Kutrib, M. *Pushdown cellular automata*. Theoretical Computer Science 215 (1999), 239–261.

[18] Mazoyer, J. and Terrier, V. *Signals in one dimensional cellular automata*. Research Report RR 94-50, Ecole Normale Supérieure de Lyon, Lyon, 1994.

[19] Seidel, S. R. *Language recognition and the synchronization of cellular automata*. Technical Report 79-02, Department of Computer Science, University of Iowa, Iowa City, 1979.

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

[21] Umeo, H., Morita, K., and Sugata, K. *Deterministic one-way simulation of two-way real-time cellular automata and its related problems*. Information Processing Letters 14 (1982), 158–161.