Automata 2010 — 16th Intl. Workshop on CA and DCS DMTCS proc. **AL**, 2010, 71–90

The Size of One-Way Cellular Automata

Martin Kutrib[1†] Jonas Lefèvre[2‡] Andreas Malcher[1†]

[1] *Institut für Informatik, Universität Giessen, Giessen, Germany*
[2] *Ecole Normale Supérieure de Lyon, Lyon, France*

We investigate the descriptional complexity of basic operations on real-time one-way cellular automata with an unbounded as well well as a fixed number of cells. The size of the automata is measured by their number of states. Most of the bounds shown are tight in the order of magnitude, that is, the sizes resulting from the effective constructions given are optimal with respect to worst case complexity. Conversely, these bounds also show the maximal savings of size that can be achieved when a given minimal real-time OCA is decomposed into smaller ones with respect to a given operation. From this point of view the natural problem of whether a decomposition can algorithmically be solved is studied. It turns out that all decomposition problems considered are algorithmically unsolvable. Therefore, a very restricted cellular model is studied in the second part of the paper, namely, real-time one-way cellular automata with a fixed number of cells. These devices are known to capture the regular languages and, thus, all the problems being undecidable for general one-way cellular automata become decidable. It is shown that these decision problems are NLOGSPACE-complete and thus share the attractive computational complexity of deterministic finite automata. Furthermore, the state complexity of basic operations for these devices is studied and upper and lower bounds are given.

Keywords: cellular automata, state complexity, descriptional complexity, formal languages, decidability

1 Introduction

Cellular automata are a well-motivated and well-investigated model for massively parallel computations which have widely been investigated from a computational capacity point of view (see, for example, the surveys [9, 10]). Basically, one-way cellular automata are linear arrays of identical copies of deterministic finite automata, sometimes called cells, that work synchronously at discrete time steps. Each cell is connected to its immediate neighbors to the right. The input is initially written into the cells.

Though real-time one-way cellular automata are one of the weakest classes of cellular automata, the class of languages accepted by them contains rather complicated non-context-free and non-semilinear languages and almost all important decidability questions turned out to be undecidable [16] and not even semidecidable [12].

Opposed to the computational capacity and complexity the descriptional complexity concerns the size of a system. One typical question is, for example, how succinctly a real-time one-way cellular automaton

†Email: {kutrib,malcher}@informatik.uni-giessen.de
‡Email: jonas.lefevre@ens-lyon.fr

can represent a language in comparison with other models. In [4, 6] more general introductions to and surveys of descriptional complexity are given. The descriptional complexity of real-time one-way cellular automata and the related model of real-time iterative arrays has been studied in [12, 14].

An important branch of descriptional complexity is the study of the complexity of operations. Here, one considers language operations such as union, intersection, or reversal, under which the language classes of the devices in question are closed. Of interest are optimal constructions with regard to the size of description. Thus, the goal is to find upper bounds that give the sufficient size necessary to represent the result of applying an operation, and lower bounds that give the sizes necessary in the worst case. Since, in general, the minimization or even reduction of the size for a given one-way cellular automaton is algorithmically unsolvable, there is no general method to prove the minimality of a given device. Moreover, the precise upper bounds on the size may depend on undecidable properties. So, tight bounds in the order of magnitude are to some extent best possible.

There are many ways to measure the size of a system. For deterministic finite automata the number of states is a reasonable and popular measure. Since basically the representation of a cellular automaton consists of the representation of their cells, the number of states is a reasonable size measure for cellular automata, too. For deterministic and nondeterministic finite automata the state complexity of many operations has been investigated. Recent surveys of results with regard to deterministic finite automata are [19, 20], where also operations on unary regular languages are discussed. In [1] an automata independent approach based on derivatives of languages is presented, that turned out to be a very useful technique for proving upper bounds for deterministic finite automata operations (cf. [2, 3]). A systematic study of language operations in connection with nondeterministic finite automata is [5]. The operation problem for two-way deterministic finite automata has been investigated recently in [8].

In this paper, we study the state complexity of real-time one-way cellular automata and we consider the Boolean operations union, intersection, and complementation as well as the operation of reversal. We obtain upper and lower bounds which are tight in order of magnitude. Interestingly, the state complexity of the Boolean operations is very similar to that of deterministic finite automata. This is not longer true for the operation of reversal. Here, deterministic finite automata have an exponential blow-up whereas the blow-up for real-time one-way cellular automata is at most quadratic. In contrast to the composition of two languages by using an operation, the somehow "inverse" problem of decomposing a given language into two languages with the help of an operation is studied. Clearly, the goal is to find a shorter representation of the given language by decomposition. It turns out that such decomposition problems are algorithmically unsolvable with regard to the Boolean operations and reversal.

These undecidability results together with the undecidability of almost all commonly investigated questions motivates the study of a restricted model, the real-time one-way cellular automata with a fixed number of cells, which have been introduced in [13]. The computational power of the restricted model is equivalent to the regular languages and, thus, all the problems undecidable for general one-way cellular automata become decidable. It is shown that these decision problems are NLOGSPACE-complete and thus share the attractive computational complexity of deterministic finite automata. Furthermore, the state complexity of basic operations for these devices is studied and upper and lower bounds are given.

2 Definitions

We denote the positive integers and zero $\{0, 1, 2, ...\}$ by $\mathbb{N}$. The empty word is denoted by λ, the reversal of a word w by w^R, and for the length of w we write $|w|$. For the number of occurrences of a subword x

in w we use the notation $|w|_x$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions. In order to avoid technical overloading in writing, two languages L and L' are considered to be equal, if they differ at most by the empty word, that is, $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$. Throughout the article two devices are said to be *equivalent* if and only if they accept the same language.

A one-way cellular automaton is a linear array of identical deterministic finite state machines, sometimes called cells. Except for the rightmost cell each one is connected to its nearest neighbor to the right. We identify the cells by positive integers. The state transition depends on the current state of a cell itself and the current state of its neighbor, where the rightmost cell receives information associated with a boundary symbol on its free input line. The state changes take place simultaneously at discrete time steps. The input mode for cellular automata is called parallel. One can suppose that all cells fetch their input symbol during a pre-initial step.

Definition 1 *A* one-way cellular automaton (*OCA*) *is a system* $\langle S, F, A, \#, \delta\rangle$, *where* S *is the finite, nonempty set of* cell states, $F \subseteq S$ *is the set of* accepting states, $A \subseteq S$ *is the nonempty set of* input symbols, $\# \notin S$ *is the permanent* boundary symbol, *and* $\delta : S \times (S \cup \{\#\}) \to S$ *is the* local transition function.

A *configuration* of a one-way cellular automaton $\langle S, F, A, \#, \delta\rangle$ at time $t \geq 0$ is a description of its global state, which is formally a mapping $c_t : \{1, 2, \ldots, n\} \to S$, for $n \geq 1$. The operation starts at time 0 in a so-called *initial configuration*, which is defined by the given input $w = a_1 a_2 \cdots a_n \in A^+$. We set $c_0(i) = a_i$, for $1 \leq i \leq n$. Successor configurations are computed according to the global transition function Δ. Let c_t, $t \geq 0$, be a configuration with $n \geq 2$, then its successor c_{t+1} is defined as follows:

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

For $n = 1$, the next state of the sole cell is $\delta(c_t(1), \#)$. Thus, Δ is induced by δ.

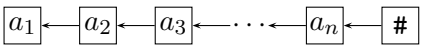

Fig. 1: A one-way cellular automaton.

An input w is accepted by an OCA $\mathcal{M}$ if at some time step during the course of its computation the leftmost cell enters an accepting state. The *language accepted by* $\mathcal{M}$ is denoted by $L(\mathcal{M})$. Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping. If all $w \in L(\mathcal{M})$ are accepted with at most $t(|w|)$ time steps, then $\mathcal{M}$ is said to be of time complexity t.

Observe that time complexities do not have to meet any further conditions. This general treatment is made possible by the way of acceptance. An input w is accepted if the leftmost cell enters an accepting state at some time $i \leq t(|w|)$. Subsequent states of the leftmost cell are not relevant. However, in the sequel we are particularly interested in fast OCAs operating in *real-time*, that is, obeying the time complexity $t(n) = n$.

So, any OCA is defined by the state set S, the set of input symbols A, the set of accepting states F, and the transition function. That means, for n states we have at most $2^n \cdot 2^n \cdot n^{n(n+1)}$ different OCAs, where, in addition, some of them are isomorphic. Since there are infinitely many languages acceptable by real-time OCAs, trivially, the number of states has to be unbounded.

3 State Complexity of Basic Operations

We consider the state complexities of the Boolean operations and reversal under which the class of languages accepted by real-time one-way cellular automata is closed [16]. First, we provide exemplarily an infinite language family over a binary alphabet that requires growing size when accepted by real-time OCAs. These languages and variants thereof are of tangible advantage for our purposes. As mentioned before, the problem is to prove a lower bound for the number of states necessary, since no general tools are available. Our lower bound misses the upper bound by one state only.

For all integers $k \geq 2$ let

$$L_k = \{\, 0^i a^{j \cdot k^i} \mid i, j \geq 1 \,\}.$$

Lemma 2 *Let $k \geq 2$ be an integer. Then $k+4$ states are sufficient for a real-time OCA to accept L_k.*

Proof: The language L_k is accepted by the real-time OCA $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$, where $A = \{a, 0\}$, $S = \{a, 0, 1, \ldots, k-1, <, +, -\}$, $F = \{+\}$, and

$$\begin{array}{ll}
\delta(a, \#) = - & \delta(p, a) = p+1,\ 0 \leq p \leq k-2 \\
\delta(a, -) = - & \delta(k-1, a) = < \\
\delta(<, -) = + & \delta(<, a) = 1 \\
\delta(<, +) = + & \delta(p, <) = p+1,\ 0 \leq p \leq k-2 \\
 & \delta(k-1, <) = < \\
 & \delta(<, q) = 0,\ 0 \leq q \leq k-1
\end{array}$$

Here and in the sequel we assume tacitly that the state is *not* changed whenever δ is not defined.

Basically, the idea of the construction is to set up a k-ary counter in the leftmost i cells, where states $0, 1, \ldots, k-1$ represent the digits and $<$ a carry-over to be processed by the left neighbor cell. The states $+$ and $-$ are used to implement a signal, which is initially started in the rightmost cell. It moves to the left, passes through the a-cells with a non-accepting state, and checks whether all cells of the counter passed through have been indicating a carry-over in the step before. Only in this case the accepting state is used. □

By almost the same reasoning the same upper bound for the complement $\overline{L_k}$ of L_k is shown.

Corollary 3 *Let $k \geq 2$ be an integer. Then $k+4$ states are sufficient for a real-time OCA to accept $\overline{L_k}$.*

Proof: We adapt the proof of Lemma 2 by defining $F = \{a, +\}$ and modifying δ such that $\delta(p, +) = +$, for all $p \in S$, $\delta(a, 0) = +$, $\delta(0, \#) = +$, $\delta(a, \#) = -$, $\delta(a, -) = -$, $\delta(<, -) = -$, and $\delta(p, -) = +$, for $0 \leq p \leq k-1$. □

Now we turn to the lower bounds.

Lemma 4 *Let $k \geq 2$ be an integer. Then at least $k+3$ states are necessary for a real-time OCA to accept L_k.*

Proof: Let $\mathcal{M}$ be a real-time OCA with state set S accepting L_k. We consider accepting computations on inputs of the form $0^i a^{j \cdot k^i}$ and, first, treat subcomputations as follows. The left part of sequences of adjacent a-cells runs through cycles according to $\delta(a,a) = a_1$, $\delta(a_1,a_1) = a_2$, $\delta(a_2,a_2) = a_3$, ... Denote the cycle length by c_a. Clearly, c_a is at most $|S|$. Therefore, for j large enough, the leftmost i cells initially carrying a 0 eventually also will run through cycles whose length is denoted by c_0. Finally, we have a possible signal from right to left initiated by $\delta(a,\#)$. Let $\delta(a,\#) = s_1$, $\delta(a_1,s_1) = s_2$, $\delta(a_2,s_2) = s_3$, ... Again, the signal eventually becomes cyclic with cycle length, say, c_s. Clearly, c_s is at most $|S|^2$.

Now we turn to states. Assume c_0 is at most $k^i - 1$. Since $0^i a^{k^i}$ is accepted, the input $0^i a^{k^i} a^{c_0 \cdot c_a \cdot c_s}$ must be accepted, too. But for i large enough, $(k^i - 1) \cdot c_a \cdot c_s = k^i \cdot c_a \cdot c_s - c_a \cdot c_s$ is not a multiple of k^i. Therefore, at least k states $z_1, z_2, \ldots, z_k$ are necessary to set up the cycle length c_0.

Furthermore, at least one state s_1 is necessary to realize the signal from right to left, where s_1 has to be different from a_1 and both are non-accepting states. Otherwise, there would be no signal and the whole computation could not accept in time. Clearly, neither $s_1, s_2, s_3, \ldots$ nor the states $a, a_1, a_2, a_3, \ldots$ and $z_1, z_2, \ldots, z_k$ can be accepting. So, in addition, one accepting state + is necessary.

If $a_1 \in \{z_1, z_2, \ldots, z_k\}$, at some time during the cycle the leftmost k cells are synchronously in state a_1 while further cells to their right are in state a_1 as well. So, some input belonging to L_k would be rejected. Similarly, if $s_1 \in \{z_1, z_2, \ldots, z_k\}$, then at some time the i-th cell from the left is in state s_1 and, thus, simulates the arrival of the signal, while the signal has not yet arrived. So, an input not belonging to L_k would be accepted.

Altogether, we have at least the $k+3$ states $\{z_1, z_2, \ldots, z_k, a_1, s_1, +\}$. □

As before, by almost the same reasoning the same lower bound for the complement $\overline{L_k}$ of L_k can be shown.

Corollary 5 *Let $k \geq 2$ be an integer. Then at least $k+3$ states are necessary for a real-time OCA to accept $\overline{L_k}$.*

3.1 Intersection and Union

Basically, the upper bounds for intersection and union are obtained by constructions based on the well-known two-track technique. That is, on two different tracks acceptors for both languages are simulated independent of each other. However, in general, an input is accepted when the leftmost cell enters an accepting state at some arbitrary time step. So, in general, the leftmost cell will enter accepting as well as non-accepting states during a computation. While this causes no problem for union, where a cell accepts when at least one of its registers is accepting, for the intersection, where a cell accepts when both of its registers are accepting, we have to provide further states. These are used to indicate that a register already has passed through an accepting state.

Theorem 6 *Let $m, n \geq 1$ be integers, $\mathcal{M}_1$ be an m-state real-time OCA with r_1 non-accepting states, and $\mathcal{M}_2$ be an n-state real-time OCA with r_2 non-accepting states. Then $m \cdot n + r_1 \cdot n + m \cdot r_2 + r_1 \cdot r_2 \in O(m \cdot n)$ states are sufficient for a real-time OCA to accept $L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$.*

Proof: We apply the two-track technique where each register remembers whether it has passed through an accepting state. First we modify $\mathcal{M}_i = \langle S_i, F_i, A, \#, \delta_i \rangle$, for $i \in \{1,2\}$ to $\hat{\mathcal{M}}_i = \langle \hat{S}_i, \hat{F}_i, A, \#, \hat{\delta}_i \rangle$,

where $\hat{S}_i = S_i \cup R_i$ with $R_i = \{\, s' \mid s \in S_i \setminus F_i \,\}$, $\hat{F}_i = F_i \cup R_i$, and

$$\hat{\delta}_i(s,t) = \begin{cases} \delta_i(s,t) & \text{if } s \in S_i \setminus F_i \\ \delta_i(s,t)' & \text{if } s \in F_i \text{ and } \delta_i(s,t) \in S_i \setminus F_i \\ \delta_i(s,t) & \text{if } s \in F_i \text{ and } \delta_i(s,t) \in F_i \end{cases},$$

$$\hat{\delta}_i(s',t) = \begin{cases} \delta_i(s,t)' & \text{if } s' \in R_i \text{ and } \delta_i(s,t) \in S_i \setminus F_i \\ \delta_i(s,t) & \text{if } s' \in R_i \text{ and } \delta_i(s,t) \in F_i \end{cases}.$$

Clearly, $\mathcal{M}_i$ and $\hat{\mathcal{M}}_i$ are equivalent. Now, $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$ accepts $L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$, where $S = \hat{S}_1 \times \hat{S}_2$, $F = \hat{F}_1 \times \hat{F}_2$, and $\delta((s_1,s_2),(t_1,t_2)) = (\hat{\delta}_1(s_1,t_1), \hat{\delta}_2(s_2,t_2))$. □

For the union the construction is slightly simpler. In this case it is not necessary to remember whether a register has passed through an accepting state. Therefore, the next upper bound follows immediately.

Theorem 7 *Let $m, n \geq 1$ be integers, $\mathcal{M}_1$ be an m-state real-time OCA and $\mathcal{M}_2$ be an n-state real-time OCA. Then $m \cdot n \in O(m \cdot n)$ states are sufficient for a real-time OCA to accept $L(\mathcal{M}_1) \cup L(\mathcal{M}_2)$.*

Now we can utilize the languages L_k for showing lower bounds which are tight in the order of magnitude.

Theorem 8 *Let $m, n \geq 6$ be integers such that $m-4$ and $n-4$ are relatively prime. Then at least $(m-4)(n-4)+3 \in \Omega(m \cdot n)$ states are necessary in the worst case for a real-time OCA to accept the intersection of an m-state real-time OCA and an n-state real-time OCA language.*

Proof: Let $k = m-4$ and $\ell = n-4$ be relatively prime. The witness languages for the assertion are L_k accepted by an m-state real-time OCA and L_ℓ accepted by an n-state real-time OCA. The intersection $L_k \cap L_\ell$ is $L_{k \cdot \ell} = \{\, 0^i a^{j \cdot k^i \cdot \ell^i} \mid i, j \geq 1 \,\}$. By Lemma 4, any real-time OCA accepting $L_{k \cdot \ell}$ has at least $k \cdot \ell + 3 = (m-4)(n-4)+3 \in \Omega(m \cdot n)$ states. □

Theorem 9 *Let $m, n \geq 6$ be integers such that $m-4$ and $n-4$ are relatively prime. Then at least $(m-4)(n-4)+3 \in \Omega(m \cdot n)$ states are necessary in the worst case for a real-time OCA to accept the union of an m-state real-time OCA and an n-state real-time OCA language.*

Proof: Let $k = m-4$ and $\ell = n-4$ be relatively prime. Now the witness languages for the assertion are $\overline{L_k}$ accepted by an m-state real-time OCA and $\overline{L_\ell}$ accepted by an n-state real-time OCA. Their union is $\overline{L_{k \cdot \ell}}$, for which at least $k \cdot \ell + 3 \in \Omega(m \cdot n)$ states are necessary by Corollary 5. □

3.2 Complementation

The precise upper bounds on the state complexity of some operations depend on the states that can appear on the diagonal of the space time diagram, that is, the states $c_i(n-i+1)$, $1 \leq i \leq n$. Given an OCA we consider the set of states D that can appear on the diagonal in some possible computation, and denote their number by d. For convenience, we simply write *states that can appear on the diagonal*.

For deterministic devices the closure under complementation is often shown by interchanging accepting and non-accepting states. The reason why this does not work in general for OCAs is once more that the leftmost cell may enter accepting as well as non-accepting states during a computation.

Theorem 10 *Let $n \geq 1$ be an integer and $\mathcal{M}$ be an n-state real-time OCA with r non-accepting states, d states that can appear on the diagonal and also at other positions, from which g are non-accepting. Then $n + r + d + g \in O(n)$ states are sufficient for a real-time OCA to accept $\overline{L(\mathcal{M})}$.*

Proof: We sketch the construction of a real-time OCA $\mathcal{M}'$ accepting $\overline{L(\mathcal{M})}$. Basically, $\mathcal{M}'$ simulates $\mathcal{M}$, but since the cells of $\mathcal{M}$ may enter accepting as well as non-accepting states during a computation, none of the states of $\mathcal{M}$ can be accepting in $\mathcal{M}'$. Instead, copies of the r non-accepting states are used in order to remember whether a cell has passed through an accepting state before. In order to accept the complement of $L(\mathcal{M})$ all of these new states are also non-accepting. Finally, it suffices to send a signal from right to left along the diagonal that causes every cell passed through that has not entered an accepting state before to accept. To this end, the states that appear on the diagonal have to be identified as signal. This is trivial for the states of $\mathcal{M}$ which appear only at the diagonal. For those d states that can appear on the diagonal and also at other positions (of $\mathcal{M}$), copies are used for this purpose. Furthermore, on the diagonal of $\mathcal{M}'$ there may appear g new non-accepting states indicating that the cell has entered an accepting state before. For these now new copies have to be used to accept. □

Theorem 11 *Let $n \geq 5$ be an integer. Then at least $2n - 3 \in \Omega(n)$ states are necessary in the worst case for a real-time OCA to accept the complement of an n-state real-time OCA language.*

Proof: For $k \geq 2$ the assertion is witnessed by the language

$$L_{c,k} = \{\, 0^i a^j \mid i \geq 1, j \geq k^i \,\}.$$

First, we construct a $(k+3)$-state real-time OCA $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$, which accepts it: $A = \{a, 0\}$, $S = \{0, 1, \ldots, k-1, a, <, -\}$, $F = \{<\}$, and

$$\begin{aligned}
&\delta(p, a) = p + 1,\ 0 \leq p \leq k-2 && \delta(k-1, <) = < \\
&\delta(k-1, a) = < && \delta(<, q) = 0,\ 0 \leq q \leq k-1 \\
&\delta(<, a) = 1 && \delta(a, \#) = - \\
&\delta(p, <) = p + 1,\ 0 \leq p \leq k-2 && \delta(a, -) = -
\end{aligned}$$

So, the real-time OCA $\mathcal{M}$ has $n = k + 3$ states, $r = k + 2$ non-accepting states, $d = k + 1$ states that can appear on the diagonal and at other positions, from which $g = k$ are non-accepting.

In order to show the lower bound on the number of states necessary to accept the complement of $L_{c,k}$, we argue as follows. At least k states are necessary to set up a cycle of length k^i in the i leftmost cells (cf. proof of Lemma 4). Clearly, these states are all non-accepting. Going into further details, at time step k cell i has to switch to a different set of at least k states. This is caused by the fact that the cycle of all the leftmost cells has to continue and, in addition, cell i can only change to an accepting state until time step k. In general, cell $1 \leq j \leq i$ has to switch to the different set of k states at time step $k^{i-j+1} + i - j$. Again, these new states are all non-accepting.

Furthermore, one additional state different from a is necessary to send a signal from right to left such that some cell initially carrying a 0 can change to an accepting state at all. Finally, an accepting state itself is necessary. In total, at least $2k+3 = 2n-3 \in \Omega(n)$ states are necessary. □

3.3 Reversal

Now we turn to the non-Boolean operation reversal.

Theorem 12 *Let $n \geq 1$ be an integer and $\mathcal{M}$ be an n-state real-time OCA with set of input symbols A and set D of states that can appear on the diagonal. Then $n \cdot |D| + |A| + 3 \in O(n^2)$ states are sufficient for a real-time OCA to accept $L(\mathcal{M})^R$.*

Proof: Let $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$ be real-time OCA with set D of states that can appear on the diagonal. In order to obtain a real-time OCA $\mathcal{M}'$ for the language $L(\mathcal{M})^R$, basically, the arguments of the local transition function are interchanged. In addition, we have to pay special attention to the boundary state. Moreover, $\mathcal{M}'$ cannot simulate the last step of $\mathcal{M}$ (see Figure 2). 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. The transitions of the real leftmost cell now correspond to the missing transitions of the previous simulation. However, the computation of the leftmost cell of $\mathcal{M}$ is simulated on the diagonal of $\mathcal{M}'$ together with the additional register. So, if an accepting state appears on the diagonal, it has to be sent to the left. On the other hand, if an accepting state appears in the additional register, it has to cause the cell to accept but must not be sent to the left. So, we conclude the construction of $\mathcal{M}'$ by providing a signal from right to left which collects the results, where state + is the accepting state to be sent to the left, $\oplus$ is the accepting state not to be sent to the left, and − is the non-accepting state of the signal. Formally, $\mathcal{M}' = \langle S', F', A, \#, \delta' \rangle$ is constructed as follows.

$S' = (D \times S) \cup A \cup \{+, \oplus, -\}, \quad F' = \{+, \oplus\}$,
for all $s_1, s_2 \in A$,
$\delta'(s_1, s_2) = (\delta(s_1, \#), \delta(s_2, s_1))$ and $\delta'(s_1, \#) = \begin{cases} + & \text{if } s_1 \in F \\ - & \text{if } s_1 \notin F \end{cases}$,
for all $d_1, d_2 \in D$, $s_1, s_2 \in S$,
$\delta'((d_1, s_1), (d_2, s_2)) = (\delta(s_1, d_1), \delta(s_2, s_1))$,
$\delta'((d_1, s_1), +) = +$,
$\delta'((d_1, s_1), -) = \delta'((d_1, s_1), \oplus) = \begin{cases} + & \text{if } s_1 \in F \\ \oplus & \text{if } s_1 \notin F \text{ and } \delta(s_1, d_1) \in F \\ - & \text{otherwise} \end{cases}$. □

Theorem 13 *Let $k \geq 2$ and n be an integer of the form $12k+7$. Then at least $\Omega(n^2)$ states are necessary in the worst case for a real-time OCA to accept the reversal of an n-state real-time OCA language.*

Proof: For $k \geq 2$, the witness language for the assertion is

$$L_{R,k} = \{\, w0 \mid w \in \{a,b\}^*, |w| \geq k^2, |w|_a \equiv 0 \bmod k, |w|_b \equiv 0 \bmod k \,\},$$

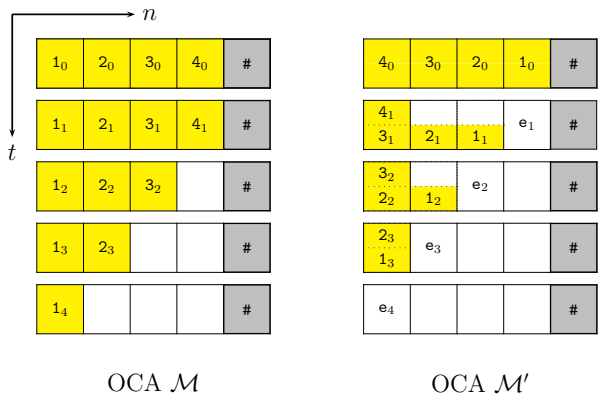

Fig. 2: Construction showing the simulation of a real-time OCA $\mathcal{M}$ by a real-time OCA $\mathcal{M}'$ on reversed input. The states e_1, e_2, e_3 are from $\{+, \oplus, -\}$. State e_4 depends on 1_3 and $1_4 = \delta(1_3, 2_3)$.

which is accepted by a $(12k + 7)$-state real-time OCA.

The formal construction is given below. We start with the idea of the construction. All cells initially carrying an a or b, behave as follows. In a first register they shift their input successively to the left. In a second register, they remember their original input. In a third register they count modulo k the number of input symbols shifted through that correspond to their own input symbol, that is, an a-cell counts all incoming symbols a and its own a, a b-cell counts all incoming symbols b and its own b. This behavior is realized by the first group of transition rules below.

In addition, initially a k-ary counter with two digits is set up at the right end. The first digit is initialized by the transitions $\delta(a, 0)$ or $\delta(b, 0)$ while the second digit is initialized by the transition $\delta(0, \#)$. The counter moves to the left. In addition to counting, both digits have two further registers. One register of the first digit indicates by + or − whether the last a-cell passed through has counted a number of a-symbols that is congruent 0 modulo k. The other register does the same for b-cells. This behavior is realized by the second group of transition rules below.

In order to distinguish between the first and the second digit of the moving counter, the second digit is primed. On every step to the left, the cell carrying the second digit simply takes the contents of the indicator registers of the first digit into their own indicator registers, and counts until a carry-over appears. Subsequently, the digit changes to an indicator + in its counting register which says that the counter has passed through at least k^2 cells. So, the cell carrying the second digit is in an accepting state if and only if the indicator + is in all of its registers. The behavior of the second digit is realized by the third group of transition rules below.

More precisely, the language $L_{R,k}$ is accepted by the real-time OCA $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$, where $S = \{a, b, 0\} \cup \{a, b\} \times \{a, b\} \times \{0, 1, \ldots, k-1\} \cup \{+, -\} \times \{+, -\} \times \{0, 1, \ldots, k-1, 0', 1', \ldots, (k-1)', +\}$,

$A = \{a, b, 0\}$, $F = \{(+,+,+)\}$, and, for $x, y, s, t \in \{a, b\}$, $p, q \in \{0, 1, \ldots, k-1\}$,

$$\begin{aligned}
&\delta(a,a) = (a, a, 2 \bmod k) && \delta((x,s,p),(y,t,q)) = (y, s, (p+1) \bmod k), \text{ if } s = y\\
&\delta(a,b) = (b, a, 1) && \delta((x,s,p),(y,t,q)) = (y, s, p), \text{ if } s \neq y\\
&\delta(b,b) = (b, b, 2 \bmod k)\\
&\delta(b,a) = (a, b, 1)
\end{aligned}$$

and for $x, s \in \{a, b\}$, $u, v \in \{+, -\}$, $p, q \in \{0, 1, \ldots, k-1\}$,

$$\begin{aligned}
&\delta(a,0) = (-,+,1)\\
&\delta(b,0) = (+,-,1)\\
&\delta((x,s,p),(u,v,q)) = \begin{cases} (+, v, (q+1) \bmod k) & \text{if } s = a \text{ and } p = 0\\ (-, v, (q+1) \bmod k) & \text{if } s = a \text{ and } p \neq 0\\ (u, +, (q+1) \bmod k) & \text{if } s = b \text{ and } p = 0\\ (u, -, (q+1) \bmod k) & \text{if } s = b \text{ and } p \neq 0 \end{cases}
\end{aligned}$$

and for $u, v, w, z \in \{+, -\}$,

$$\begin{aligned}
&\delta(0, \#) = (+, +, 0')\\
&\delta((u,v,p),(w,z,q')) = (u, v, q'),\ 1 \leq p \leq k-1, 0 \leq q \leq k-1\\
&\delta((u,v,0),(w,z,q')) = (u, v, (q+1)'),\ 0 \leq q \leq k-2\\
&\delta((u,v,0),(w,z,(k-1)')) = (u, v, +)\\
&\delta((u,v,p),(w,z,+)) = (u, v, +),\ 0 \leq p \leq k-1
\end{aligned}$$

Without further proof we state that any real-time OCA accepting the reversal $L^R_{R,k}$ needs at least $\Omega(k^2) = \Omega(n^2)$ states. □

Since the upper bounds on the state complexity of complementation and reversal depend on the states that can appear on the diagonal of the space time diagram, it is natural to ask whether the constructions are effective. That is, to ask whether it is decidable which states appear on the diagonal. More general, the decidability of reachability problems such as whether there is an input and a time step at which a given configuration is reached by a given real-time OCA, or at which time a certain cell enters a given state, are of particular interest. We will show that the first question is decidable whereas the latter is undecidable.

Lemma 14 *Let $\mathcal{M} = \langle S, F, A, \#, \delta\rangle$ be a real-time OCA, $n \geq 1$ be an integer, and $c : \{1, 2, \ldots, n\} \to S$ be a configuration. Then it is decidable whether there is an input $w \in A^n$ such that $\mathcal{M}$ reaches c on input w.*

Proof: We consider a brute-force algorithm which generates successively all inputs of length n, simulates the computation of $\mathcal{M}$ on these inputs until it becomes cyclically at latest at time step $|S|^n$, and finally checks whether the given configuration c occurred. As soon as such an input has been identified, the algorithm stops and returns *yes*. Otherwise, the algorithm stops after having negatively checked all possibilities and returns *no*. □

Lemma 15 *Let $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$ be a real-time OCA, s be a state from S, and $i \geq 1$ be a cell. Then it is undecidable whether there is an input $w \in A^*$ such that on input w cell i enters state s at some time $t \geq 0$.*

Proof: Assume that the question is decidable. Then we can check for every accepting state $s \in F$ whether there is an input such that leftmost cell $i = 1$ enters s at some time $t \geq 0$. If this is not true for all $s \in F$, then $L(\mathcal{M})$ is empty and $L(\mathcal{M})$ is not empty otherwise. But in [12, 16] it has been shown that it is undecidable whether or not a given real-time OCA accepts the empty language, a contradiction. □

In particular, the last lemma reveals that it is not decidable which states appear on the diagonal. So, the constructions relying on these states are not effective. However, the effectiveness can be obtained by using the whole state set instead. On the other hand, we have to pay with unnecessary additional states for the effectiveness. Thus, to some extent tight bounds in the order of magnitude are best possible.

4 Unsolvability of Decompositions

So far, we have derived tight bounds in the order of magnitude for the number of states we have to pay when applying operations on real-time OCAs. Conversely, these bounds also show the maximal number of states that can be saved when a given minimal real-time OCA is decomposed into smaller ones. For example, given a minimal n-state real-time OCA that is equivalently to be represented by the union of two smaller real-time OCAs, we know that the product of the sizes of the smaller devices is at least n. Therefore, at least $2\sqrt{n}$ states are necessary for the decomposition into two smaller devices. From this descriptional complexity point of view, natural problems concern the question of whether such decompositions can algorithmically be solved. Given a k-ary operation under which the family of real-time OCA languages is closed, does there exist an algorithm that decomposes any given real-time OCA into k smaller ones if such a decomposition exists? We refer to such problems as *operation decomposition problems*. It turns out that such algorithms cannot exist for the operations in question. The proofs are reductions of undecidability problems for real-time OCAs. In [12, 16] it has been shown that it is neither decidable whether a given real-time OCA accepts no input (emptiness) nor whether it accepts all inputs (universality).

Theorem 16 ([12, 16]) *The emptiness and universality problems for real-time OCAs are undecidable.*

Theorem 17 *The union decomposition problem for real-time OCAs is algorithmically unsolvable.*

Proof: In contrast to the assertion, we assume there is an algorithm that solves the union decomposition. We obtain a contradiction by showing that in this case the emptiness for real-time OCAs is decidable. Clearly, any OCA has at least as many states as input symbols. Moreover, there is an OCA accepting the empty language which has exactly as many states, where none of them is accepting.

In order to decide whether a given real-time OCA accepts no input, we proceed as follows. First we inspect the set of accepting states. If it is empty, the answer is *yes*. If it contains at least one input symbol, the answer is *no*. Otherwise we apply the union decomposition algorithm. If as a result the algorithm reports that there is no decomposition, the answer is *no*. If the algorithm results in two smaller OCAs, we recursively apply the decision process to these devices. Now the answer is *yes* if and only if both smaller OCAs accept the empty language.

Why does this procedure give the correct answer? This is evident for the cases where the set of accepting states is empty or contains at least one input symbol. If otherwise the union decomposition algorithm is applied, we know that there is at least one accepting non-input state. So, if the OCA accepts the empty language, there is always a possible decomposition into two smaller OCAs having only input states which are all non-accepting. □

The same result for the intersection decomposition problem follows dual to the proof of the union decomposition problem. Now, a reduction of the undecidability of universality is used. Note that there is an OCA accepting all inputs which has exactly as many states as input symbols all of which are accepting.

Theorem 18 *The intersection decomposition problem for real-time OCAs is algorithmically unsolvable.*

The next results concern the unary language operations complementation and reversal.

Theorem 19 *The complementation decomposition problem for real-time OCAs is algorithmically unsolvable.*

Proof: Assume in contrast to the assertion that there is an algorithm that solves the complementation decomposition. Given a real-time OCA $\mathcal{M}$, we apply the algorithm successively until it reports that there is no further decomposition, whereby the number of applications is counted. Then we inspect the result and determine whether it has as many states as input symbols and, if so, whether these are all accepting or all non-accepting. So, we can decide for the result whether it accepts the empty language or all inputs or another language. The result is equivalent to $\mathcal{M}$ if the number of applications is even. Therefore, in this case we know whether $\mathcal{M}$ accepts the empty language or all inputs or another language. If, on the other hand, the number of applications is odd, we know whether the complement of $\mathcal{M}$ accepts the empty language or all inputs or another language. So, we can decide emptiness and universality of $\mathcal{M}$, a contradiction. □

Theorem 20 *The reversal decomposition problem for real-time OCAs is algorithmically unsolvable.*

Proof: As in the proof of Theorem 19, given a real-time OCA $\mathcal{M}$ we apply the algorithm successively until it reports that there is no further decomposition. Then we inspect the result and decide whether it accepts the empty language or all inputs or another language. Since the reversal of the empty language is the empty language and the same for the language of all words, we can decide emptiness and universality of $\mathcal{M}$, a contradiction. □

5 Real-time OCAs With a Fixed Number of Cells

Since for real-time OCAs almost all classical decidability questions are undecidable [16] and not even semidecidable [12], real-time OCAs are on the one hand a powerful parallel model, but on the other hand very unwieldy from a practical perspective. It would be interesting to know which resources of real-time OCAs have to be restricted in order to obtain decidable questions. In [11] real-time OCAs with sparse communication have been investigated, but still a very small amount of information communicated in one time step suffices to yield undecidability of the above questions. Other resources to be bounded are

classically time and space. Obviously, real time is the minimum time needed for useful computations. Concerning space constraints, logarithmic or sublogarithmic space bounds have been investigated for Turing machines [17] and real-time iterative arrays, which differ from real-time OCAs by a sequential processing of the input. It has been shown for the latter model [15], that logarithmic space still leads to undecidability whereas sublogarithmic space reduces the computational capacity of the model to the regular languages. For real-time OCAs it is not clear yet how logarithmic or, in general, sublinear space bounds should by defined properly. One problem to overcome is that the restricted model should be not more powerful than the unrestricted model. Consider an intuitively defined real-time OCA on unary input which possesses a logarithmic number of cells depending on the length of the input. Then it would be possible to accept the non-regular language $\{ a^{2^n} \mid n \geq 1 \}$ by implementing a binary counter in the provided cells. Since the latter language cannot be accepted by any real-time OCA, we obtain a stronger model.

Thus, it might be useful to consider in a first step real-time OCAs with a fixed number of cells. This model has been introduced and investigated in [13] with regard to descriptional complexity aspects. Since the computational capacity of the model is equivalent to the regular languages, all above-mentioned decidability questions become decidable and it is particularly interesting to compare this parallel model for the regular languages with the classical model of deterministic finite automata (DFAs) from a descriptional complexity point of view. Here, we will complement the results shown in [13] by investigating the state complexity of the Boolean operations and reversal. Furthermore, the computational complexity of the decidable problems turns out not to be more complicated than that for deterministic finite automata.

A k cells one-way cellular automaton works similar to the unrestricted model, but the input is processed as follows. At the beginning all k cells are in the quiescent state. The rightmost cell is the cell receiving the input. At every time step one input symbol is processed. All other cells behave as usual. The input is accepted, if at some time step the leftmost cell enters an accepting state. Since the minimal time to read the input and to send all information from the rightmost cell to the leftmost cell is the length of the input plus k, we provide a special end-of-input symbol $\triangledown$ to the rightmost cell after reading the input.

Definition 21 *A k cells one-way cellular automaton (kC-OCA) is a tuple $\mathcal{M} = \langle S, F, A, s_0, \triangledown, k, \delta_r, \delta \rangle$ where S is the finite, nonempty set of* cell states, *$F \subseteq S$ is the set of* accepting states, *A is the nonempty set of* input symbols, *$s_0 \in S$ is the* quiescent state, *$\triangledown \notin S \cup A$ is the* end-of-input symbol, *k is the* number of cells, *and $\delta_r : S \times (A \cup \{\triangledown\}) \to S$ is the* local transition function for the rightmost cell, *satisfying $\delta_r(s_0, \triangledown) = s_0$, and $\delta : S \times S \to S$ is the* local transition function for the other cells, *satisfying $\delta(s_0, s_0) = s_0$.*

A configuration of a kC-OCA at some time step $t \geq 0$ is a pair (c_t, w_t), where $w_t \in A^*$ denotes the remaining input and c_t is a description of the k cell states, formally a mapping $c_t : \{1, 2, \ldots, k\} \to S$. For an input $w = a_1 a_2 \cdots a_n \in A^*$ the initial configuration at time 0 is defined by $c_0(i) = s_0$, $1 \leq i \leq k$ and $w_0 = w$. Successor configurations are computed according to the global transition function Δ. Let (c_t, w_t), $t \geq 0$, be a configuration, then its successor configuration is defined as follows:

$$(c_{t+1}, w_{t+1}) = \Delta(c_t, w_t) \iff \begin{cases} c_{t+1}(i) = \delta(c_t(i), c_t(i+1)), i \in \{1, 2, \ldots, k-1\} \\ c_{t+1}(k) = \delta_r(c_t(k), a) \end{cases}$$

where $a = \triangledown$ and $w_{t+1} = \lambda$, if $w_t = \lambda$, and $a = a_1$ and $w_{t+1} = a_2 a_3 \cdots a_n$, if $w_t = a_1 a_2 \cdots a_n$. Thus, Δ is induced by δ_r and δ.

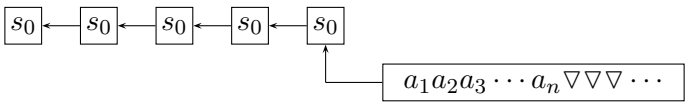

Fig. 3: Initial configuration of a 5 cells one-way cellular automaton (5C-OCA).

An input string w is accepted by a kC-OCA if at some time step during its computation the leftmost cell enters an accepting state. Real-time for kC-OCAs is defined as $|w| + k$ time steps.

Now, we investigate the state complexities of the Boolean operations and reversal for kC-OCAs and we start with two lemmas which will be useful to show lower bounds.

For all integers $k \geq 2$ and $\ell \geq 2$ let

$$L_{\ell,k} = \{\, a^i \mid i \equiv 0 \bmod \ell^k \,\}.$$

Lemma 22 *Let $k \geq 2$ and $\ell \geq 2$ be integers. Then $\ell + 2$ states are sufficient for a real-time kC-OCA to accept $L_{\ell,k}$.*

Proof: To accept the language $L_{\ell,k}$ one has to set up an ℓ-ary counter in the k cells and to check, when the whole input has been read, whether the leftmost cell has generated a carry-over in the last but one time step. Thus, we need $\ell + 1$ states to realize the ℓ-ary counter and one additional accepting state for the final check. Altogether, $\ell + 2$ states are sufficient to accept $L_{\ell,k}$. □

Lemma 23 *Let $k \geq 2$ and $\ell \geq 2$ be integers. Then at least ℓ states are necessary for a real-time kC-OCA to accept $L_{\ell,k}$.*

Proof: Let $\mathcal{M}$ be a kC-OCA accepting $L_{\ell,k}$ with s states. We first show that $\mathcal{M}$ has to distinguish at least ℓ^k configurations. By way of contradiction, we assume that there are two different inputs a^n and a^m with $0 \leq n < m \leq \ell^k - 1$ leading to the same configuration c. From c we obtain on further input $a^{\ell^k - n}$ a configuration c'. Since $a^{n+\ell^k-n} = a^{\ell^k} \in L_{\ell,k}$, we have that $c'(1)$ is an accepting state. Then, $a^{m+\ell^k-n}$ belongs to $L_{\ell,k}$ as well. On the other hand, we can derive $0 < m - n < \ell^k$ which implies that $a^{m+\ell^k-n} \notin L_{\ell,k}$. This is a contradiction.

Hence, $\mathcal{M}$ must be able to represent at least ℓ^k different configurations and we obtain that $s^k \geq \ell^k$. Thus, $s \geq \ell$. □

5.1 Intersection and Union

The constructions for real-time kC-OCAs accepting the intersection or union of languages accepted by two given real-time kC-OCAs are very similar to the constructions for real-time OCAs given in Theorem 6 and Theorem 7. The constructions are again based on the two-track technique. Additionally, for intersection one has to keep track whether some register has already passed through an accepting state. Altogether, both constructions lead to the same bounds and we omit the details here.

Theorem 24 *Let $k \geq 2$ and $m, n \geq 1$ be integers, $\mathcal{M}_1$ be an m-state real-time kC-OCA with r_1 non-accepting states, and $\mathcal{M}_2$ be an n-state real-time kC-OCA with r_2 non-accepting states. Then $m \cdot n + r_1 \cdot n + m \cdot r_2 + r_1 \cdot r_2 \in O(m \cdot n)$ states are sufficient for a real-time kC-OCA to accept $L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$.*

Theorem 25 *Let $k \geq 2$ and $m, n \geq 1$ be integers, $\mathcal{M}_1$ be an m-state real-time kC-OCA and $\mathcal{M}_2$ be an n-state real-time kC-OCA. Then $m \cdot n \in O(m \cdot n)$ states are sufficient for a real-time kC-OCA to accept $L(\mathcal{M}_1) \cup L(\mathcal{M}_2)$.*

Next, we will obtain that both upper bounds are tight in order of magnitude by showing the following lower bounds.

Theorem 26 *Let $k \geq 2$ be an integer and let $m, n \geq 4$ be integers such that m and n are relatively prime. Then at least $(m-2)(n-2) \in \Omega(m \cdot n)$ states are necessary in the worst case for a real-time kC-OCA to accept the intersection of an m-state real-time kC-OCA and an n-state real-time kC-OCA language.*

Proof: Let $m, n \geq 4$ be two integers which are relatively prime. We consider the languages $L_{m,k}$ and $L_{n,k}$ and obtain that $L_{m,k} \cap L_{n,k} = L_{mn,k}$. Due to Lemma 22 $L_{m,k}$ and $L_{n,k}$ can be accepted with $m+2$ and $n+2$ states, respectively. Owing to Lemma 23 we know that every real-time kC-OCA accepting the intersection $L_{mn,k}$ needs at least mn states. □

Theorem 27 *Let $k \geq 2$ and $m, n \geq 4$ be integers such that m and n are relatively prime. Then at least $(m-2)(n-2) \in \Omega(m \cdot n)$ states are necessary in the worst case for a real-time kC-OCA to accept the union of an m-state real-time kC-OCA and an n-state real-time kC-OCA language.*

Proof: Let $m, n \geq 4$ be two integers which are relatively prime. We consider the union of the languages $L_{m,k}$ and $L_{n,k}$. Due to Lemma 22 $L_{m,k}$ and $L_{n,k}$ can be accepted with $m+2$ and $n+2$ states, respectively.

Let $\mathcal{M}$ be a kC-OCA accepting $L_{n,k} \cup L_{m,k}$ with s states. It remains for us to show that $s \geq mn$. To this end, we prove that $\mathcal{M}$ has to distinguish at least $(mn)^k$ configurations. Then, we obtain that $s^k \geq (mn)^k$ which implies that $s \geq mn$.

By way of contradiction, we assume that there are two different inputs a^p and a^q with $0 \leq p < q \leq (mn)^k - 1$ leading to the same configuration c. Let $p' = p \bmod n^k$, $q' = q \bmod n^k$, $p'' = p \bmod m^k$, and $q'' = q \bmod m^k$. At first, we can show that $p' \neq q'$ or $p'' \neq q''$. Otherwise, we would have that $p' = q'$ and $p'' = q''$. Then, $p = t \cdot n^k + p'$ and $q = t' \cdot n^k + p'$ which implies that $q - p$ is a multiple of n^k. Analogously, we obtain that $q - p$ is a multiple of m^k. Thus, $q - p$ is a multiple of $(mn)^k$. This is a contradiction, since $0 < q - p < (mn)^k$.

From now on we assume without loss of generality that $p' \neq q'$ and $p' < q'$. Otherwise, we consider $p'' \neq q''$ or interchange the roles of p' and q' or p'' and q'', respectively. Let $0 < l \leq n^k$ be the unique integer such that $p' + l = n^k$. Then, $p + l \equiv 0 \bmod n^k$. Furthermore, we have that $n^k < q' + l < 2n^k$ which implies that $q + l \not\equiv 0 \bmod n^k$. Finally, we consider $q + l$ and distinguish two cases. If $q + l \not\equiv 0 \bmod m^k$, then we know that $a^{p+l} \in L_{n,k}$ and $a^{q+l} \notin L_{n,k} \cup L_{m,k}$. From configuration c we obtain on further input a^l a configuration c'. Since $a^{p+l} \in L_{n,k}$, we have that $c'(1)$ is an accepting state. Then, a^{q+l} belongs to $L_{n,k} \cup L_{m,k}$ as well which is a contradiction.

If $q + l \equiv 0 \bmod m^k$, then we know that $q'' + l = t''m^k$. Now, we consider $q'' + l + n^k$ and obtain that $q'' + l + n^k = t''m^k + n^k$ is not a multiple of m^k. Then, $q + l + n^k \not\equiv 0 \bmod m^k$. Moreover,

$q' + l + n^k$ is not a multiple of n^k, since $q' + l$ is not. Thus, $q + l + n^k \not\equiv 0 \bmod n^k$. Finally, $p' + l + n^k$ is a multiple of n^k, since $p' + l$ is. So, $p + l + n^k \equiv 0 \bmod n^k$ and we know that $a^{p+l+n^k} \in L_{n,k}$ and $a^{q+l+n^k} \notin L_{n,k} \cup L_{m,k}$. From configuration c we obtain on further input a^{l+n^k} a configuration c''. Since $a^{p+l+n^k} \in L_{n,k}$, we have that $c''(1)$ is an accepting state. Then, a^{q+l+n^k} belongs to $L_{n,k} \cup L_{m,k}$ as well which is a contradiction and concludes the proof. □

5.2 Complementation

The construction of a real-time kC-OCA accepting the complement of the language accepted by a given real-time kC-OCA is slightly different to the construction for real-time OCAs given in the proof of Theorem 10. However, the blow-up concerning the number of states is similar and we can show that the upper bound is tight in order of magnitude as well.

Theorem 28 *Let $k \geq 2$ and $n \geq 1$ be integers and $\mathcal{M}$ be an n-state real-time kC-OCA with r non-accepting states. Then $2(n + r) \in O(n)$ states are sufficient for a real-time kC-OCA to accept $\overline{L(\mathcal{M})}$.*

Proof: Let S and F denote the set of states and accepting states of $\mathcal{M}$, respectively. At first, we have to modify $\mathcal{M}$ such that $\mathcal{M}$ only accepts when the whole input has been processed. To this end, the rightmost cell emits a signal when it reads the end-of-input symbol for the first time. This signal moves to the left and remembers the state of the cell passed through, respectively. Finally, the signal will arrive at the leftmost cell exactly when the whole input has been processed and all information has been sent to the leftmost cell. At this time step we want to make the final decision whether to accept or to reject the input. So, the leftmost cell has to remember whether it has entered an accepting state at some time before. This can realized the same way as before by introducing a copy of the non-accepting states S' of the state set $S \setminus F$ of $\mathcal{M}$ and modifying the local transition function suitably. Then, the modified automaton $\hat{\mathcal{M}}$ accepts, if and only if the leftmost cell is in some state of $S' \cup F$ when the signal arrives. In order to accept the complement of $L(\mathcal{M}) = L(\hat{\mathcal{M}})$, it suffices to let the automaton accept, if and only if the leftmost cell is in some state of $S \setminus F$ when the signal arrives.

Disregarding the realization of the signal, the number of states needed is $n + r$. The implementation of the signal may at most double this number and we obtain $2(n + r)$ states as an upper bound. □

Theorem 29 *Let $k \geq 2$ and $n \geq 3$ be integers. Then at least $n - 1 \in \Omega(n)$ states are necessary in the worst case for a real-time kC-OCA to accept the complement of an n-state real-time kC-OCA language.*

Proof: We consider the witness languages

$$L'_{n,k} = \{\, a^i \mid i \geq n^k \,\}.$$

First, we construct an $(n+1)$-state real-time kC-OCA accepting $L'_{n,k}$. To this end, one has to set up an n-ary counter and to define the state denoting a carry-over as the only accepting state (see also Example 2.1 in [13]).

On the other hand, every kC-OCA accepting

$$\overline{L'_{n,k}} = \{\, a^i \mid i < n^k \,\}$$

needs at least n states, since at least n^k configurations have to be distinguished. □

5.3 Reversal

The construction of a real-time kC-OCA accepting the reversal of the language accepted by a given real-time kC-OCA is completely different to the construction for real-time OCAs given in the proof of Theorem 12 where a quadratic upper bound is shown. Here, we will obtain an exponential upper bound which is almost tight in order of magnitude.

Theorem 30 *Let $k \geq 2$ and $n \geq 1$ be integers and $\mathcal{M}$ be an n-state real-time kC-OCA. Then at most $2^{n^k - n^{k-1}+1} + 1 \in O(2^{n^k})$ states are sufficient for a real-time kC-OCA to accept $L(\mathcal{M})^R$.*

Proof: We present the intuitive construction. At first, we convert $\mathcal{M}$ to an equivalent DFA $\mathcal{N}$ having at most $n^k - n^{k-1} + 1$ states according to the construction given in [13]. Then, $\mathcal{N}$ is converted to a DFA $\mathcal{N}^R$ accepting the reversal of $L(\mathcal{N})$. By using the standard construction, $\mathcal{N}^R$ has at most $2^{n^k - n^{k-1}+1}$ states. Finally, $\mathcal{N}^R$ is converted to an equivalent kC-OCA $\mathcal{M}^R$. Due to the construction given in [13] we need one additional state which gives the upper bound claimed. □

The above construction arises the question whether it is in fact the best possible. In particular, the construction does not make use of the parallelism of kC-OCAs. The next lemma provides a lower bound which roughly says that the construction cannot be improved or parallelized essentially with regard to kC-OCAs. This shows that reversal is a very expensive operation for kC-OCAs whereas only a quadratic blow-up occurs for real-time OCAs.

Theorem 31 *Let $k \geq 2$ and $n \geq 3$ be integers such that $n \geq k$. Then at least $\Omega(2^{(n-1)^{k-1}})$ states are necessary in the worst case for a real-time kC-OCA to accept the reversal of an n-state real-time kC-OCA language.*

Proof: We consider the witness languages

$$L''_{n,k} = \{\, a^{n^k} \{a,b\}^i \mid i \geq 0 \,\}.$$

To accept the language $L''_{n,k}$, we can use the same construction as in the proof of Theorem 29. Thus, $L''_{n,k}$ can be accepted with $n+1$ states.

Now, let $\mathcal{M}$ be a kC-OCA accepting

$$L''^R_{n,k} = \{\, \{a,b\}^i a^{n^k} \mid i \geq 0 \,\}$$

with s states. We first show that $\mathcal{M}$ has to distinguish at least 2^{n^k} configurations. By way of contradiction, we assume that there are two different inputs $u, v \in \{a,b\}^{n^k}$ leading to the same configuration c. Since u and v are different, we obtain without loss of generality that $u = xaa^t$ and $v = yba^t$ with $0 \leq t \leq n^k - 1$. From configuration c we obtain on further input a^{n^k-t-1} a configuration c'. Since $ua^{n^k-t-1} \in L''^R_{n,k}$, we have that $c'(1)$ is an accepting state. Then, va^{n^k-t-1} belongs to $L''^R_{n,k}$ as well. This is a contradiction, since $va^{n^k-t-1} \notin L''^R_{n,k}$.

Since $\mathcal{M}$ must be able to represent at least 2^{n^k} different configurations, we obtain that $s^k \geq 2^{n^k}$. Thus, $s \geq 2^{\frac{n^k}{k}} \geq 2^{\frac{n^k}{n}} = 2^{n^{k-1}}$ since $n \geq k$. □

We may summarize the state complexity of the operations studied as follows. The state complexity of intersection and union for kC-OCAs of size m and n, respectively, is in $\Theta(mn)$. The state complexity of complementation for a kC-OCA of size n is in $\Theta(n)$. The upper bound of the state complexity of reversal for a kC-OCA of size n is in $O(2^{n^k})$ and the lower bound is in $\Omega(2^{(n-1)^{k-1}})$.

5.4 Computational Complexity

Finally, we discuss the computational complexity of typical decidability questions. For real-time OCAs these questions are known to be undecidable. Here, we show that the questions are decidable for kC-OCAs with $k \geq 2$ and, moreover, are NLOGSPACE-complete. Thus, the questions for kC-OCAs have the same computational complexity as for deterministic finite automata.

Theorem 32 *Let $k \geq 2$ be an integer. Then for kC-OCAs the problems of testing emptiness, universality, inclusion, and equivalence are NLOGSPACE-complete.*

Proof: First, we show that the problem of non-emptiness belongs to NLOGSPACE. Since NLOGSPACE is closed under complementation, emptiness belongs to NLOGSPACE as well. We describe a two-way nondeterministic Turing machine $\mathcal{M}$ which receives an encoding of some kC-OCA $\mathcal{A}$ on its read-only input tape and produces on its write-only output tape an answer *yes* or *no* while the space used on its working tape is bounded by $O(\log |cod(\mathcal{A})|)$. Then, the work space is bounded by $O(\log n)$ as well where n denotes the maximum of the number of states in $\mathcal{A}$ and the size of the input alphabet of $\mathcal{A}$, since both parameters are part of the encoding of $\mathcal{A}$ on the input tape of $\mathcal{M}$. It is shown in [13] that $\mathcal{A}$ can be converted to an equivalent DFA $\mathcal{A}'$ having at most $n^k - n^{k-1} + 1$ states. It has been shown in [7] by using the pumping lemma for regular languages that $L(\mathcal{A}')$ is not empty if and only if $L(\mathcal{A}')$ contains a word of length at most n^k. Thus, the idea for the Turing machine $\mathcal{M}$ is to guess a word of length at most n^k and to check whether it is accepted by $\mathcal{A}$. We implement on $\mathcal{M}$'s working tape a binary counter C which counts up to n^k. With the usual construction this needs at most $O(\log n^k) = O(k \log n) = O(\log n)$ tape cells. Additionally, we have to keep track of the current states of the k cells of $\mathcal{A}$. Clearly, the state of each cell can be represented by $O(\log n)$ tape cells. Altogether, a configuration of $\mathcal{A}$ can be represented by $O(\log n)$ tape cells. Now, $\mathcal{M}$ guesses one input symbol a, $\mathcal{M}$ increases the counter C, and updates all cells of $\mathcal{A}$ according to the transition function of $\mathcal{A}$ encoded on the input tape. This behavior is iterated until either the simulated leftmost cell of $\mathcal{A}$ enters an accepting state of $\mathcal{A}$ or the counter C has been counted up to n^k. In both cases $\mathcal{M}$ halts and outputs *yes* in the first case and outputs *no* in the latter. Altogether, $\mathcal{M}$ decides the non-emptiness of $\mathcal{A}$ and uses at most a logarithmic number of tape cells with regard to the length of the input.

For the problem of non-universality of a given kC-OCA $\mathcal{A}$ we test the non-emptiness of a kC-OCA $\mathcal{A}'$ accepting the complement of $L(\mathcal{A})$. The only difference to the above construction is that we have to simulate a computation in $\mathcal{A}'$ instead of $\mathcal{A}$. To this end, we consider the construction for the complement given in Theorem 28. Having programmed this modification of the transition functions of a kC-OCA in the finite control of the Turing machine $\mathcal{M}$ suitably, we can simulate a transition of $\mathcal{A}'$ when reading and translating a transition of $\mathcal{A}$ from the input tape. Additionally, we have to observe that the number of states of $\mathcal{A}'$ increases only by a linear factor of 4. Thus, it suffices for the counter C to count up to $(4n)^k$. Altogether, we obtain that non-universality is in NLOGSPACE. Due to the closure under complementation, universality is in NLOGSPACE as well.

The constructions for testing inclusion and equivalence are similar. For two kC-OCAs $\mathcal{A}_1$ and $\mathcal{A}_2$ we have that $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ if and only if $L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)}$ is empty. Due to the construction given in Theorem 24, we can reduce the question of inclusion to the question of testing the emptiness of a kC-OCA whose size is linearly bounded with regard to the size of $\mathcal{A}_1$ and $\mathcal{A}_2$. By similar observations as for non-universality, we obtain that the problem of inclusion is in NLOGSPACE. Finally, two kC-OCAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if and only if both $L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)}$ and $\overline{L(\mathcal{A}_1)} \cap L(\mathcal{A}_2)$ are empty. Thus, equivalence is in NLOGSPACE as well.

The hardness results follow directly from the hardness results for DFAs (see, e.g., the summary in [18]), since any DFA can be effectively converted to an equivalent kC-OCA [13] which simulates the given DFA in the rightmost cell and sends an additional accepting state to the leftmost cell when the end-of-input symbol is read and the input is accepted by the DFA. Obviously, this construction can be done in deterministic logarithmic space. □

References

[1] Brzozowski, J.: Quotient complexity of regular languages. In: Descriptional Complexity of Formal Systems (DCFS 2009), Otto-von-Guericke-Universität Magdeburg (2009) 25–42

[2] Brzozowski, J., Jirásková, G., Li, B.: Quotient complexity of ideal languages. In: Latin 2010: Theoretical Informatics. LNCS, Springer (2010) to appear

[3] Brzozowski, J., Jirásková, G., Zou, C.: Quotient complexity of closed languages. In: Computer Science Symposium in Russia (CSR 2010). LNCS, Springer (2010) to appear

[4] Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. UCS **8** (2002) 193–234

[5] Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Int. J. Found. Comput. Sci. **14** (2003) 1087–1102

[6] Holzer, M., Kutrib, M.: Descriptional complexity – an introductory survey. In: Scientific Applications of Language Methods. Imperial College Press (2010) to appear

[7] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Massachusetts (1979)

[8] Jirásková, G., Okhotin, A.: On the state complexity of operations on two-way finite automata. In: Developments in Language Theory (DLT 2008). Volume 5257 of LNCS, Springer (2008) 443–454

[9] Kutrib, M.: Cellular automata – a computational point of view. In: New Developments in Formal Languages and Applications. Springer (2008) 183–227

[10] Kutrib, M.: Cellular automata and language theory. In: Encyclopedia of Complexity and System Science. Springer (2009) 800–823

[11] Kutrib, M., Malcher, A.: Cellular automata with sparse communication. In: Implementation and Application of Automata (CIAA 2009). Volume 5642 of LNCS, Springer (2009) 34–43

[12] Malcher, A.: Descriptional complexity of cellular automata and decidability questions. J. Autom., Lang. Comb. **7** (2002) 549–560

[13] Malcher, A.: On one-way cellular automata with a fixed number of cells. Fund. Inform. **58** (2003) 355–368

[14] Malcher, A.: On the descriptional complexity of iterative arrays. IEICE Trans. Inf. Syst. **E87-D** (2004) 721–725

[15] Malcher, A., Mereghetti, C., Palano, B.: Sublinearly space bounded iterative arrays. In: Automata and Formal Languages (AFL 2008), Hungarian Academy of Sciences (2008) 292–301

[16] Seidel, S.R.: Language recognition and the synchronization of cellular automata. Technical Report 79-02, Department of Computer Science, University of Iowa (1979)

[17] Szepietowski, A.: Turing Machines with Sublogarithmic Space. Volume 843 of LNCS, Springer (1994)

[18] Yu, S.: Regular languages. In: Handbook of Formal Languages. Volume 1. Springer (1997) 41–110

[19] Yu, S.: State complexity of regular languages. J. Autom., Lang. Comb. **6** (2001) 221–234

[20] Yu, S.: State complexity of finite and infinite regular languages. Bull. EATCS **76** (2002) 142–152