6

Cellular Automata – A Computational Point of View

Martin Kutrib

Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
kutrib@informatik.uni-giessen.de

Summary. The advantages of homogeneous arrays of interacting processing elements are simplicity and uniformity. It turned out that a large array of not very powerful elements operating in parallel can be programmed to be very powerful. One type of system is of particular interest: cellular automata whose homogeneously interconnected deterministic finite automata (the cells) work synchronously at discrete time steps obeying one common transition function. Cellular automata have extensively been investigated from different points of view. Here we discuss some of the main aspects from a computational point of view. The focus is on very simple types, that is, on one-dimensional cellular automata with nearest neighbor interconnections. In particular, we consider universality issues, the problem how to simulate data structures as stacks, queues, and rings without any loss of time, the famous Firing Squad Synchronization Problem, signals, and time constructible functions as well as several aspects of cellular automata as language acceptors. Some open problems are addressed.

6.1 Introduction

Cellular automata are an old branch of computer science. In the late forties of the last century they were proposed by John von Neumann in order to solve the logical problem of nontrivial self-reproduction. From this biological point of view he employed a mathematical device which is a multitude of interconnected automata operating in parallel to form a larger automaton, a macroautomaton built by microautomata. His famous early result reveals that it is logically possible for such a nontrivial computing device to replicate itself ad infinitum [72]. The name of these automata originates from the context in which they were developed. Due to their intuitive and colorful concepts, cellular automata have soon been considered from a computational point of view. So, from the very beginning, they were both, an interesting and challenging model for theoretical computer science and an interesting model for practical applications. Their inherent massive parallelism renders obvious applications as model for systems that are beyond direct measurements.

M. Kutrib: *Cellular Automata – A Computational Point of View*, Studies in Computational Intelligence (SCI) **113**, 183–227 (2008)
www.springerlink.com © Springer-Verlag Berlin Heidelberg 2008

Cellular automata are a young branch of computer science. Besides applications in industry, nowadays they open up new fields of application and modeling of natural phenomena in physics, biology, chemistry as well as in sociology, economics, and other human sciences. The development of practical and theoretical issues of cellular automata is impressive. In particular, it seems that currently the studies from a theoretical point of view follow two main branches. One focuses on the global behavior of cellular automata. Based on some topology the space of configurations is investigated. An important challenge with practical aspects is the characterization of cellular automata on the basis of their global transition function.

The other branch may be seen to deal with information. The flexibility of cellular automata to serve as programming tools can be utilized to develop tricky algorithms in order to solve classical problems as well as problems concerning the very nature of the system itself. An example for the latter case is the problem of synchronization, which gave rise to intensive research. In this connection, sources of questions are complexity issues as well as classifications in terms of formal language recognition. These questions are objects of the present article. More precisely, cellular automata are seen from a computational point of view. The main focus is on one-dimensional cellular automata which are linear arrays of cells that are connected to their nearest neighbors. The cells are exactly in one of a finite number of states, which is changed according to local rules depending on the current state of a cell itself and the current states of its neighbors. The state changes take place simultaneously at discrete time steps.

The presented topics are far from being complete. From the many interesting ones only a few could be chosen. In the following Section 6.2 basic definitions and preliminaries are given. Higher-dimensional systems with arbitrary cell interconnections are introduced as generalizations of one-dimensional systems with the mentioned nearest neighbor connections. For unbounded cellular spaces universality is obtained. After presenting an approach to evidence based on the possibility to model logical gates and information transition in two-dimensional cellular spaces with the simple rules of the Game of Life, it is shown how cellular spaces can simulate Turing machines. Besides, investigations concerning universality (often combined with other properties) are done, for example, in [1, 10, 33, 45, 46, 52, 53, 54, 30]. A survey of universality and decidability versus undecidability in cellular automata and several other models of discrete computations can be found in [44]. Next we turn to show how to simulate stacks, queues, and rings by one-dimensional cellular automata without any loss of time. The simulations may serve as tools for designing algorithms or as subroutines for programming cellular automata [6, 29].

The famous Firing Squad Synchronization Problem is dealt with in Section 6.3. It was raised by Myhill in 1957 and emerged in connection with the problem to start several parts of a parallel machine at the same time. The first published reference appeared with a solution found by McCarthy and Minsky in [50]. Roughly speaking, the problem is to set up a cellular space such that

all cells in a region change to a special state for the first time after the same number of steps.

Section 6.4 is devoted to the study of signals and constructibility of functions. Signals are used to solve problems. Examples are the basic signals that appear in solutions of the Firing Squad Synchronization Problem, or complex signals that allow to generate prime numbers. So, they can be seen as tools for algorithm design. In general, signals are able to transmit or encode information in cellular spaces. They have been used for a long time, but the systematic study originated from [49]. Basic questions are what kind of signals can be sent, or which speed is possible.

One of the main branches in the theory of cellular automata is considered in Section 6.5. Clearly, the data supplied to some device can be arranged as strings of symbols. Instances of problems to solve can be encoded as strings with a finite number of different symbols. Furthermore, complex answers to problems can be encoded as binary sequences such that the answer is computed bit by bit. In order to compute one piece of the answer, the set of possible inputs is split into two sets associated with the binary outcome. From this point of view, the computational capabilities of the devices are studied in terms of string acceptance, that is, the determination to which of the two sets a given string belongs. These investigations are done with respect to and with the methods of language theory. For cellular spaces and automata they originated from [11, 12] and [61, 31]. Over the years substantial progress has been achieved, but there are still some basic open problems with deep relations to other fields. A basic hierarchy of cellular language families is established, and the levels are compared with well-known families of the Chomsky hierarchy. The results are depicted in Figure 6.31. Closure properties are summarized in Table 6.1, and decidability problems are briefly discussed.

6.2 Basics and Preliminaries

We denote the set of integers by $\mathbb{Z}$ and the set of nonnegative integers by $\mathbb{N}$. The data supplied to the devices in question can be arranged as strings of symbols. In connection with formal languages, strings are called *words*. Let A^* denote the set of all words over a finite alphabet A. The *empty word* is denoted by λ, and we set $A^+ = A^* - \{\lambda\}$. For the *reversal of a word* w we write w^R, and for its *length* we write $|w|$. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*.

6.2.1 Cellular Spaces

Basically, the idea of cellular automata is to form a massively parallel device as a multitude of interacting simple processing elements. In order to keep the systems tractable, a high degree of homogeneity is preferable. Moreover, the processing elements have to be chosen as simple as possible. So, the elements – which sometimes are called cells – are represented by finite Moore automata.

Due to the need for homogeneity all cells are identical. In addition, they are arranged as grid where one dimension, that is, a linear array whose cells are identified by integers, is of particular interest in the sequel. Homogeneous local communication structures are achieved by a unique interconnection scheme that defines the cells which are interconnected to a given cell. Eventually, the cells operate synchronously at discrete time steps obeying a local transition function, which maps the current state of the cell itself and the current states of its connected cells (neighbors) to the next state.

So, a multitude of finite automata operating in parallel form a larger automaton such that global results are achieved by local interactions only.

To be more precise, we define a cellular space formally. The notion *space* is due to the fact that, potentially, we have an infinite number of cells, that is, we deal with the entire Euclidean space $\mathbb{Z}$. In order to obtain two-way information flow we assume that each cell is connected to its both nearest neighbors.

Definition 1. *A* (one-dimensional) two-way cellular space (*CS*) *is a system* $\langle S, \delta, q_0, A, F \rangle$*, where*

1. S *is the finite, nonempty set of* cell states*,*
2. $\delta : S^3 \to S$ *is the* local transition function*,*
3. $q_0 \in S$ *is the* quiescent state *such that* $\delta(q_0, q_0, q_0) = q_0$*,*
4. $A \subseteq S$ *is the set of* input symbols*, and*
5. $F \subseteq S$ *is the set of* final states*.*

Basically, we have an infinite space but are interested in finite supports only. That is, beginning a computation with a finite number of non-quiescent cells, by definition we obtain only finitely many non-quiescent cells at every time step. This determines the role played by the quiescent state. The set of final states has been included with an eye towards applications.

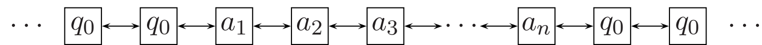

Fig. 6.1. A (one-dimensional) two-way cellular space.

In general, the global behavior of a cellular space is of interest. It is induced by the local behavior of all cells, that is, by the local transition function. More precisely, a *configuration* of a cellular space $\langle S, \delta, q_0, A, F \rangle$ at time $t \geq 0$ is a description of its global state, which is formally a mapping $c_t : \mathbb{Z} \to S$. The configuration at time 0 is defined by the given input $w = a_1 \cdots a_n \in A^+$, $n \geq 1$. We set $c_0(i) = a_i$, for $1 \leq i \leq n$, and $c_0(i) = q_0$ otherwise. Configurations may be represented as words over the set of cell states in their natural ordering, where the quiescent state is represented by the empty word. For example, the initial configuration for w is represented by $a_1 a_2 \cdots a_n$. Successor configurations are computed according to the global transition function Δ.

Let c_t, $t \geq 0$, be a configuration. Then its successor $c_{t+1} = \Delta(c_t)$ is defined by $c_{t+1}(i) = \delta(c_t(i-1), c_t(i), c_t(i+1))$, for all $i \in \mathbb{Z}$. A computation can be represented as *space-time diagram*, where each row is a configuration and the rows appear in chronological ordering.

An elementary technique in automata theory is the usage of multiple tracks. Basically, this means to consider the state set as Cartesian product of some smaller sets. Each component of a state is called *register*, and the same register of all cells together form a *track*.

In the sequel, for convenience and readability we may omit the definition of local transition functions for situations that do not change the state. Especially, we omit $\delta(q_0, q_0, q_0) = q_0$.

Example 1. The following cellular space $\mathcal{M} = \langle S, \delta, q_0, A, F \rangle$ reverses its input $w \in A^+$ in $|w|$ time steps (cf. Figure 6.2). It uses two tracks that are implemented by the state set $S = (A \cup \{\sqcup\})^2 \cup \{q_0\}$. Let (s_1, s_2), (s_3, s_4) and (s_5, s_6) be arbitrary states from $S \setminus \{q_0\}$.

$$\begin{aligned} \delta(q_0, (s_3, s_4), q_0) &= (s_4, s_3) \\ \delta(q_0, (s_3, s_4), (s_5, s_6)) &= (s_5, s_3) \\ \delta((s_1, s_2), (s_3, s_4), q_0) &= (s_4, s_2) \\ \delta((s_1, s_2), (s_3, s_4), (s_5, s_6)) &= (s_5, s_2) \end{aligned}$$

□

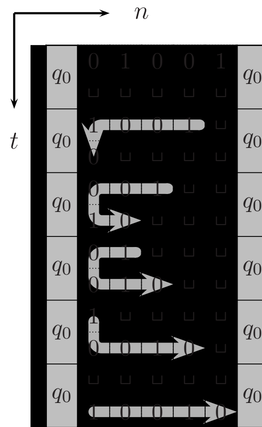

Fig. 6.2. Space-time diagram of a two-way cellular space reversing its input.

6.2.2 Important Generalizations

So far, cellular spaces have been introduced as one-dimensional arrays whose cells are connected to their immediate neighbors. Certainly, these types belong

to the most important and natural ones. In particular, from a computational perspective they are best investigated. However, there are many generalizations which are just as interesting and natural. More generally speaking, the specification of a cellular space includes the type and specification of the cells, their interconnection scheme (which can imply a dimension of the system), the local rules which are formalized as local transition function, and the input and output modes. In the present subsection we briefly deal with two generalizations. First, we consider arbitrary unique interconnection schemes which are called *neighborhood-indices* and, secondly, devices whose cells are arranged as *d-dimensional grids*.

So, assume that the cells of a cellular space are arranged as d-dimensional grid such that we deal with the Euclidean space $\mathbb{Z}^d$.

Definition 2. *1. Let $d, k \geq 1$ be positive integers. A d-*dimensional neighborhood-index of degree *k is a k-tuple $N = (n_1, n_2, \ldots, n_k)$ of different elements from $\mathbb{Z}^d$.*
2. Some cell $j \in \mathbb{Z}^d$ is called a neighbor *of cell $i \in \mathbb{Z}^d$, if there is a $k' \in N$ such that $j = i + k'$. The cells i and j are called* neighbors*, if either i is neighbor of j, or vice versa.*

In order to identify the neighbors of a cell i one has to add the elements of N to i. In particular, if 0 belongs to N, then cell i is its own neighbor. Only in this case the next state of a cell depends on its current state. Configurations are now mappings $c_t : \mathbb{Z}^d \to S$, and the global transition function Δ is induced by the local transition function $\delta : S^k \to S$ as follows:

$$c_{t+1} = \Delta(c_t) \iff c_{t+1}(i) = \delta(c_t(i + n_1), \ldots, c_t(i + n_k)), \text{ for all } i \in \mathbb{Z}^d.$$

There are general methods that allow to simulate a cellular space by another one having a (reduced) standard neighborhood-index. So, it suffices to consider the most important standard ones. Whenever the ordering of the elements of a neighborhood-index does not matter, we may specify it as a set.

Example 2. Let $d \geq 1$, $k \geq 0$, and $m_1, \ldots, m_d$ denote the components of $m \in \mathbb{Z}^d$. Then

$$H_k^d = \{m \in \mathbb{Z}^d \mid k \geq \textstyle\sum_{i=1}^d |m_i|\} \quad \text{or}$$
$$\bar{H}_k^d = \{m \in \mathbb{Z}^d \mid k \geq \textstyle\sum_{i=1}^d |m_i| \wedge m_i \geq 0, \text{ for } 1 \leq i \leq d\}$$

are (generalized) *von-Neumann* neighborhoods. Similarly,

$$M_k^d = \{m \in \mathbb{Z}^d \mid k \geq \max\{|m_i| \mid 1 \leq i \leq d\}\} \quad \text{or}$$
$$\bar{M}_k^d = \{m \in \mathbb{Z}^d \mid k \geq \max\{|m_i| \mid 1 \leq i \leq d\} \wedge m_i \geq 0, \text{ for } 1 \leq i \leq d\}$$

are (generalized) *Moore* neighborhoods (cf. Figure 6.3). □

The following famous cellular space is known as *Game of Life*. While the underlying rules are quite simple, the global behavior is rather complex. In fact, it is unpredictable.

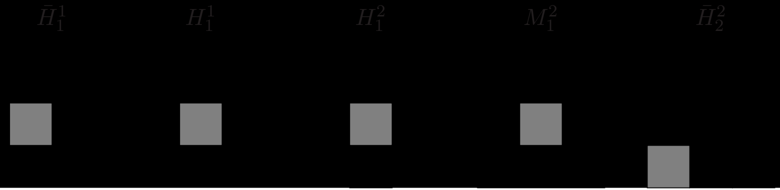

Fig. 6.3. Standard neighborhoods (the origin is shaded).

Example 3. We consider the two-dimensional space $\mathbb{Z}^2$. The cells are connected according to the Moore-neighborhood M_1^2, where each cell is connected to itself and to its eight immediate neighbors. Cells may be dead or alive, so the state set is chosen to be $\{0, 1\}$. The local transition function is defined dependent on the number of living cells in the neighborhood. In particular, a cell stays or becomes alive, if there are exactly three living cells within its Moore-neighborhood. It stays in its current state, if there are exactly four living cells within its Moore-neighborhood, and it dies from overpopulation or isolation otherwise.

The Game of Life made its first appearance in [16]. Over the years very interesting properties have been discovered. Some of them are based on the behavior of patterns that represent the arrangement of dead and living cells (cf. Figures 6.4 and 6.5). □

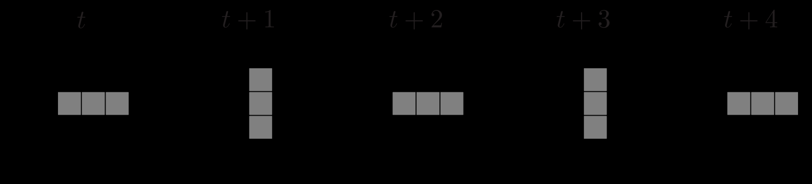

Fig. 6.4. Evolution of a periodical stationary pattern (blinker) in the Game of Life. Living cells are shaded.

6.2.3 Universality

In order to explore the power of general cellular spaces, we are now going to prove their universality. To this end, it is shown how cellular spaces can simulate Turing machines. Moreover, given a Turing machine, the corresponding cellular space should be as simple as possible. Therefore, we present a direct simulation by a one-dimensional space, where the number of states depends on the number of states and tape symbols of the Turing machine [30].

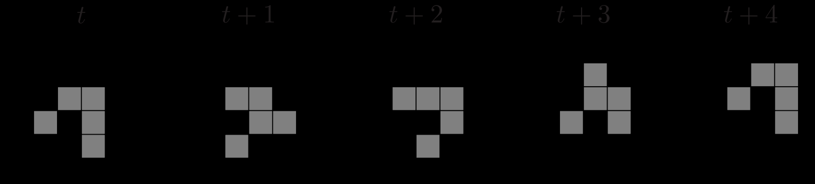

Fig. 6.5. Evolution of a periodical non-stationary pattern (glider) in the Game of Life. Living cells are shaded. Within four time steps the glider moves diagonally one cell to the north east.

But first we have another approach to evidence based on the generalizations. Roughly speaking, the idea is to model logical gates and information transmission in two-dimensional cellular spaces. Then universal computers can be build and embedded into the space. Interestingly, the constructions can be done with the simple rules of the Game of Life. So, two states are sufficient [4].

A stream of information is modeled as stream of bits. Consider a stream of gliders moving with the same space between, and assume some of the gliders are missing. Then the stream can be interpreted as stream of bits where the presence of a glider means 1 and the absence means 0. The following pattern depicted in Figure 6.6 is known as *glider gun*. The core of the gun behaves

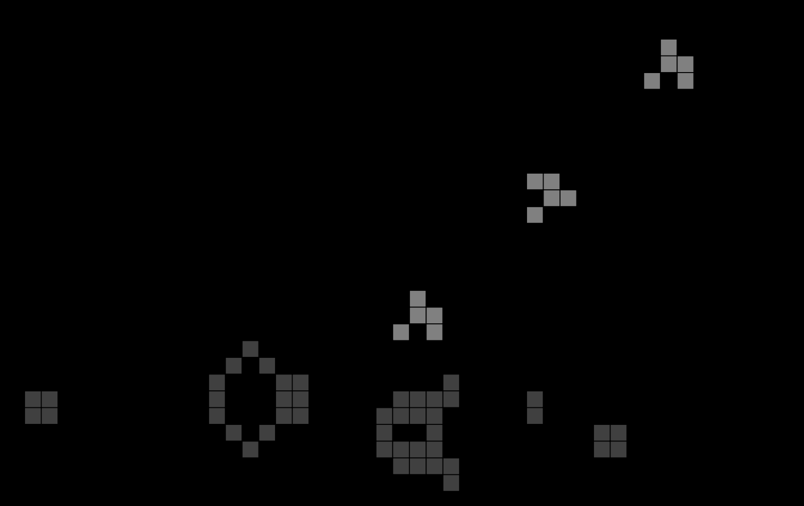

Fig. 6.6. Evolution of a glider gun in the Game of Life. Living cells are shaded. Within 30 time steps a glider is emitted to the north east. The arrows indicate the direction of the stream of gliders.

periodical. In addition, it emits a glider every 30 time steps. So, a glider gun can be seen as a source of a stream of bits consisting of ones only.

Now we turn to logical gates. In order to obtain a NOT gate, one observes that whenever two gliders collide at a right angle, then all wreckage disappears. So, the input stream to negate can be directed to a bit stream emitted by a glider gun. If a 1 (a glider) of the input stream reaches the collision area, it will collide with the incoming glider from the gun and is destroyed. If a 0 of the input stream reaches the collision area, the incoming glider will pass the collision area. In this way a 1 yields to a 0, and vice versa (cf. Figure 6.7).

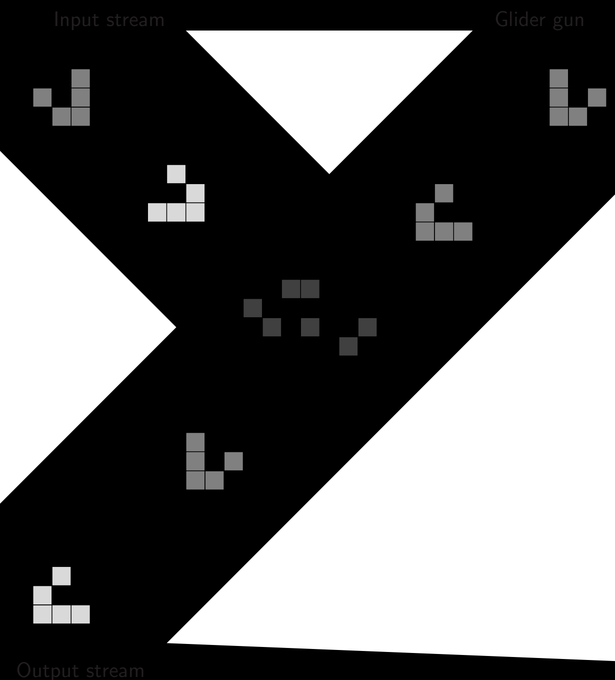

Fig. 6.7. A NOT gate in the Game of Life. Living cells are shaded. The cells shaded lightgray are not alive. They indicate the missing glider representing a 0.

Similarly, AND and OR gates are constructed. Figure 6.8 shows the schematic diagrams, where G means glider gun, and E is a pattern called *eater*. An eater absorbs incoming gliders.

The universality of cellular spaces follows since universal computers can be build from logical gates and bit streams. These computers can be embedded into the space. But it is worth mentioning that the effective construction requires to start with finite configurations of the cellular space. On the other hand, the computers may use potentially infinite memory. Nevertheless, by

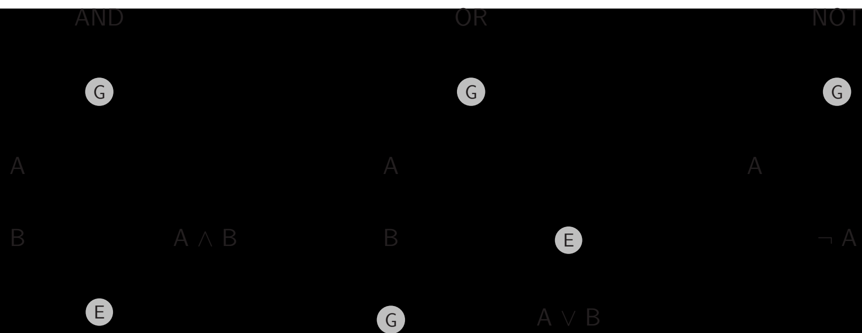

Fig. 6.8. Schematic diagrams of logical gates in the Game of Life. Glider guns and eaters are denoted by G and E.

nontrivial constructions it is possible to extend the available memory on demand of the computation [4].

Next we show how to simulate an arbitrary Turing machine by a one-dimensional cellular space with von-Neumann H_1^1 neighborhood, where the number of states depends on the number of states and tape symbols of the Turing machine. Since the Turing machine is arbitrary, in particular, the simulation of universal Turing machines is possible. There are universal Turing machines, for example, with four states and six tape symbols [56]. So, the next theorem gives also an upper bound on the size necessary for a (universal) cellular space.

Theorem 1. *Let $\mathcal{T} = \langle S, T, \delta, s_0, \sqcup \rangle$ be a one-tape Turing machine with state set S, tape symbols T, transition function δ, initial state s_0, and blank symbol $\sqcup$. Then there is a cellular space $\mathcal{M}$ with $|T|+4|S|$ states, that simulates $\mathcal{T}$ in twice the time.*

Proof. Without loss of generality, we assume that S and T are disjoint. Each symbol of the tape inscription is stored in one cell of $\mathcal{M}$. The left neighbor of the cell storing the currently scanned tape symbol represents the current state of $\mathcal{T}$ (cf. Figure 6.9).

At first glance, due to the H_1 neighborhood of the cellular space, the problem arises that a possible left move of the head cannot be observed by the cell at the left of the cell representing the state of $\mathcal{T}$. But an intermediate step can solve the problem. In particular, the cell representing the state of $\mathcal{T}$ changes to some new state that indicates the next state as well as the intended head movement. For simplicity, we do the same for right moves and no moves. The formal construction of $\mathcal{M} = \langle S', \delta', q_0, A, F \rangle$ is as follows:

$$S' = S \cup T \cup (S \times \{\mathit{stay}, \mathit{right}, \mathit{left}\})$$

The local transition function δ' is defined dependent on δ. Let $s, s' \in S$ and $a, a' \in T$. For all $a_1, a_2 \in T$,

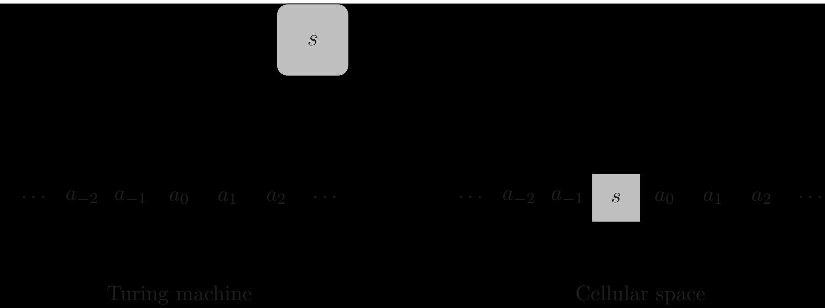

Fig. 6.9. Correspondent configurations of a Turing machine and a simulating cellular space.

$$
\begin{aligned}
\delta(s,a) = (s',a',\mathit{stay}) \Longrightarrow (\ & \delta'(a_1,s,a) = (s',\mathit{stay}),\\
& \delta'(s,a,a_1) = a',\\
& \delta'(a_1,(s',\mathit{stay}),a') = s'),\\
\delta(s,a) = (s',a',\mathit{right}) \Longrightarrow (\ & \delta'(a_1,s,a) = (s',\mathit{right}),\\
& \delta'(s,a,a_1) = a',\\
& \delta'(a_1,(s',\mathit{right}),a') = a',\\
& \delta'((s',\mathit{right}),a',a_1) = s'),\\
\delta(s,a) = (s',a',\mathit{left}) \Longrightarrow (\ & \delta'(a_1,s,a) = (s',\mathit{left}),\\
& \delta'(s,a,a_1) = a',\\
& \delta'(a_1,(s',\mathit{left}),a') = a_1,\\
& \delta'(a_2,a_1,(s',\mathit{left})) = s').
\end{aligned}
$$

In all other situations cells do not change their states. □

6.2.4 Simulation of Data Structures

This subsection is devoted to show how to simulate certain data structures by (one-dimensional) cellular spaces without any loss of time. The simulations may serve as tools for designing algorithms or as subroutines for programming cellular spaces. First we consider pushdown stores (stacks) [6, 29], that is, stores obeying the principle *last in first out*. Assume without loss of generality that at most one symbol is pushed onto or popped from the stack at each time step. We distinguish one cell that simulates the top of the pushdown store. It suffices to use three additional tracks for the simulation. Let the three pushdown registers of each cell be numbered one, two, and three from top to bottom, and suppose that the third register is connected to the first register of the right neighbor. The content of the pushdown store is identified by scanning the registers in their natural ordering beginning in the distinguished cell, whereby empty registers are ignored (cf. Figure 6.10).

The pushdown store dynamics of the transition function is defined such that each cell prefers to have only the first two registers filled. The third

Fig. 6.10. Pushdown registers exemplarily storing the string *PUSHDOWNER*.

register is used as a buffer. In order to reach that charge it obeys the following rules (cf. Figure 6.11).

1. If all three registers of its left (upper) neighbor are filled, it takes over the symbol from the third register of the neighbor and stores it in its first register. The old contents of the first and second registers are shifted to the second and third register.
2. If the second register of its left neighbor is free, it erases its own first register. Observe that the erased symbol is taken over by the left neighbor. In addition, the cell stores the content of its second register into its first one, if the second one is filled. Otherwise, it takes the symbol of the first register of its right neighbor, if this register is filled.
3. Possibly more than one of these actions are superimposed.

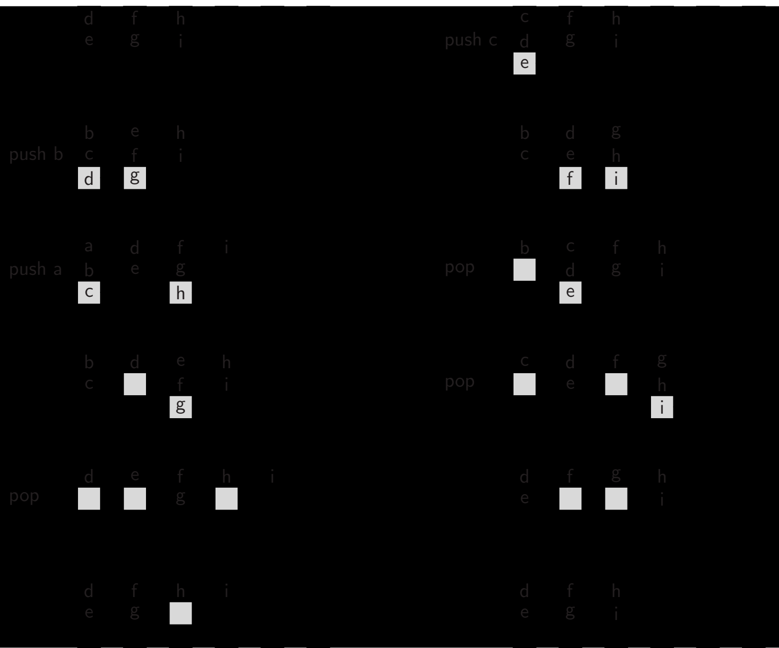

Fig. 6.11. Principle of a pushdown store simulation. Subfigures are in row-major order.

The main difference between pushdown stores and rings or queues is the way how to access the data. A *ring* obeys the principle *first in first out*, that is, the first symbol of the stored string is read and possibly erased while, in addition, a new symbol may be added at the end of the string. So, a ring can write and erase at the same time. A *queue* is a special case of a ring. It can either write or erase a symbol, but not both at the same time. In order to simulate a ring or queue, also no more than three additional registers are needed. The first two registers are used to store the symbols, where the second one is needed to cope with the situation when symbols are erased consecutively. The third track is used to move the new symbols from the front to the back of the string (cf. Figure 6.12).

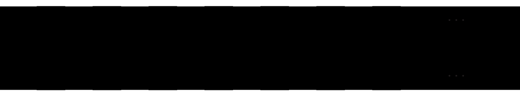

Fig. 6.12. Logical connections between ring registers.

Again, without loss of generality, we may assume that at most one symbol is entered to or erased from the ring at every time step. Moreover, each cell prefers to have the first two registers filled. Altogether, it obeys the following rules (cf. Figure 6.13).

1. If the third register of its left neighbor is filled, it takes over the symbol from that register. The cell stores the symbol into its first free register, if possible. Otherwise, it stores the symbol into its own third register.
2. If the third register of its left neighbor is free, it marks its own third register as free.
3. If the second register of its left neighbor is free, it erases its own first register. Observe that the erased symbol is taken over by the left neighbor. In addition, the cell stores the content of its second register into its first one, if the second one is filled. Otherwise it takes the symbol of the first register of its right neighbor, if this register is filled.
4. If the second register of its left neighbor is filled and its own second register is free, then the cell takes the symbol from the first register of its right neighbor and stores it into its own second register.
5. Possibly, more than one of these actions are superimposed.

6.3 Synchronization

The famous *Firing Squad Synchronization Problem* (FSSP) was raised by Myhill in 1957. It emerged in connection with the problem to start several parts of a parallel machine at the same time. The first published reference

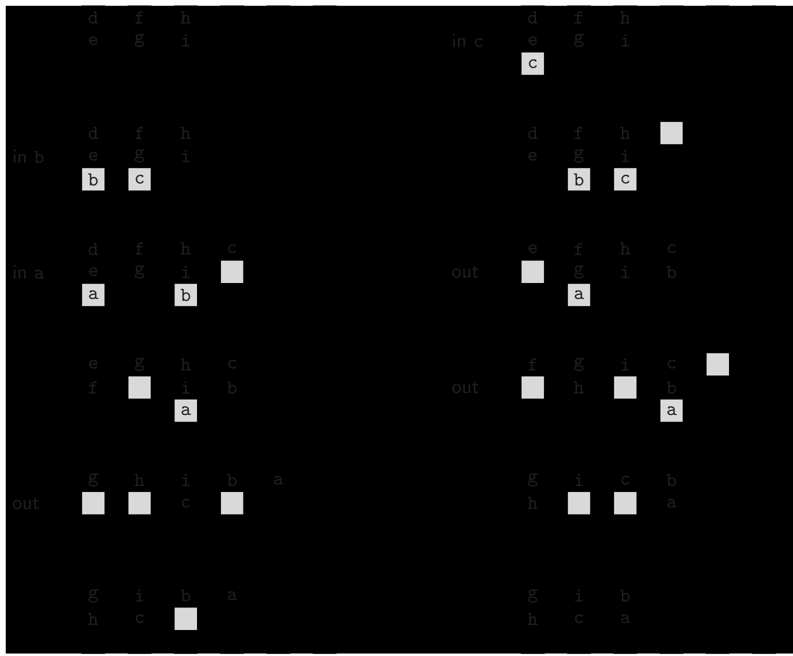

Fig. 6.13. Principle of a ring (queue) simulation. Subfigures are in row-major order.

appeared with a solution found by McCarthy and Minsky in [50]. Roughly speaking, the problem is to set up a cellular space such that all cells in a region change to a special state for the first time after the same number of steps. Originally, the problem has been stated as follows: Consider a finite but arbitrary long chain of finite automata that are all identical except for the automata at the ends. The automata are called soldiers, and the automaton at the left end is the general. The automata work synchronously, and the state of each automaton at time step $t+1$ depends on its own state and on the states of its both immediate neighbors at time step t. The problem is to find states and state transitions such that the general may initiate a synchronization in such a way that all soldiers enter a distinguished state, the firing state, for the first time at the same time step. At the beginning all non-general soldiers are in the quiescent state. More formally, the FSSP is defined as follows.

Definition 3. *Let C be the set of all cellular space configurations of the form $\#gq_0 \cdots q_0\#$, that is, for some $n \geq 1$, $c(0) = c(n+1) = \#$, $c(1) = g$ and $c(i) = q_0$, for $i \notin \{0, 1, n+1\}$. The* Firing Squad Synchronization Problem *is to specify a cellular space $\langle S, \delta, q_0, A, F\rangle$ such that for all $c \in C$,*

1. *there is a $t \geq 1$ such that $\big(\Delta^t(c)\big)(i) = f$, for $1 \leq i \leq n$ and some $f \in S$,*
2. *for all $0 \leq t' < t$ it holds $\big(\Delta^{t'}(c)\big)(i) \neq f$, for $1 \leq i \leq n$, and*
3. *$\delta(q_0, q_0, \#) = \delta(\#, q_0, q_0) = \delta(q_0, q_0, q_0) = q_0$.*

While the first solution of the problem takes $3n$ time steps to synchronize the n cells in between the cells in state #, Goto [18] was the first who presented a minimal time solution.

Lemma 1. *The minimal solution time for the FSSP is* $2n-2$*, where* n *is the number of cells to be synchronized.*

Proof. In contrast to the assertion assume there is a faster solution taking some time $t_f < 2n - 2$. Observe that the cells which are initially in the quiescent state may leave the quiescent state not before their left neighbor is in a non-quiescent state. Therefore, the rightmost cell n cannot leave the quiescent state before time $n - 1$. It takes another $n - 1$ time steps to send a feedback of this activation back to the general. Since $t_f < 2n - 2$, the general fires independently of such a feedback.

Now consider the problem with $2n - 1$ cells. Since the cells are deterministic, the general fires again at time $t_f < 2n - 2$. But at this time step the rightmost cell $2n-1$ is still in the quiescent state, since it takes at least $2n-2$ time steps to activate it. □

Next we present an algorithm that is not time optimal. It takes $3n$ time, but reveals basic procedural methods.

Algorithm 1. The FSSP can be solved by dividing the array in two, four, eight etc. parts of (almost) the same length until all cells are cut-points. Exactly at this time the cells change to the firing state synchronously. The divisions are performed recursively. First the array is divided into two parts. Then the process is applied to both parts in parallel, etc.

In order to divide the array into two parts, the general sends two signals $S1$ and $S2$ to the right (cf. Figure 6.14). Signal $S1$ moves with speed 1, that is, one cell per time step, and signal $S2$ with speed $1/3$, that is, one cell every three time steps. When signal $S1$ reaches the right end, a signal $S3$ is sent back to the left with speed 1. Signals $S2$ and $S3$ meet in the center of the array. Dependent on whether the length of the array is even or odd the center is represented by two or one cell. Next, the center cell(s) becomes a general. It sends signals $S1$ and $S2$ to the left and to the right. This process repeats until all cells are generals. At this time they change to the firing state synchronously.

Since the times needed to divide the sub-arrays are bounded by $3n/2$, $3n/4$, $3n/8$, and so on, altogether the algorithm takes at most $3n$ time steps. □

The next step towards a time optimal solution is to set up additional signals in order to determine the cut-points earlier.

Algorithm 2. The previous algorithm is modified as follows (cf. Figure 6.15). When signal $S1$ arrives at the right end, the end cell becomes a general and sends two signals $S3$ and $S4$ to the left. Signal $S4$ behaves as signal $S2$ except for the moving direction, that is, it moves with speed $1/3$ to the left. The center

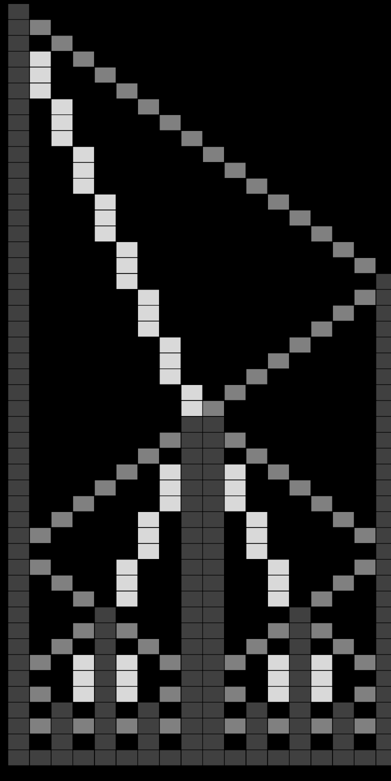

Fig. 6.14. Firing Squad Synchronization with a slow algorithm. Darkgray cells are generals, gray cells contain a signal with speed 1, lightgray cells a signal with speed 1/3, and crosshatched cells are in the firing state.

of the array is again determined by the collision of $S2$ and $S3$. The center cell(s) behaves as for the previous algorithm. In particular, it sends signals $S1$ and $S3$ to the right. The collision of $S1$ and $S4$ determines the center of the right half of the array after $3n/2 + n/4$ time steps. After another $n/8$ time steps the center of the third quarter of the array is known. If the remaining cut-points could be determined similarly, the total synchronization time would not exceed $2n$ time steps: $3n/2 + n/4 + n/8 + n/16 + \cdots = 2n$. Since without general there are only $n - 1$ cells to be passed through, the synchronization obeys the optimal time bound $2(n - 1)$.

Unfortunately, the presented procedure is not a solution, since only one of two cut-points is found, respectively. Clearly, one can determine the center of the left half of the array, if the general sends an additional signal $S5$ with speed 1/7 at initial time to the right. But then the next problem is to find the center of the left quarter of the array. To this end, the general can send another signal with speed 1/15 to the right. Altogether, for a solution the

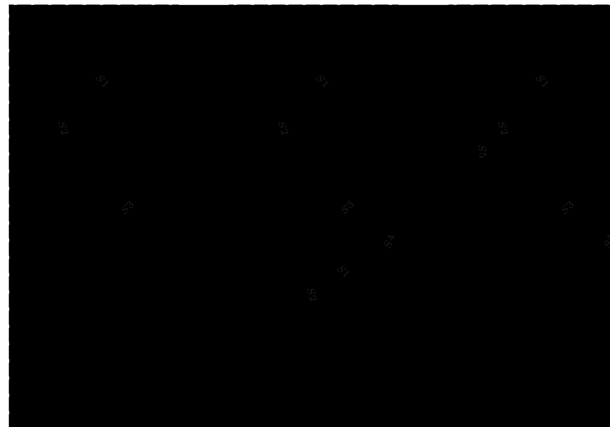

Fig. 6.15. Schematic diagrams of signals. Slow FSSP algorithm (left), additional signals for right cut-points (center), and additional signals for left cut-points (right).

general has to send signals with speeds $1/(2^k - 1)$, $k \geq 1$. Thus, the number of signals depends on the length of the array, and the problem is not solved. □

Nevertheless, there is a solution based on this approach [73]. The idea is rather simple, the additional signals are generated and moved by trigger signals (cf. Figure 6.16). The trigger signals themselves are emitted by signals $S1$ and $S3$ in the opposite direction at each other time step. Whenever a trigger signal reaches the leftmost or rightmost cell, a new signal to be triggered is generated. Whenever a trigger signal reaches a triggered signal, the latter is moved one cell ahead. On the other hand, any triggered signal absorbs each other trigger signal. That way, the desired behavior is achieved, and a minimal time solution for the FSSP is obtained.

Apart from time optimality there is a natural interest in efficient solutions with respect to the number of states or the number of bits to be communicated to neighbors. While there exists a time optimal solution where just one bit of information is communicated [47], the minimal number of states is still an open problem. The first time optimal solution [18] uses several thousand states. The presented algorithm from [73] takes 16 states. About one year later, an eight state time optimal solution was published [3]. Currently, a six state solution is known [48]. In the same paper it is proved that there does not exist a time optimal four state algorithm. It is a challenging open problem to prove or disprove that there exists a five state solution.

Many modifications and generalizations of the FSSP have been investigated. Just to mention a few of them, solutions for higher dimensions can be found in [19, 57, 60, 63, 70], fault tolerant synchronizations are studied in [41, 67], generalized positions of the general are considered in [51], and

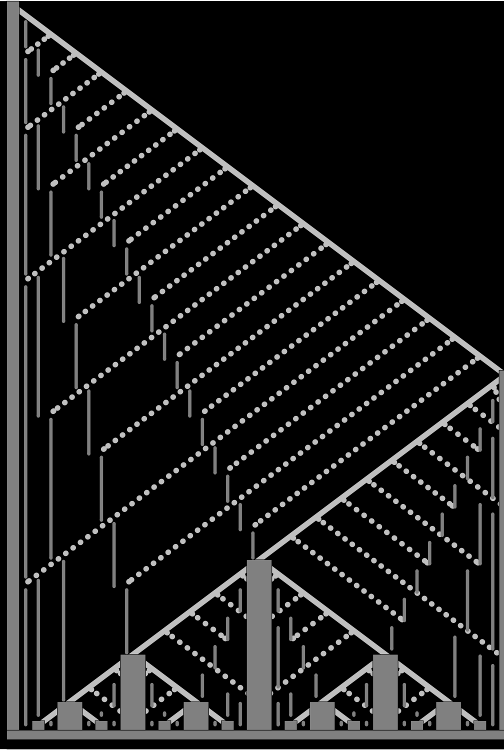

Fig. 6.16. Firing Squad Synchronization with a time optimal algorithm using trigger signals.

growing squads in [21]. In [32] the problem is solved for reversible cellular spaces, and in [27, 35, 38, 39] more general graphs are considered.

6.4 Signals and Time Constructibility

Signals are used to solve problems. Examples are the basic signals that appear in solutions of the FSSP, or complex signals that allow to generate prime numbers. So, they can be seen as tools for algorithm design. In general, signals are used to transmit or encode information in cellular spaces. They have been used for a long time, but the systematic study originated from [49]. Basic questions are what kind of signals can be send, or what speed is possible.

6.4.1 Signals

Roughly speaking, signals are described as follows: If some cell changes to the state s of its neighbor after some $k \geq 1$ time steps, and if subsequently its

neighbors and their neighbors do the same, then the basic signal s moves with speed $1/k$ in the corresponding direction.

By this description it becomes intuitively clear what signals are. But the concept is much more complex. So a formal treatment is advisable. Obviously, the maximal speed is one, that is, one cell per time step. Signals are formalized as mappings, where the signal is distinguished from its implementation, since not every mapping of the appropriate type can be implemented. The mapping takes a time step and yields the cell in which the signal resides at this time.

Definition 4. *A* signal *is a mapping* $\xi : \mathbb{N} \to \mathbb{Z}$, *where for all* $t \geq 0$, $\xi(t+1) \in \{\xi(t) - 1, \xi(t), \xi(t) + 1\}$.

The current site of an implemented signal is indicated by special states.

Definition 5. *A signal* ξ *is* CS-practicable, *if there is a cellular space* $\langle S, \delta, q_0, A, F\rangle$ *with distinguished state* $s \in S$, *subset* $S' \subseteq S$, *and initial configuration* $c_0(0) = s$, $c_0(i) = q_0$, *for* $i \neq 0$, *such that* $c_t(i) \in S' \iff \xi(t) = i$.

It is evident that there are simple and complex signals. In general, auxiliary signals are needed in order to implement complex ones. Signal ξ is said to be *basic*, if the sequence of elementary moves $(\xi(t + 1) - \xi(t))_{t\geq 0}$ is ultimately periodic. It is *rightmoving* (*leftmoving*), if it never moves to the left (right), that is, $\xi(t + 1) \in \{\xi(t), \xi(t) + 1\}$ $(\xi(t + 1) \in \{\xi(t) - 1, \xi(t)\})$.

Example 4. The signal $\xi : \mathbb{N} \to \mathbb{Z}$ with $\xi(n) = \lfloor \frac{n}{3} \rfloor$ is basic, since the sequence $0, 0, 1, 0, 0, 1, \ldots$ of elementary moves is periodic (cf. Figure 6.17). □

Example 5. The signal $\xi : \mathbb{N} \to \mathbb{Z}$ with

$$\xi(0) = 0 \quad \text{and} \quad \xi(n) = \frac{1}{4} \cdot 2^{\lceil \log_2 n \rceil} - \left| n - \frac{3}{4} \cdot 2^{\lceil \log_2 n \rceil} \right|$$

is obviously not basic (cf. Figure 6.18). □

The next lemma clarifies the relation between basic signals and implementations.

Lemma 2. *A signal* ξ *is basic if and only if it can be implemented in a cellular space such that all cells not containing* ξ *are in the quiescent state* $(i \neq \xi(t) \iff c_t(i) = q_0)$.

With other words, a signal is basic if and only if it can be implemented without auxiliary signals.

Definition 6.

1. *Let* ξ *be a basic signal whose sequence of elementary moves after some time* n_0 *is periodic with period length* p. *Let* $u = \xi(t + p) - \xi(t)$, *for some* $t > n_0$.

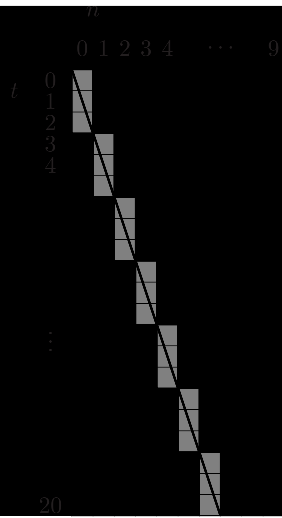

Fig. 6.17. The basic signal of Example 4.

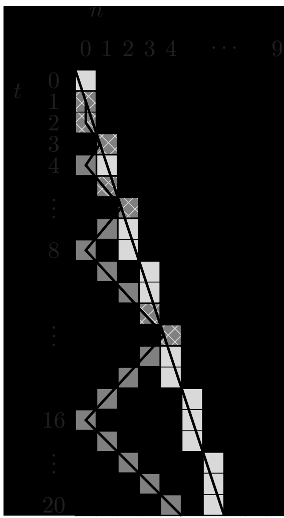

Fig. 6.18. Gray cells contain the signal of Example 5, lightgray cells a basic auxiliary signal.

a) The slope *of* ξ *is* p/u.
b) The speed *of* ξ *is* u/p.

2. A monotone increasing (decreasing) function $\rho : \mathbb{N} \to \mathbb{N}$ *is called* characteristic function *of a rightmoving (leftmoving) signal* ξ*, if* $\xi(\rho(n)) = n$ *and* $\xi(\rho(n) - 1) \neq n$.

Since the speed is at most 1, the slope is at least 1. The characteristic function takes a cell and yields the time step at which the signal arrives at the cell for the first time. Clearly, $\rho(n) \geq n$ for a characteristic function of a CS-practicable signal that is generated in cell 0.

6.4.2 Practicable Signals

In order to obtain a rich family of practicable signals we first show that certain classes of signals are practicable. Then we provide operations that preserve this property. So, one can construct new practicable signals from practicable ones by applying the operations.

Signals with exponential characteristic function

Lemma 3. *Let* $b \geq 2$ *be a positive integer. Then the signal* ξ *with characteristic function* b^n *is CS-practicable.*

Proof. At initial time signal ξ resides in cell 0. At each time step b^n, $n \geq 1$, it moves one cell to the right. To this end, two auxiliary signals α and β are used. In general, signals with speed $\frac{y}{x} \leq 1$ may be implemented by alternating y right moves and $x - y$ no moves. Signal α is generated at time $b-2$ in cell 0. Signal β is generated at time $\frac{1}{2}(b^2 + b - 2)$ in cell $\frac{1}{2}b(b-1)$ (cf. Figure 6.19). Whenever ξ meets α, signal ξ stays for one time step and then moves one cell to the right. Signal α also stays for one time step, and then it starts to move right with speed 1 until it meets β. Next, it moves back to the left with speed 1 until it meets ξ again. Initially and whenever β meets α, signal β moves b cells to the right within $b+1$ time steps. Subsequently, it moves with speed $\frac{(b-1)}{(b+1)}$ to the right.

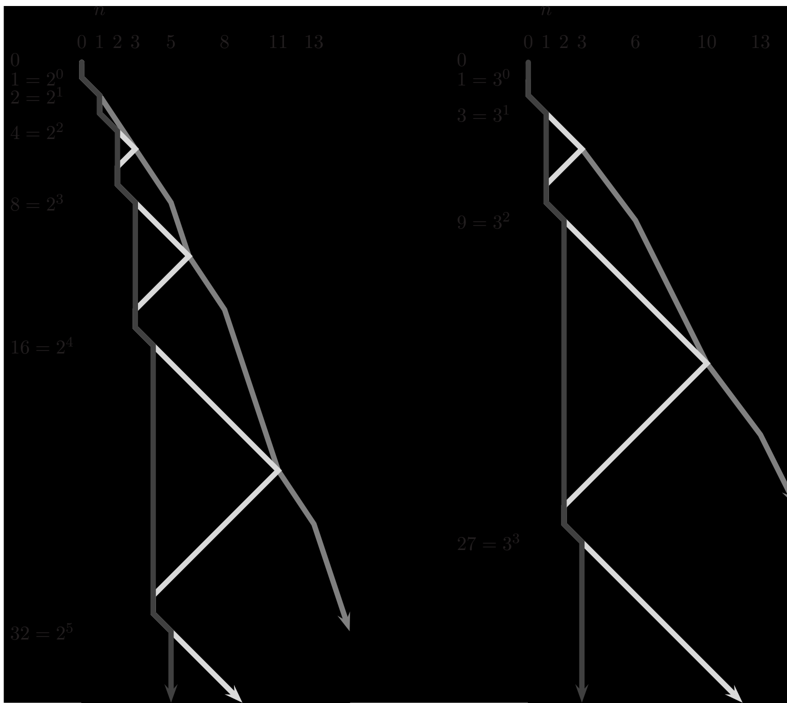

Fig. 6.19. Signals ξ with characteristic functions 2^n and 3^n (darkgray), auxiliary signals α (lightgray) and β (gray).

Exemplarily, the correctness of the construction is shown by induction. It is proved that α meets ξ at time $b^n - 2$ in cell $n-1$ and, subsequently, meets β at time $\frac{1}{2}(b^{n+1} + b^n - 2)$ in cell $n - 1 + \frac{1}{2}(b^n(b-1))$.

The induction basis $n = 1$ follows immediately from the generations of the signals. Assume now, the assertion is true for some $n \geq 1$. After meeting β, signal α meets ξ at time $\frac{b^{n+1}+b^n-2}{2} - 1 + \frac{b^n(b-1)}{2} = b^{n+1} - 2$ in cell n. At

time $b^{n+1}-1$ both signals stay in cell n. Subsequently, at time b^{n+1} they move to cell $n+1$. Next, signal α passes through cells $n+1+k$ at time steps $b^{n+1}+k$, $k=1,2,\dots$ Especially for $k=-1+\frac{1}{2}(b^{n+1}(b-1))$, signal α is in cell $n+\frac{1}{2}(b^{n+1}(b-1))$ at time $\frac{1}{2}(b^{n+2}+b^{n+1}-2)$.

After its last meeting with α, signal β first has moved b cells to the right within $b+1$ time steps. Next it started to move with speed $\frac{(b-1)}{(b+1)}$ to the right. Therefore, it passes through cells $n-1+\frac{1}{2}(b^n(b-1))+b+k(b-1)$ at time steps $\frac{1}{2}(b^{n+1}+b^n-2)+b+1+k(b+1)$, $k=1,2,\dots$ Especially for $k=\frac{1}{2}(b^{n+1}-b^n-2)$, signal β is in cell $n+\frac{1}{2}(b^{n+1}(b-1))$ at time $\frac{1}{2}(b^{n+2}+b^{n+1}-2)$. □

Signals with polynomial characteristic function

A signal with characteristic function n^2 can be derived from $(n+1)^2 = n^2+2n+1$. In particular, before signal ξ may move from cell n to $n+1$ it has to stay for $2n$ time steps in cell n. The delay is exactly the time needed by an auxiliary signal α that moves from cell n to cell 0 and back (cf. Figure 6.20). Proceeding inductively, a signal with characteristic function n^b can be implemented by utilizing auxiliary signals with polynomial characteristic functions whose degrees are less than b.

Lemma 4. *Let $b \geq 1$ be a positive integer. Then the signal with characteristic function n^b is CS-practicable.*

Proof. Exemplarily, the construction for $b=3$ is shown, where an auxiliary signal with characteristic function n^2b is used (cf. Figure 6.21). Constructions for arbitrary b are straightforward.

First, we derive $(n+1)^3 = n^3+3n^2+3n+1$, and obtain the necessary time of delay. A signal with characteristic function n^3 has to stay for $3n^2+3n$ time steps in cell n before it moves to cell $n+1$. The delay $3n$ is exactly the time needed by an auxiliary signal α that moves from cell n to cell 0 and back, and once more to cell 0. Subsequently, in cell 0 a quadratic signal β is generated, which moves from cell 0 to cell n and back, and once more to cell n. □

Signals whose characteristic functions contain square roots

The problem whether the following lemma is true for $k=1$ was left open in [49]. It has been solved in [66].

Lemma 5. *Let $k \geq 1$ be a positive integer. Then the signal with characteristic function $kn+\lfloor\sqrt{n}\rfloor$ is CS-practicable.*

Signals whose characteristic functions contain logarithms

Lemma 6. *Let $b \geq 2$ be a positive integer. Then the signal with characteristic function $n+\lfloor\log_b(n)\rfloor$ is CS-practicable.*

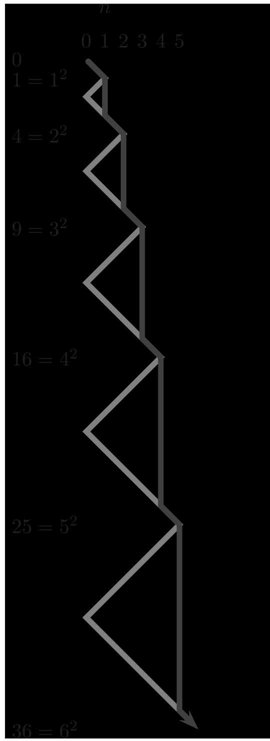

Fig. 6.20. Signal ξ with characteristic function n^2 (darkgray), auxiliary signal α (gray).

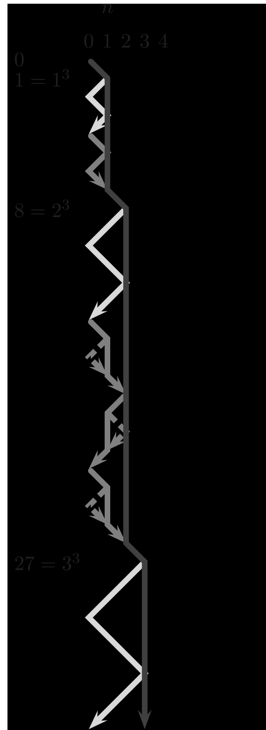

Fig. 6.21. Signal ξ with characteristic function n^3 (darkgray), auxiliary signals α (lightgray) and β (gray, gray dashed).

A gap in the family of practicable signals

Signals with characteristic functions of the form $n + \log_b(n)$ are lower bounds of CS-practicable signals beyond the identity (plus some constant). In between there is a gap.

Lemma 7. *Let $\rho(n) \geq n$, for all $n \geq 0$, be the characteristic function of a CS-practicable signal. Then $\rho(n) - n$ either is ultimately constant or there is some $b \geq 2$ such that $\rho(n) \geq n + \lfloor \log_b(n) \rfloor$, for all $n \geq 1$.*

Proof. Let $\mathcal{M}$ be a cellular space with state set S implementing the signal with characteristic function ρ. As usual, we denote its configurations by c_t, $t \geq 0$. We assume that $\rho(n) \geq n + \lfloor \log_b(n) \rfloor$ does *not* hold for all $b \geq 2$. In particular, it does not hold for $b = |S|$, where we may assume $|S| \geq 2$ without loss of generality. Therefore, there is an n_0 such that $\rho(n_0) < n_0 + \lfloor \log_b(n_0) \rfloor$. Since $\rho(n_0) \geq n_0$, we obtain $n_0 \geq b$.

Observe that due to the maximal speed of auxiliary signals, any cell $i \geq 0$ cannot participate in the implementation of the signal before time i. So, we consider the sequence of $m \geq 1$ successive states of some cell $i \geq 0$ beginning at time step i, that is, $c_i(i)c_{i+1}(i)\cdots c_{i+m-1}(i)$, and denote it by $w(i,m)$. The number of different sequences of length $\lfloor \log_b(n_0) \rfloor$ is at most n_0. Therefore, there are numbers $i \geq 0$ and $j \geq 1$ with $i+j \leq n_0$ such that $w(i, \lfloor \log_b(n_0) \rfloor) = w(i+j, \lfloor \log_b(n_0) \rfloor)$. This implies $w(\ell, \lfloor \log_b(n_0) \rfloor) = w(\ell + kj, \lfloor \log_b(n_0) \rfloor)$, for all $k \geq 0$ and $\ell \geq i$.

At time $\rho(n_0)$ the signal resides in cell n_0 which is indicated by a distinguished state. By $\rho(n_0) - n_0 < \lfloor \log_b(n_0) \rfloor$ follows that at time steps $\rho(n_0) + kj$ the cells $n_0 + kj$ are in the same state. Therefore, $\rho(n_0 + kj) = \rho(n_0) + kj$ and due to the maximal speed of signals we obtain $\rho(n_0 + k) = \rho(n_0) + k$, for all $k \geq 0$. We derive $\rho(n_0) - n_0 = \rho(n) - n$, for all $n \geq n_0$. Thus, ρ is ultimately constant. □

6.4.3 Time Constructibility

The investigation of time constructible functions in cellular spaces originates from [15], where a cellular space is constructed whose cell at the origin distinguishes exactly the time steps that are prime numbers. In [49] the systematic study of this concept was started. Since all values of a function have to be constructed, we consider strictly increasing functions. Initially, all cells except the one at the origin are quiescent.

Definition 7. *A strictly increasing function $f : \mathbb{N} \to \mathbb{N}$ is* CS-time-constructible *if there is a cellular space $\langle S, \delta, q_0, A, F \rangle$ with distinguished state $s \in S$ and initial configuration $c_0(0) = s$, $c_0(i) = q_0$, for $i \neq 0$, such that cell 0 is in some state from F at time t, if and only if $t = f(i)$ for some $i \geq 1$. The* family of CS-time-constructible functions *is denoted by $\mathscr{F}(CS)$.*

Lemma 8. *Let $b \geq 2$ be a positive integer. Then the function b^n is CS-time-constructible.*

Proof. In order to time construct the function b^n, an auxiliary signal β with speed $\frac{(b-1)}{(b+1)}$ is generated at time 0 in cell 0. It arrives at cells $\frac{kb(b-1)}{2}$ at time steps $\frac{kb(b+1)}{2}$. A second auxiliary signal α is generated at time b in cell 0. Subsequently, it repeatedly moves with speed 1 to the right until it meets β, bounces and moves with speed 1 back to cell 0. At its arrival cell 0 changes to some state from F.

If α leaves cell 0 at some time b^n, then it arrives at cell $\frac{1}{2}b^n(b-1)$ at time $b^n + \frac{1}{2}b^n(b-1)$. Exactly at this time signal β is in the same cell (for $k = b^{n-1}$). Therefore, signal α is back at cell 0 at time $b^n + b^n(b-1) = b^{n+1}$. □

At first glance, it seems that CS-time-constructible functions cannot grow faster than exponential functions. Among others, the next lemma says that this is a false impression.

Lemma 9. *1. The factorial function $n!$ is CS-time-constructible.*
2. The function that maps n to the nth prime number is CS-time-constructible.

The two families of CS-time-constructible functions and CS-practicable signals are very rich. Moreover, they are closely related. The next two results bridge the gap between the notions.

Theorem 2. *Let $h : \mathbb{N} \to \mathbb{N}$ be a strictly increasing function. If the signal with characteristic function h is CS-practicable, then h is CS-time-constructible.*

With other words, all characteristic functions of Section 6.4.2 are CS-time-constructible. Unfortunately, the converse is not true in general. For example, functions of the form $n + \lfloor \log^i \rfloor$, where $\log^i$ denotes the $i$fold iterated logarithm, $i \geq 2$, are CS-time-constructible. But by Lemma 7 they are not characteristic functions of CS-practicable signals. Nevertheless, for most of the relevant functions, the converse is true. Whenever the difference between $f(n)$ and n is at least linear, a corresponding signal can be derived from a CS-time-constructible function f.

Theorem 3. *Let f be a CS-time-constructible function. If $(k-1)f(n) \geq kn$, for some positive integer $k \geq 1$, then the signal with characteristic function f is CS-practicable.*

Finally, we summarize closure properties of the family $\mathscr{F}(\mathrm{CS})$ in order to be able to construct new functions by certain operations.

Theorem 4. *Let f and g be functions belonging to $\mathscr{F}(CS)$.*

1. *Let k be a positive rational constant such that $\lfloor k \cdot f \rfloor$ is strictly increasing. Then $\lfloor k \cdot f \rfloor$ belongs to $\mathscr{F}(CS)$.*
2. *The sum $f + g$ belongs to $\mathscr{F}(CS)$.*
3. *If $f(n) \geq g(n)$, for all $n \geq 1$, and $(k+1)f - kg$ is strictly increasing, for some positive integer $k \geq 1$, then the function $(k+1)f - kg$ belongs to $\mathscr{F}(CS)$.*
4. *The composition $f(g)$ belongs to $\mathscr{F}(CS)$.*

Further results about signals as well as time constructible and time computable functions can be found, for example, in [5, 6, 7, 13, 34, 68, 69].

6.5 Cellular Language Acceptors

Now we turn to one of the main branches in the theory of automata. Clearly, the data supplied to some device can be arranged as strings of symbols. Instances of problems to solve can be encoded as strings with a finite number of different symbols. Furthermore, complex answers to problems can be encoded

as binary sequences such that the answer is computed bit by bit. In order to compute one piece of the answer, the set of possible inputs is split into two sets associated with the binary outcome. From this point of view, the computational capabilities of the devices are studied in terms of string acceptance, that is, the determination to which of the two sets a given string belongs. These investigations are done with respect to and with the methods of language theory. For cellular spaces and automata they originated from [11, 12] and [61, 31]. Over the years substantial progress has been achieved, but there are still some basic open problems with deep relations to other fields.

6.5.1 Cellular Automata

Once we have a universal device there is a natural interest in realistic models that meet certain restrictions. Similar to the step from Turing machines to linear bounded automata, that is in terms of formal languages, from recursively enumerable to context-sensitive languages, the step from cellular spaces to cellular automata is to bound the number of available cells by the length of the input. For simplicity, the boundaries in space are modelled by a so-called permanent *boundary symbol* #. Due to the nearest neighbor connections, cells cannot communicate across a boundary. So, we may focus on the computations in between the boundaries and may disregard the computations outside. A widely studied question is to what extend one-way information flow reduces the computational capabilities of cellular automata. One-way information flow from right to left is achieved by providing the $\bar{H}_1$ neighborhood (cf. Example 2), that is, the next state of a cell depends on the current states of the cell itself and its immediate neighbor to the right.

Definition 8. *A* (one-dimensional) two-way cellular automaton *(CA) is a system* $\langle S, \delta, \#, A, F\rangle$*, where*

1. S *is the finite, nonempty set of* cell states*,*
2. $\# \notin S$ *is the permanent* boundary symbol*,*
3. $A \subseteq S$ *is the nonempty set of* input symbols*,*
4. $F \subseteq S$ *is the set of* final states*, and*
5. $\delta : (S \cup \{\#\}) \times S \times (S \cup \{\#\}) \to S$ *is the* local transition function*.*

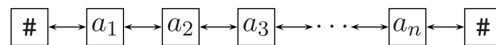

Fig. 6.22. A two-way cellular automaton.

If the flow of information is restricted to one-way, the resulting device is a *one-dimensional one-way cellular automaton* (OCA).

A *configuration* of a cellular automaton $\langle S, \delta, \#, A, F\rangle$ at time $t \geq 0$ is formally a mapping $c_t : \{1, \ldots, n\} \to S$, for $n \geq 1$. The configuration at

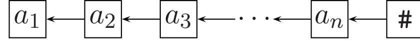

Fig. 6.23. A one-way cellular automaton.

time 0 is defined by the given input $w = a_1 \cdots a_n \in A^+$. We set $c_0(i) = a_i$, for $1 \le i \le n$. So, $\#a_1 a_2 \cdots a_n\#$ represents the initial configuration for w including the boundary symbols. Let c_t, $t \ge 0$, be a configuration with $n \ge 2$, then c_{t+1} is defined as follows:

$$c_{t+1} = \Delta(c_t) \iff \begin{cases} c_{t+1}(1) = \delta(\#, c_t(1), c_t(2)) \\ c_{t+1}(i) = \delta(c_t(i-1), c_t(i), c_t(i+1)), i \in \{2, \dots, n-1\} \\ c_{t+1}(n) = \delta(c_t(n-1), c_t(n), \#) \end{cases}$$

for CAs, and

$$c_{t+1} = \Delta(c_t) \iff \begin{cases} c_{t+1}(i) = \delta(c_t(i), c_t(i+1)), i \in \{1, \dots, n-1\} \\ c_{t+1}(n) = \delta(c_t(n), \#) \end{cases}$$

for OCAs. For $n = 1$, the next state of the sole cell is $\delta(\#, c_t(1), \#)$ or $\delta(c_t(1), \#)$.

6.5.2 Mode of Acceptance and Speed-Up

What is the result of the computation? One can partition the whole set of possible configurations into accepting and rejecting ones. This general approach is insufficient, since it could be much harder to determine whether a resulting configuration is accepting or not. So, it should be easy, say trivial, to recognize an accepting configuration. We define a configuration to be accepting when the cell receiving the first symbol of the input (cell 1) is in a final state from F. Further definitions of accepting configurations are studied, for example, in [26, 62], while more general input modes are considered in [42].

More precisely, an input w is accepted by an OCA, CA, or CS $\mathcal{M}$, if at some time during its course of computation cell 1 enters a final state. The *language accepted by* $\mathcal{M}$ is denoted by $L(\mathcal{M})$. Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \ge n$ be a mapping. If all $w \in L(\mathcal{M})$ are accepted within at most $t(|w|)$ time steps, then $L(\mathcal{M})$ is said to be of time complexity t. The family of languages that are accepted by OCAs (CAs, CSs) with time complexity t is denoted by $\mathscr{L}_t(\text{OCA})$ ($\mathscr{L}_t(\text{CA})$, $\mathscr{L}_t(\text{CS})$). The index is omitted for arbitrary time. Actually, arbitrary time in linearly space bounded devices is exponential time. If $t(n) = n$, acceptance is said to be in *real time* and we write $\mathscr{L}_{rt}(\text{OCA})$ ($\mathscr{L}_{rt}(\text{CA})$, $\mathscr{L}_{rt}(\text{CS})$). The *linear-time* languages $\mathscr{L}_{lt}(\text{OCA})$ are defined according to $\mathscr{L}_{lt}(\text{OCA}) = \bigcup_{k \in \mathbb{Q},\, k \ge 1} \mathscr{L}_{k \cdot n}(\text{OCA})$, and similarly for CAs and CSs.

In order to avoid technical overloading in writing, two languages L and L' are considered to be equal, if they differ at most in the empty word, that is, $L - \{\lambda\} = L' - \{\lambda\}$.

Example 6. The language $\{a^n b^n \mid n \geq 1\}$ is accepted by some OCA in real time (cf. Figure 6.24). During the first step, each cell with input symbol a changes into a state a'. In addition, the rightmost cell recognizes its position by means of the neighboring boundary symbol, and changes into a state r. Afterwards, at each time step the cell states b and r are shifted to the left. Whenever a b meets an a, the corresponding cell changes into state c. When r meets an a, the corresponding cell enters a final state R that is no longer shifted to the left. The construction is easily modified to reject inputs having a wrong format. □

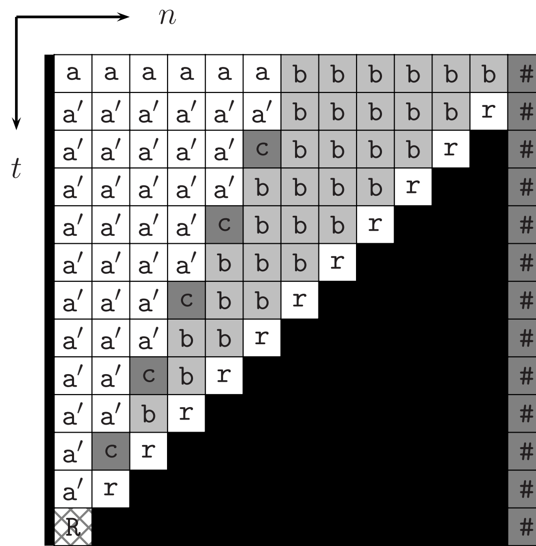

Fig. 6.24. Space-time diagram of an OCA accepting an input from the language $\{a^n b^n \mid n \geq 1\}$ in real time.

Helpful tools in connection with time complexities are speed-up theorems. Strong results are obtained in [24, 25], where the parallel language families are characterized by certain types of customized sequential machines. Among others, such machines have been developed for CSs, CAs, and OCAs. In particular, it is possible to speed up the time beyond real time linearly. Therefore, linear-time computations can be sped up close to real time. Later, the question whether real time can be achieved is discussed in detail later.

Theorem 5. *Let $\mathcal{M}$ be a CS, CA, or OCA obeying time complexity $rt+r(n)$, where $r : \mathbb{N} \to \mathbb{N}$ is a mapping and rt denotes real time. Then for all $k \geq 1$ an equivalent device $\mathcal{M}'$ of the same type obeying time complexity $rt + \lfloor \frac{r(n)}{k} \rfloor$ can effectively be constructed.*

The next example states that any constant beyond real time can be omitted.

Example 7. Let $k_0 \geq 1$ and $\mathcal{M}$ be a device in question with time complexity $rt + k_0$. Then there is an equivalent real-time device $\mathcal{M}'$ of the same type. It suffices to set $k = k_0 + 1$ and to apply Theorem 5 in order to obtain $rt + \lfloor \frac{k_0}{k} \rfloor = rt + \lfloor \frac{k_0}{k_0+1} \rfloor = rt$ for the time complexity of $\mathcal{M}'$. □

Next, a linear-time computation is sped up close to real time.

Example 8. Let $k_0 \geq 1$ and $\mathcal{M}$ be a device in question with time complexity $rt + k_0 \cdot rt$. Then for all rational numbers $\varepsilon > 0$ there is an equivalent device $\mathcal{M}'$ of the same type with time complexity $\lfloor (1+\varepsilon) \cdot rt \rfloor$. We set $k = \lceil \frac{k_0}{\varepsilon} \rceil$ and apply Theorem 5 in order to obtain $rt + \left\lfloor \frac{k_0 \cdot rt}{\lceil k_0/\varepsilon \rceil} \right\rfloor \leq rt + \left\lfloor \frac{k_0 \cdot rt}{k_0/\varepsilon} \right\rfloor = rt + \lfloor \varepsilon \cdot rt \rfloor = \lfloor (1 + \varepsilon) \cdot rt \rfloor$. □

6.5.3 Basic Hierarchy of Languages

The goal of this section is to establish a basic hierarchy of cellular language families, and to compare the levels with well-known families of the Chomsky hierarchy. The properness of some inclusions are long-standing open problems with deep relations to sequential complexity problems. In order to establish the hierarchy we start at the upper end.

In Theorem 1 it is shown how to simulate deterministic Turing machines by cellular spaces. Since the number of non-quiescent cells is just one more than the space complexity of the Turing machine, CAs can simulate linearly space-bounded Turing machines. Conversely, a straightforward construction of Turing machines from CSs and of linearly space-bounded Turing machines from CAs shows the following lemma [31].

Lemma 10. *The family $\mathscr{L}(CS)$ is identical with the recursively enumerable languages. The family $\mathscr{L}(CA)$ is identical with the complexity class* DSPACE(n)*, that is, with the deterministic context-sensitive languages.*

Corollary 1. *The family $\mathscr{L}(CA)$ is properly included in $\mathscr{L}(CS)$.*

The family $\mathscr{L}$(OCA) is very powerful. It contains the context-free languages as well as a PSPACE-complete language [8, 22]. For structural reasons it is contained in $\mathscr{L}$(CA). It is an open problem whether or not the inclusion is proper.

Corollary 2. *The family $\mathscr{L}(OCA)$ is included in $\mathscr{L}(CA)$.*

We continue with the lower end of the hierarchy, and consider the weakest devices in question, the real-time OCAs.

Lemma 11. *The regular languages are properly included in $\mathscr{L}_{rt}(OCA)$.*

Proof. Let L be a regular language represented by some deterministic finite automaton $\mathcal{E}$. We construct a real-time OCA $\mathcal{M}$ with two tracks that simulates $\mathcal{E}$. In fact, the first register of each cell is used to simulate $\mathcal{E}$, whereas the second track is used to shift the input to the left, that is, to feed it into the simulation of $\mathcal{E}$. So, the first register of the leftmost cell fetches the whole input and simulates $\mathcal{E}$ completely.

The properness of the stated inclusion follows from Example 6 which shows that the non-regular language $\{a^n b^n \mid n \geq 1\}$ belongs to $\mathscr{L}_{rt}(\text{OCA})$. □

In order to reach the next level of the hierarchy we consider unary languages. It turns out that even massively parallel OCAs with a certain time bound cannot accept more unary languages than a single deterministic finite automaton [59].

Lemma 12. *Let $L \subseteq \{a\}^+$ be a unary language accepted by some OCA $\mathcal{M}$. If for all $b \geq 2$ there is a $w_b \in L$ which is accepted by $\mathcal{M}$ in $t(|w_b|) < |w_b| + \lfloor \log_b(|w_b|) \rfloor$ time steps, then there are $k_0, k \geq 1$ such that $a^{k_0+m\cdot k} \in L$ for all $m \geq 0$.*

Proof. Let $\mathcal{M} = \langle S, \delta, \#, A, F \rangle$. In particular, for $b = (|S|+1)^3$ there exists a $w_b \in L$ whose length is denoted by n_0, and which is accepted in $t(n_0)$, $n_0 \leq t(n_0) < n_0 + \lfloor \log_{(|S|+1)^3}(n_0) \rfloor$, time steps. It follows $\lfloor \log_{(|S|+1)^3}(n_0) \rfloor \geq 1$, and thus $n_0 > |S|^3$. Moreover, we have $\lfloor n_0^{\frac{1}{2}} \rfloor > |S|$, for $|S| > 1$.

For convenience now we assume that the cells of the OCA are numbered from right to left. For a computation with initial configuration $\#a^{n_0}\#$ we consider the words $c_{n-1}(n)c_n(n)c_{n+1}(n)\cdots c_{n+\lfloor \log_{|S|^2}(n_0)\rfloor -1}(n)$, for all $1 \leq n \leq n_0$, and denote them by e_n. All these words have the same length $\lfloor \log_{|S|^2}(n_0) \rfloor + 1$. The number of different words is at most

$$\begin{aligned}
|S|^{\lfloor \log_{|S|^2}(n_0)\rfloor+1} &= |S| \cdot |S|^{\lfloor \log_{|S|^2}(n_0)\rfloor} \leq |S| \cdot \lfloor |S|^{\log_{|S|^2}(n_0)} \rfloor \\
&= |S| \cdot \lfloor |S|^{\frac{1}{2}\log_{|S|}(n_0)} \rfloor = |S| \cdot \lfloor (|S|^{\log_{|S|}(n_0)})^{\frac{1}{2}} \rfloor \\
&= |S| \cdot \lfloor n_0^{\frac{1}{2}} \rfloor < \lfloor n_0^{\frac{1}{2}} \rfloor \cdot \lfloor n_0^{\frac{1}{2}} \rfloor \leq n_0.
\end{aligned}$$

Therefore, at least two of $e_1, \ldots, e_{n_0}$ are identical, say e_i and e_j with $i < j$. Since initially all cells are in the same state, e_{n+1} is uniquely determined by e_n. So, $e_i = e_j$ implies $e_{n_0-(j-i)} = e_{n_0}$ and, furthermore, if the array is long enough, $e_{n_0+m(j-i)} = e_{n_0}$, for all $m \geq -1$. For $k_0 = n_0$ and $k = j - i$, $e_{k_0+m\cdot k} = e_{n_0}$ follows, for all $m \geq -1$. Since a^{n_0} is accepted in less than $n_0 + \lfloor \log_{(|S|+1)^3}(n_0) \rfloor$ time steps, word e_{n_0} contains an accepting state due to $\lfloor \log_{(|S|+1)^3}(n_0) \rfloor \leq \lfloor \log_{|S|^2}(n_0) \rfloor + 1$. Therefore, for all $m \geq 1$, input $a^{k_0+m\cdot k}$ is also accepted. □

For real-time computations a closer look at the proof of the previous lemma reveals the following lemma.

Lemma 13. *Each unary real-time OCA language is regular.*

Proof. Considering the proof of Lemma 12 in case of real time, one observes that the relevant information of the words e_n consists of the first two states only. Moreover, the first state appears in all cells to the left at the same time step. So, it is easy to construct an equivalent deterministic finite automaton with two registers that computes the first state of the next word e_{n+1} by applying the transition function to twice the current first state, and the second state of the next word e_{n+1} by applying the transition function to the current first state and the current second state. □

Example 9. In general, Lemma 12 cannot be used to prove that an accepted unary language is regular. For example, consider the non-regular language $L = \{a^{2^n} \mid n \geq 1\} \cup \{a^{2n-1} \mid n \geq 1\}$, and suppose there is an OCA accepting $\{a^{2^n} \mid n \geq 1\}$ with time complexity $t(n)$ that is at least of order $n + \log(n)$ (cf. Example 10). Clearly, the second subset $\{a^{2n-1} \mid n \geq 1\}$ which contains all words of odd length can be accepted in real time. So, an OCA accepting L by accepting the subsets on different tracks in parallel obeys the time complexity $t(n)$ if n is even, and real time if n is odd. Therefore, the conditions of Lemma 12 are met, and it is applicable for $k_0 = 1$ and $k = 2$. □

On the other hand, in particular cases Lemma 12 *can* be used to prove that a non-regular unary language is not accepted in less than $n+\log(n)$ time.

Theorem 6. *Let $r \in o(\log)$, $r : \mathbb{N} \to \mathbb{N}$, be a function. Then language $L = \{a^{2^n} \mid n \geq 1\}$ does not belong to $\mathscr{L}_{rt+r}(OCA)$.*

Proof. In contrast to the assertion, assume $L \in \mathscr{L}_{rt+r}(\text{OCA})$. Then, for all $b \geq 1$, there is a $w_b \in L$ which is accepted in $t(|w_b|) < |w_b| + \lfloor \log_b(|w_b|) \rfloor$ time steps. By Lemma 12 we conclude that there are $n_0, k \geq 1$, such that $a^{2^{n_0}} \in L$ and $a^{2^{n_0}+m \cdot k} \in L$, for all $m \geq 1$, which is a contradiction. □

The next example gives a tight bound for the OCA time complexity necessary to accept language $\{a^{2^n} \mid n \geq 1\}$.

Example 10. The following OCA $\mathcal{M} = \langle S, \delta, \texttt{\#}, A, F \rangle$ accepts the unary language $\{a^{2^n} \mid n \geq 1\}$ with time complexity $t(n) = n + \log(n)$.

The basic idea of the construction is to generate a binary counter in the rightmost cell with one step delay (cf. Figure 6.25). The counter moves to the left whereby the cells passed through are counted. The length of the counter is increased when necessary. In addition, cells which are passed through by the counter have to check whether all bits are 1. In this case the value of the counter is $2^n - 1$, for some $n \geq 1$. Due to the delayed generation this indicates a correct input length and the cell enters the final state. Clearly, the desired time complexity is obeyed. A formal construction is as follows.

$S = \{a, e, 1, \texttt{+}, 0, \overset{\bullet}{0}, \overset{+}{1}\}$, $A = \{a\}$, $F = \{\texttt{+}\}$, and for all $s_1, s_2 \in S$:

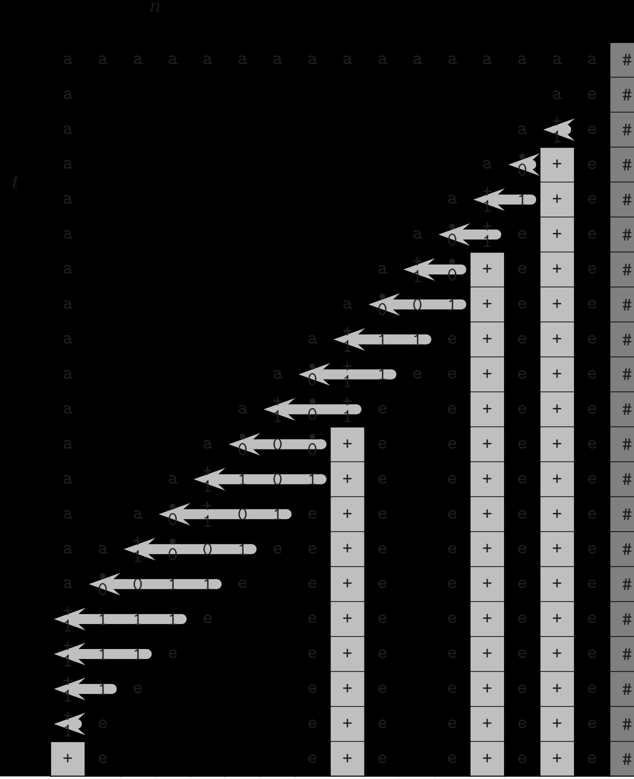

Fig. 6.25. Space-time diagram of an OCA accepting an input from the language $\{a^{2^n} \mid n \geq 1\}$ in $n + \log(n)$ time. Lightgray arrows mark the moving counter, whose digits are 0, 1, or $\overset{\bullet}{0}$. The latter is a 0 reporting a carry-over. A $\overset{+}{1}$ indicates that, so far, the cell has been passed through by 1s only.

$$\delta(s_1, s_2) = \begin{cases} e \text{ if } \big(s_1 \notin \{\overset{\bullet}{0}, \overset{+}{1}, +, a\} \wedge s_2 \in \{e, +\}\big) \vee \big(s_1 = a \wedge s_2 = \#\big) \\ + \text{ if } \big(s_1 = \overset{+}{1} \wedge s_2 = e\big) \\ \overset{+}{1} \text{ if } \big(s_1 = a \wedge s_2 \in \{e, \overset{\bullet}{0}\}\big) \vee \big(s_1 = \overset{+}{1} \wedge s_2 = 1\big) \\ \overset{\bullet}{0} \text{ if } \big(s_1 = a \wedge s_2 = \overset{+}{1}\big) \vee \big(s_1 = \overset{\bullet}{0} \wedge s_2 \in \{\overset{+}{1}, 1\}\big) \\ 0 \text{ if } \big(s_1 \neq a \wedge s_2 \in \{\overset{\bullet}{0}, 0\}\big) \\ 1 \text{ if } \big(s_1 = \overset{\bullet}{0} \wedge s_2 \in \{0, e, +\}\big) \vee \big(s_1 \neq \overset{+}{1} \wedge s_2 = 1\big) \\ s_1 \text{ otherwise} \end{cases}$$

□

Corollary 3. *The family $\mathscr{L}_{rt}(OCA)$ is properly included in $\mathscr{L}_{rt+\log}(OCA)$.*

For structural reasons, the next inclusion follows immediately. Its properness and, in fact, infinite proper hierarchies in between $\mathscr{L}_{rt}(\text{OCA})$ and $\mathscr{L}_{lt}(\text{OCA})$ have been shown in [37].

Corollary 4. *The family $\mathscr{L}_{rt+\log}(OCA)$ is properly included in $\mathscr{L}_{lt}(OCA)$.*

Since real-time and linear-time CSs use at most linearly many cells, they can be simulated by real-time and linear-time CAs. So, we do not need to consider them separately. Once we know that, in general, a linear-time OCA language cannot be accepted by any real-time OCA, the question arises whether two-way information flow can help in this respect. The next result gives a (partial) answer [9, 71]. The answer is not complete, since the input has to be reversed. Alternatively, one could reverse the neighborhood of the cells in an OCA. Then the rightmost cell indicates the result of the computation. In this case the input could remain as it is. In any case, the condition cannot be relaxed since it is an open problem whether the corresponding language families are closed under reversal.

Theorem 7. *A language is accepted by a linear-time OCA if and only if its reversal is accepted by a CA in real time.*

Proof. Let $\mathcal{M}$ be a real-time CA. The cells of a linear-time OCA $\mathcal{M}'$ accepting $L^R(\mathcal{M})$ collect the information necessary to simulate one transition of $\mathcal{M}$ in an intermediate step. Therefore, the first step of $\mathcal{M}$ is simulated in the second step of $\mathcal{M}'$. We obtain a behavior as depicted in Figure 6.26.

Altogether, $\mathcal{M}'$ cannot simulate the last step of $\mathcal{M}$. So, the construction has to be extended slightly. Each cell has an extra register that is used to simulate transitions of $\mathcal{M}$ under the assumption that the cell is the leftmost one (cf. Figure 6.27). The transitions of the real leftmost cell now correspond to the missing transitions of the previous simulation. □

It turned out that for OCAs linear time is strictly more powerful than real time. The problem is still open for CAs. The next inclusions follow for structural reasons and by the closure of $\mathscr{L}_{lt}(\text{CA})$ under reversal.

Corollary 5. *Any linear-time OCA language as well as its reversal belong to $\mathscr{L}_{lt}(CA)$.*

Now we can join the upper and the lower part of the hierarchy. The question whether or not one-way information flow is a strict weakening of two-way information flow for unbounded time is a long-standing open problem. Even the inclusion does not follow for structural reasons. It is proved in [8, 22] in terms of simulations of equivalent sequential machines. In the same paper it is shown that a PSPACE-complete language is accepted by OCAs. In fact, it is an open question whether real-time CAs are strictly weaker than unbounded time CAs. If both classes coincide, then a PSPACE-complete language would be accepted in polynomial time! The basic hierarchy obtained is depicted in Figure 6.31 on page 221.

Theorem 8. *The family $\mathscr{L}_{lt}(CA)$ is included in $\mathscr{L}(OCA)$.*

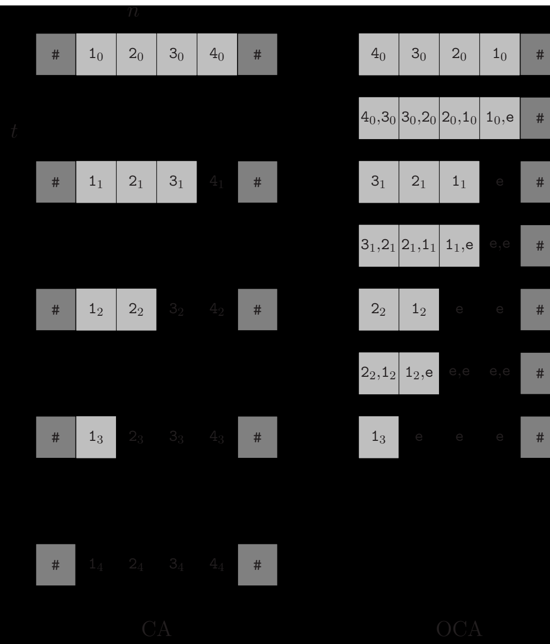

Fig. 6.26. Intermediate steps in the construction of the proof of Theorem 7.

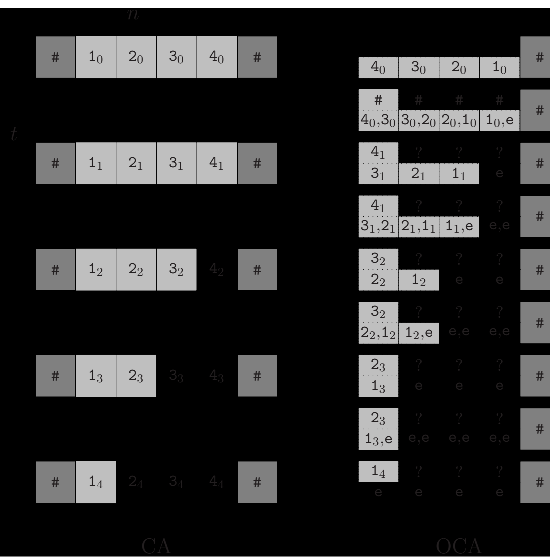

Fig. 6.27. Example of a linear-time OCA simulation of a real-time CA computation on reversed input.

6.5.4 Relations to Context-Free Languages

The relations between the language families in question and the regular, (deterministic) context-sensitive and recursively enumerable languages of the Chomsky hierarchy are quite clear. But what about the context-free languages? In [8] it is shown that they are properly included in the family $\mathscr{L}(\text{OCA})$. On the other hand, the family $\mathscr{L}_{rt}(\text{OCA})$ is incomparable with the family of context-free languages [65] since it contains, for example, the language $\{a^n b^n c^n \mid n \geq 1\}$, and does not contain the two-linear language LL with

$$L = \{a^n b^n \mid n \geq 1\} \cup \{a^n bwab^n \mid w \in \{a,b\}^*, n \geq 1\}.$$

Theorem 9.

1. *The context-free languages are properly included in the family $\mathscr{L}(OCA)$.*
2. *The family of context-free languages is incomparable with the families $\mathscr{L}_{rt}(OCA)$ and $\mathscr{L}_{rt+\log}(OCA)$.*

Nevertheless, even the real-time OCA languages contain important subfamilies, for example, the linear context-free languages [61], the Dyck languages [59], and the bracketed context-free languages [14]. Furthermore, the non-semilinear language $\{(a^i b)^* \mid i \geq 0\}$ [59] and the inherently ambiguous language $\{a^i b^j c^k \mid i = j \text{ or } j = k \text{ for } i,j,k \geq 1\}$ [31] belong to $\mathscr{L}_{rt}(\text{OCA})$.

Whether or not the context-free languages are included in the family $\mathscr{L}_{rt}(\text{CA})$ is an open question raised in [31]. It is related to the open question whether or not sequential one-tape Turing machines are able to accept the context-free languages in square-time. A proof for the inclusion would imply the existence of square-time Turing machines. In fact, also the problem whether or not the context-free languages are included in $\mathscr{L}_{lt}(\text{CA})$ is open. But for the important metalinear and deterministic context-free languages we can answer the inclusion problem in the affirmative [40].

Theorem 10. *The metalinear languages are properly included in the family $\mathscr{L}_{rt}(CA)$.*

Proof. Let L be a metalinear language. Then there exists a $k \geq 1$ such that L is k-linear. Therefore, we can represent L as union of finitely many concatenations $L_1 \cdot L_2 \cdot \cdots \cdot L_k$, where each L_i is a linear context-free language. The family $\mathscr{L}_{rt}(\text{CA})$ is closed under union. The family $\mathscr{L}_{rt}(\text{OCA})$ is closed under reversal [59]. Since the linear context-free languages [61] belong to the family $\mathscr{L}_{rt}(\text{OCA})$, there exist real-time OCAs for each of the languages $L_1^R, \ldots, L_k^R$. Since the concatenation of a real-time and a linear-time OCA language is again a linear-time OCA language [22], we obtain $L_k^R \cdot \cdots \cdot L_1^R \in \mathscr{L}_{lt}(\text{OCA})$. From the equality $\mathscr{L}_{lt}(\text{OCA}) = \mathscr{L}_{rt}^R(\text{CA})$ it follows $L_1 \cdot \cdots \cdot L_k = L \in \mathscr{L}_{rt}(\text{CA})$. □

Theorem 11. *The deterministic context-free languages are properly included in the family $\mathscr{L}_{rt}(CA)$.*

Proof. Here we cannot use an ordinary stack simulation because we are concerned with deterministic pushdown automata that are allowed to perform λ-transitions. But without loss of generalization we may assume that a given deterministic pushdown automaton $\mathcal{M}$ pushes at most $k \geq 1$ symbols onto the stack in every non-λ-transition, and erases exactly one symbol from the stack in every λ-transition [17]. Moreover, the first transition is a non-λ-transition.

By the equality $\mathscr{L}_{lt}(\text{OCA}) = \mathscr{L}_{rt}^{R}(\text{CA})$ it suffices to construct a $((k+1)\cdot n)$-time OCA $\mathcal{M}'$ that accepts the language $L^R(\mathcal{M})$. To this end, let $\mathcal{M}$ be a deterministic pushdown automaton with state set S, set of stack symbols G, set of input symbols A, initial state s_0, bottom-of-stack symbol $\perp \in G$, set of accepting states F, and transition function $\delta : S \times G \times (A \cup \{\lambda\}) \to S \times G^*$.

Now we construct the OCA $\mathcal{M}' = \langle S', \delta', \texttt{\#}, A, F' \rangle$.

Each cell of $\mathcal{M}$ has $k + 2$ registers, where the first one can store either an input symbol, or a distinguished special symbol $\texttt{\$}$, or a state of $\mathcal{M}$. The second register is used to implement a finite counter with range 0 to k. The remaining k registers can store stack symbols of $\mathcal{M}$ and may be empty. Accordingly, S' is defined to be $(S \cup A \cup \{\texttt{\$}\}) \times \{0, \dots, k\} \times (G \cup \{\lambda\})^k$. The transition function δ' ensures that at every time step $t \geq 1$ exactly one cell contains a symbol from S in its first register. This symbol is the current state of $\mathcal{M}$. So, F' is defined to be $F \times \{0, \dots, k\} \times (G \cup \{\lambda\})^k$.

Let $a_1 a_2 \cdots a_n$ be an input of $\mathcal{M}$. We consider $\mathcal{M}'$ when fed with the reverse input $a_n a_{n-1} \cdots a_1$. Initially all counter registers are set to 0, and all k registers for stack symbols are empty. Since the first transition of $\mathcal{M}'$ is a non-λ-transition and the rightmost cell can identify itself, for all $a \in A$, the initial step of $\mathcal{M}'$ is defined as follows (cf. Figure 6.28).

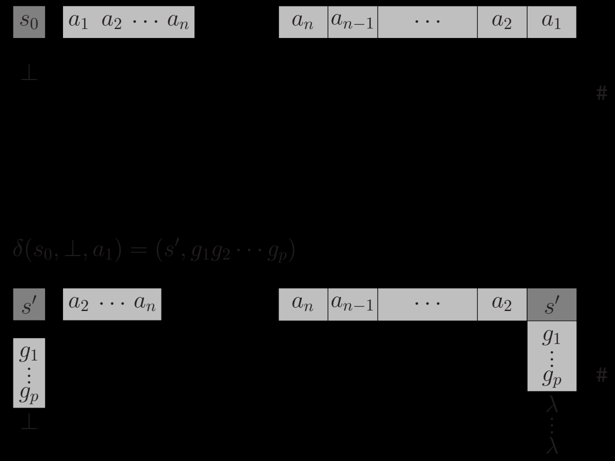

Fig. 6.28. The initial step of the pushdown automaton $\mathcal{M}$ (left) and the corresponding transition of the OCA $\mathcal{M}'$ (right). Counters are not depicted.

$$\delta(s_0, \bot, a) = (s', g_1 g_2 \cdots g_p) \iff \delta'((a, 0, \lambda^k), \#) = (s', 0, g_1 g_2 \cdots g_p \lambda^{k-p})$$

Proceeding inductively, at every time step there is exactly one distinguished cell containing the current state of $\mathcal{M}$ in its first register. All cells to its right are marked by the special symbol $\$$, and all cells to its left store still their input symbols (cf. Figures 6.28,6.29,6.30).

Every simulation of a transition of $\mathcal{M}$ is performed in two phases. During the first phase, the new state and the new symbols at the top of the stack of $\mathcal{M}$ are computed. (The first phase is indicated by 0 in the counter registers.) Let $\mathcal{M}$ perform a non-λ-transition (cf. Figure 6.29). The cell to the left of the distinguished cell has the necessary information. For all $a \in A$, $s \in S$, and $g_j \in G$,

$$\begin{aligned}&\delta(s, g_1, a) = (s', g'_1 g'_2 \cdots g'_p) \iff\\ &\delta'((a, 0, \lambda^k), (s, 0, g_1 g_2 \cdots g_k)) = (s', k-p, g'_1 g'_2 \cdots g'_p \lambda^{k-p}).\end{aligned}$$

All other cells of the left part keep their states. For all $a, \tilde{a} \in A$,

$$\delta'((a, 0, \lambda^k), (\tilde{a}, 0, \lambda^k)) = (a, 0, \lambda^k).$$

The distinguished cell observes that $\mathcal{M}$ does not perform a λ-transition. It stores the special symbol $\$$ in its first register. For all $s \in S$,

$$\begin{aligned}&\delta(s, g_1, a) \text{ is defined for some } a \in A \iff\\ &\delta'((s, 0, g_1 g_2 \cdots g_k), \#) = (\$, 0, g_2 g_3 \cdots g_k \lambda) \text{ and}\\ &\delta'((s, 0, g_1 g_2 \cdots g_k), (\$, 0, g_{k+1} g_{k+2} \cdots g_{2k})) = (\$, 0, g_2 g_3 \cdots g_k g_{k+1}).\end{aligned}$$

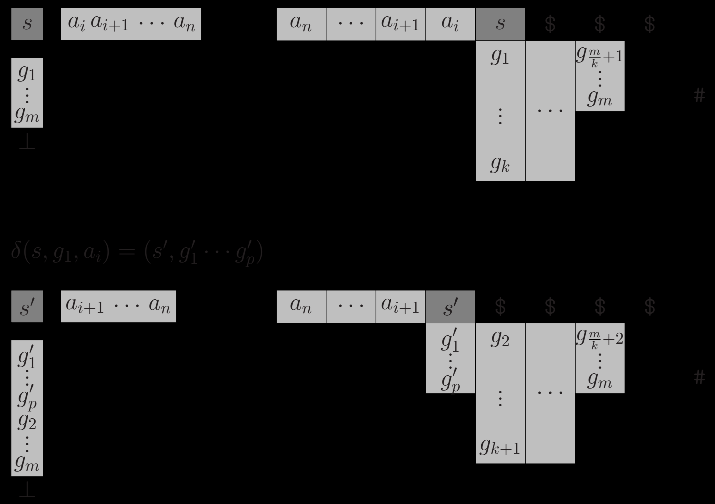

Fig. 6.29. A non-λ-transition of the pushdown automaton $\mathcal{M}$ (left) and the corresponding transition of the OCA $\mathcal{M}'$ (right). Counters are not depicted.

At each time step, cells containing the special symbol \$ shift the contents of the k stack symbol registers one position to the top where the last register is filled with the symbol shifted out by the right neighbor. For all $g_j \in G$,

$$\delta'((\$, 0, g_1 g_2 \cdots g_k), \#) = (\$, 0, g_2 g_3 \cdots g_k \lambda) \text{ and}$$
$$\delta'((\$, 0, g_1 g_2 \cdots g_k), (\$, 0, g_{k+1} g_{k+2} \cdots g_{2k})) = (\$, 0, g_2 g_3 \cdots g_k g_{k+1})$$

The purpose of the counter is to pack the stack symbols after a non-λ-transition. If the content of the counter is greater than 0, the second phase is performed in the distinguished cell. For all $s \in S$, $g_j \in G$, and $1 \leq i \leq k$,

$$\delta'((s, i, g_1 g_2 \cdots g_p \lambda^i), (\$, 0, g_{p+1} g_{p+2} \cdots g_{p+k})) = (s, i-1, g_1 g_2 \cdots g_p g_{p+1} \lambda^{i-1}).$$

If the counter has been decreased to 0, then the next transition of $\mathcal{M}$ is simulated. The distinguished cell as well as its left neighbor recognize whether it is a λ-transition. Since during λ-transitions the top-of-stack symbol is erased, from the above described behavior we get the packing for free (cf. Figure 6.30). For all $a \in A$, $s \in S$, and $g_j \in G$,

$$\delta(s, g_1, \lambda) \text{ is defined or } i > 0 \iff$$
$$\delta'((a, 0, \lambda), (s, i, g_1 g_2 \cdots g_k)) = (a, 0, \lambda)$$

$$\delta(s, g_1, \lambda) = (s', \lambda) \iff$$
$$\delta'((s, 0, g_1 g_2 \cdots g_k), \#) = (s', 0, g_2 g_3 \cdots g_k \lambda) \text{ and}$$
$$\delta'((s, 0, g_1 g_2 \cdots g_k), (\$, 0, g_{k+1} g_{k+2} \cdots g_{2k})) = (s', 0, g_2 g_3 \cdots g_k g_{k+1})$$

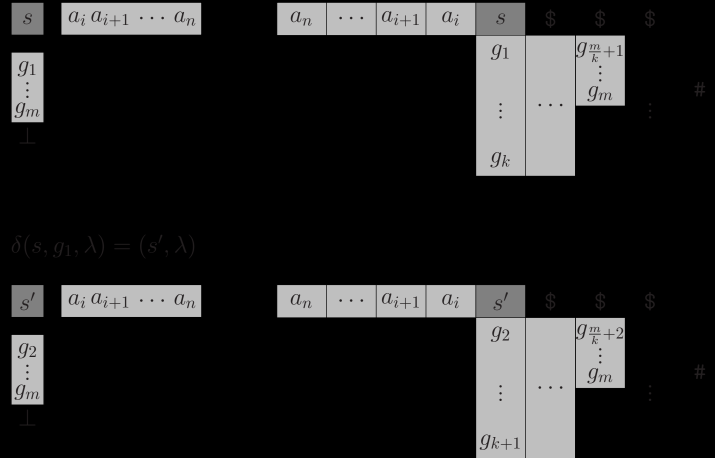

Fig. 6.30. A λ-transition of the pushdown automaton $\mathcal{M}$ (left) and the corresponding transition of the OCA $\mathcal{M}'$ (right). Counters are not depicted.

The OCA $\mathcal{M}'$ takes at most $(k+1) \cdot n$ time steps. It has to simulate n non-λ-transitions of $\mathcal{M}$. This takes n time steps. During each of these transitions some p symbols are pushed onto the stack which cause $k-p$ packing steps. In addition, there are at most p additional λ-transitions that erase the p symbols. So, every non-λ-transition causes at most k further steps. It follows that $\mathcal{M}'$ obeys the time complexity $(k+1) \cdot n$. □

Altogether we obtain the hierarchy depicted in Figure 6.31, where the only known proper inclusions are at the top and the lower end.

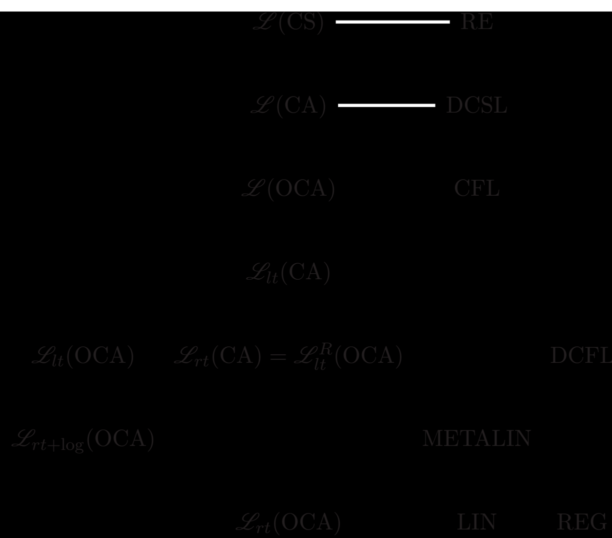

Fig. 6.31. Basic hierarchy of language families. A solid arrow indicates a proper inclusion, a dashed arrow an inclusion, and a double arrow an equality. Linear, metalinear, and deterministic context-free languages are denoted by LIN, METALIN, and DCFL. Regular, context-free, deterministic context-sensitive, and recursively enumerable languages are denoted by REG, CFL, DCSL, and RE.

6.5.5 Summary of Closure Properties and Decidability Problems

Finally, this subsection is devoted to summarize closure properties of and decidability results for the language families in question.

Closure properties

The closure properties of $\mathscr{L}$(CS) and $\mathscr{L}$(CA) are those of the recursively enumerable and deterministic context-sensitive languages. In [8, 22] strong closure properties are derived for the family of OCA languages. It is shown that

$\mathscr{L}$(OCA) is an AFL, that is, an abstract family of languages (cf., e.g., [58]) which is in addition closed under reversal.

The closure under reversal is of crucial importance. It is an open problem for $\mathscr{L}_{rt}$(CA) and, equivalently, for $\mathscr{L}_{lt}$(OCA). Moreover, it is linked with the open closure property under concatenation for the same family. If the answer to the open reversal closure of $\mathscr{L}_{rt}$(CA) is negative, we have to deal with two different language families. Since the properness of the inclusion $\mathscr{L}_{rt}(\text{CA}) \subseteq \mathscr{L}_{lt}(\text{CA})$ is also open, the problem gains in importance. A negative answer of the former problem would imply a proper inclusion. A language $L \in \mathscr{L}_{rt}(\text{CA})$ whose reversal does not belong to $\mathscr{L}_{rt}$(CA) may serve as witness since $\mathscr{L}_{lt}$(CA) is closed under reversal. In fact, the following stronger relation is shown in [23].

Theorem 12. *The family $\mathscr{L}_{rt}(CA)$ is closed under reversal if and only if $\mathscr{L}_{rt}(CA)$ and $\mathscr{L}_{lt}(CA)$ are identical.*

The question whether or not the family $\mathscr{L}_{rt}$(OCA) is closed under concatenation was open for a long time. It has been solved negatively in [64].

The question whether or not one of the families $\mathscr{L}_{rt}(\text{CA}) = \mathscr{L}_{lt}^{R}(\text{OCA})$ or $\mathscr{L}_{lt}$(CA) is closed under concatenation is another famous open problem in this field. Nevertheless, it is shown in [23] that the closure of $\mathscr{L}_{rt}$(CA) under reversal implies its closure under concatenation. Since in this case we obtain $\mathscr{L}_{rt}(\text{CA}) = \mathscr{L}_{lt}(\text{CA})$, the family of linear-time CA languages were also closed under concatenation. The concatenation closure for *unary* real-time CA languages has been solved in the affirmative [23].

Table 6.1 summarizes some closure properties of the language families in question.

Decidability problems

It is well known that all nontrivial decidability problems for Turing machines are undecidable [55]. Moreover, many of them are not even semidecidable, for example, neither finiteness nor infiniteness. Now we turn to summarize undecidable properties of cellular automata. Most of the early results are shown in [59] by reductions of Post Correspondence Problems. In terms of trellis automata the undecidability of emptiness, equivalence, and universality is derived in [28]. Here we present improved results that show the *non-semidecidability* of the properties. Almost all results in this section are proved in [43] by reductions of Turing machine problems. To this end, valid computations of Turing machines are considered. Roughly speaking, these are histories of accepting Turing machine computations which can be encoded in small grammars [20]. The generated languages are accepted by real-time OCAs.

Theorem 13. *For any language family that effectively contains $\mathscr{L}_{rt}(OCA)$ emptiness, universality, finiteness, infiniteness, equivalence, inclusion, context-freeness, and regularity are not semidecidable.*

	$\mathscr{L}_{rt}$(OCA)	$\mathscr{L}_{rt}$(CA)	$\mathscr{L}_{lt}$(CA)	$\mathscr{L}$(OCA)	$\mathscr{L}$(CA)	$\mathscr{L}$(CS)
$\cup$, $\cap$	+	+	+	+	+	+
complementation, $-$	+	+	+	+	+	$-$
reversal	+	?	+	+	+	+
concatenation	$-$	?	?	+	+	+
λ-free iteration	$-$	?	?	+	+	+
concatenation REG	+	+	?	+	+	+
REG concatenation	+	?	?	+	+	+
marked concatenation	+	+	+	+	+	+
marked λ-free iteration	+	+	+	+	+	+
hom^{-1}	+	+	+	+	+	+
deterministic gsm^{-1}	+	+	+	+	+	+
gsm^{-1}	$-$	?	?	+	+	+
inj. length-pres. *hom*	+	+	+	+	+	+
λ-free *hom*	$-$	?	?	+	+	+
λ-free *gsm*	$-$	?	?	+	+	+
λ-free substitution	$-$	?	?	+	+	+
hom	$-$	$-$	$-$	$-$	$-$	+

Table 6.1. Summary of closure properties. Concatenation REG denotes the concatenation with regular languages at the right, REG concatenation at the left, *hom* denotes homomorphisms, *gsm* generalized sequential machine mappings, and inj. length-pres. abbreviates injective length-preserving. A + indicates closure, a $-$ non-closure, and a question mark an open problem.

Next the question arises whether some structural properties of cellular language acceptors are (semi)decidable. For example, whether or not a real-time two-way language is a real-time one-way language. The questions turned out to be not even semidecidable.

Theorem 14. *For any language family $\mathscr{L}$ that effectively contains $\mathscr{L}_{rt}(CA)$ it is not semidecidable whether $L \in \mathscr{L}$ is a real-time OCA language.*

In general, a family $\mathscr{L}$ of languages possesses a pumping lemma in the narrow sense if for each $L \in \mathscr{L}$ there exists a constant $n \geq 1$ computable from L such that each $z \in L$ with $|z| > n$ admits a factorization $z = uvw$, where $|v| \geq 1$ and $u'v^iw' \in L$, for infinitely many $i \geq 0$. The prefix u' and the suffix w' depend on u, w and i.

Theorem 15. *Any language family whose word problem is semidecidable and that effectively contains $\mathscr{L}_{rt}(OCA)$ does not possess a pumping lemma (in the narrow sense).*

Theorem 16. *There is no minimization algorithm converting some CA or OCA (with arbitrary time complexity) to an equivalent automaton of the same type with a minimal number of states.*

Nevertheless, there are nontrivial decidable properties of cellular spaces. It is known that injectivity of the global transition function is equivalent to

the reversibility of the automaton. It is shown in [2] that global reversibility is decidable for one-dimensional CSs, whereas the problem is undecidable for higher dimensions [36].

References

1. K. Albert, Čulik II. A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1:1–16, 1987.
2. S. Amoroso and Y.N Patt. Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. *J. Comput. System Sci.*, 6:448–464, 1972.
3. R.M. Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Inform. Control*, 10:22–42, 1967.
4. E.R. Berlekamp, J.H. Conway, and R.K. Guy. Winning Ways for your Mathematical Plays. volume 2, chapter 25, Academic Press, 1982.
5. Th. Buchholz and M. Kutrib. On the power of one-way bounded cellular time computers. In *In Developments in Language Theory (DLT 1997)*, pp. 365-375, 1997.
6. Th. Buchholz and M. Kutrib. Some relations between massively parallel arrays. *Parallel Comput.*, 23:1643–1662, 1997.
7. Th. Buchholz and M. Kutrib. On time computability of functions in one-way cellular automata. *Acta Inform.*, 35:329–352, 1998.
8. J.H. Chang, O.H. Ibarra, and A. Vergis. On the power of one-way communication. *J. ACM*, 35:697–726, 1998.
9. C. Choffrut and K. Čulik II. On real-time cellular automata and trellis automata. *Acta Inform.*, 21:393–407, 1984.
10. E.F. Codd. *Cellular Automata*. Academic Press, New York, 1968.
11. S.N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machines. In *IEEE Symposium on Switching and Automata Theory (SWAT 1966)*.
12. S.N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machines. *IEEE Trans. Comput.*, pages 349–365, 1969.
13. J.C. Dubacq and V. Terrier. Signals for cellular automata in dimension 2 or higher. In *Theoretical Informatics (LATIN 2002)*, LNCS, vol. 2286, pp. 147-163.
14. C.R. Dyer. One-way bounded cellular automata. *Inform. Control*, 44:261–281, 1980.
15. P.C. Fischer. Generation of primes by a one-dimensional real-time iterative array. *J. ACM*, 12:388–394, 1965.
16. M. Gardner. Mathematical games. *Sci. Amer*, 224:112–117, 1971.
17. S. Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw Hill, New York, 1996.
18. E. Goto. A minimal time solution of the firing squad problem. Technical report.
19. A. Grasselli. Synchronization of cellular arrays: The firing squad problem in two dimensions. *Inform. Control*, 28:113–124, 1975.
20. J. Hartmanis. Context-free languages and Turing machine computations. In *Proceedings of the Symposia in Applied Mathematics, 19*, pages 42–51, 1967.
21. G.T. Herman, W.H. Liu, S. Rowland, and A. Walker. Synchronization of growing cellular arrays. *Inform. Control*, 25:103–122, 1974.

22. O.H. Ibarra and T. Jiang. On one-way cellular arrays. *SIAM J. Comput.*, 16:1135–1154, 1987.
23. O.H. Ibarra and T. Jiang. Relating the power of cellular arrays to their closure properties. *Theoret. Comput. Sci.*, 57:225–238, 1998.
24. O.H. Ibarra, S.M. Kim, and S. Moran. Sequential machine characterizations of trellis and cellular automata and applications. *SIAM J. Comput.*, 14:426–447, 1985.
25. O.H. Ibarra and M.A. Palis. Some results concerning linear iterative (systolic) arrays. *J. Parallel Distributed Comput.*, 2:182–218, 1985.
26. O.H. Ibarra, M.A. Palis, and S.M. Kim. Fast parallel language recognition by cellular automata. *Theoret. Comput. Sci.*, 41(2): 231-246, 1985.
27. K. Čulik II and S. Dube. An efficient solution of the firing mob problem. *Theoret. Comput. Sci.*, 91:57–69, 1991.
28. K. Čulik II, J. Gruska, and A. Salomaa. Systolic trellis automata: Stability, decidability and complexity. *Inform. Control*, 71:218–230, 1986.
29. K. Čulik II and S. Yu. Iterative tree automata. *Theoret. Comput. Sci.*, 32:227–247, 1984.
30. A.R. Smith III. Simple computation–universal cellular spaces. *J. ACM*, 18:339–353, 1971.
31. A.R. Smith III. Real-time language recognition by one-dimensional cellular automata. *J. Comput. System Sci.*, 6:233–253, 1972.
32. K. Imai and K. Morita. Firing squad synchronization problem in reversible cellular automata. *Theoret. Comput. Sci.*, 165:475–482, 1996.
33. K. Imai and K. Morita. A computation-universal two-dimensional 8-state triangular reversible cellular automaton. *Theoret. Comput. Sci.*, 231:181–191, 2000.
34. C. Iwamoto, T. Hatsuyama, K. Morita, and K. Imai. Constructible functions in cellular automata and their applications to hierarchy results. *Theoret. Comput. Sci.*, 270:797–809, 2002.
35. T. Jiang. The synchronization of nonuniform networks of finite automata. *Inform. Comput.*, 97:234–261, 1992.
36. J. Kari. Reversibility and surjectivity problems of cellular automata. *J. Comput. System Sci.*, 48:149–182, 1994.
37. A. Klein and M. Kutrib. Fast one-way cellular automata. *Theoret. Comput. Sci.*, 1(3):233–250, 2003.
38. K. Kobayashi. The firing squad synchronization problem for a class of polyautomata networks. *J. Comput. System Sci.*, 17:300–318, 1978.
39. K. Kobayashi. On the minimal firing time of the firing squad synchronization problem for polyautomata networks. *Theoret. Comput. Sci.*, 7:149–167, 1978.
40. M. Kutrib. *Automata arrays and context-free languages*, pages 139–148. Kluwer Academic Publishers, 2001.
41. M. Kutrib and R. Vollmar. The firing squad synchronization problem in defective cellular automata. *IEICE Trans. Inf. Syst.*, pages 895–900, 1995.
42. M. Kutrib and Th. Worsch. Investigation of different input modes for cellular automata. In Ch. Jesshope, V. Jossofov and W. Wihelmi, editors, *Parallel Processing by Cellular Automata and Arrays (Parcella 1994)*, Akademie Verlag, Berlin, pp. 141-150, 1994.
43. A. Malcher. Descriptional complexity of cellular automata and decidability questions. *J. Autom. Lang. Comb.*, 7:549–560, 2002.
44. M. Margenstern. Frontier between decidability and undecidability: a survey. *Theoret. Comput. Sci.*, 231:217–251, 2000.

45. B. Martin. Efficient unidimensional universal cellular automaton. In *Mathematical Foundations of Computer Science (MFCS 1992)*, volume 629 of *Lecture Notes in Computer Science*, pages 374–382, Berlin, 1992.
46. B. Martin. A universal cellular automaton in quasi-linear time and its S–m–n form. *Theoret. Comput. Sci.*, 123:199–237, 1994.
47. J. Mazoyer. A minimal time solution to the firing squad synchronization problem with only one bit of information exchanged. Technical report.
48. J. Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theoret. Comput. Sci.*, 50:183–238, 1987.
49. J. Mazoyer and V. Terrier. Signals in one-dimensional cellular automata. *Theoret. Comput. Sci.*, 217:53–80, 1999.
50. E.F. Moore. *The firing squad synchronization problem*, pages 213–214. Addison-Wesley, 1964.
51. F.R. Moore and G.C. Langdon. A generalized firing squad problem. *Inform. Control*, 12:17–33, 1968.
52. K. Morita. Computation-universality of one-dimensional one-way reversible cellular automata. *Inform. Process. Lett.*, 42:325–329, 1992.
53. K. Morita. Reversible simulation of one-dimensional irreversible cellular automata. *Theoret. Comput. Sci.*, 148:157–163, 1995.
54. K. Morita and S. Ueno. Computation-universal models of two-dimensional 16-state reversible cellular automata. *Trans. IEICE*, 75:141, 1992.
55. H.G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 89:25–59, 1953.
56. Y. Rogozhin. Small universal turing machines. *Theoret. Comput. Sci.*, 168:215–240, 1996.
57. P. Rosenstiehl, J.R. Fiksel, and A. Holliger. *Intelligent graphs: Networks of finite automata capable of solving graph problems*. Academic Press, New York.
58. A. Salomaa. *Formal Languages*. Academic Press, New York, 1973.
59. S.R. Seidel. Language recognition and the synchronization of cellular automata. Technical report.
60. I. Shinahr. Two- and three-dimensional firing-squad synchronization problems. *Inform. Control*, 24:163–180, 1974.
61. A.R. Smith. Cellular automata and formal languages. In *IEEE Symposium on Switching and Automata Theory (SWAT 1970)*. IEEE Press, pp. 216-224, 1970.
62. R. Sommerhalder and S.C. van Westrhenen. Parallel language recognition in constant time by cellular automata. *Acta Inform.*, 19:397–407, 1983.
63. H. Szwerinski. Time optimal solution of the firing squad synchronization problem for n-dimensional rectangles with the general at an arbitrary position. *Theoret. Comput. Sci.*, 19:305–320, 1982.
64. V. Terrier. On real time one-way cellular array. *Theoret. Comput. Sci.*, 141:331–335, 1995.
65. V. Terrier. Language not recognizable in real time by one-way cellular automata. *Theoret. Comput. Sci.*, 156:281–287, 1996.
66. V. Terrier. Construction of a signal of ratio $n + \lfloor\sqrt{n}\rfloor$. Unpublished manuscript, 2002.
67. H. Umeo. A simple design of time-efficient firing squad synchronization algorithms with fault-tolerance. *IEICE Trans. Inf. Syst.*, pages 733–739, 2004.
68. H. Umeo and N. Kamikawa. A design of real-time non-regular sequence generation algorithms and their implementations on cellular automata with 1-bit inter-cell communications. *Fund. Inform.*, 52:257–275, 2002.

69. H. Umeo and N. Kamikawa. Real-time generation of primes by a 1-bit-communication cellular automaton. *Fund. Inform.*, 58:421–435, 2003.
70. H. Umeo, M. Maeda, M. Hisaoka, and M. Teraoka. A state-efficient mapping scheme for designing two-dimensional firing squad synchronization algorithms. *Fund. Inform.*, 74:603–623, 2006.
71. H. Umeo, K. Morita, and K. Sugata. Deterministic one-way simulation of two-way real-time cellular automata and its related problems. *Inform. Process. Lett.*, 14:158–161, 1982.
72. J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press. Edited and completed by Arthur W. Burks.
73. A. Waksman. An optimum solution to the firing squad synchronization problem. *Inform. Control*, 9:66–78, 1996.