Iterative Arrays with Self-verifying Communication Cell

Martin Kutrib, Thomas Worsch

Iterative Arrays with Self-Verifying Communication Cell

Martin Kutrib[1] and Thomas Worsch[2]

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

[2] Karlsruhe Institute of Technology
worsch@kit.edu

Abstract. We study the computational capacity of self-verifying iterative arrays (SVIA). A self-verifying device is a nondeterministic device whose nondeterminism is symmetric in the following sense. Each computation path can give one of the answers *yes*, *no*, or *do not know*. For every input word, at least one computation path must give either the answer *yes* or *no*, and the answers given must not be contradictory. It turns out that, for any time-computable time complexity, the family of languages accepted by SVIAs is a characterization of the so-called complementation kernel of nondeterministic iterative array languages, that is, languages accepted by such devices whose complementation is also accepted by such devices. SVIAs can be sped-up by any constant multiplicative factor as long as the result does not fall below realtime. We show that even realtime SVIA are as powerful as lineartime self-verifying cellular automata and vice versa. So they are strictly more powerful than the deterministic devices. Closure properties and various decidability problems are considered.

1 Introduction

One of the central questions in complexity and language theory asks for the power of nondeterminism in bounded-resource computations. In order to gain a better understanding of nondeterminism it has been viewed as an additional limited resource at the disposal of time or space bounded computations. The concept of so-called *self-verification* at least dates back to the paper [5]. It applies to automata for decision problems and makes use of stronger notions of acceptance and rejection of inputs.

A self-verifying device is a nondeterministic device whose nondeterminism is symmetric in the following sense. Each computation path can give one of the answers *yes*, *no*, or *unknown*. For every input word, at least one computation path must give either the answer *yes* or *no*, and the answers given must not be contradictory. So, if a computation path gives the answer *yes* or *no*, in both cases the answer is definitely correct. This justifies the notion *self-verifying* and is in contrast to the general case, where an answer different from *yes* does not

allow to conclude whether or not the input belongs to the language. Here we study the computational capacity of self-verifying iterative arrays (SVIA).

Self-verifying finite automata have been introduced and studied in [5] and others mainly in connection with randomized Las Vegas computations. Descriptional complexity issues for self-verifying finite automata have been studied in [8]. The computational and descriptional complexity of self-verifying pushdown automata has been studied in [7]. Self-verifying cellular automata have been introduced in [12]. Some of the results in the present paper look very similar, but they require different proofs.

The paper is organized as follows. In Section 2 we present the basic notation and the definitions of self-verifying iterative arrays as well as an introductory example. In general, the symmetric conditions for acceptance/rejection of self-verifying devices imply immediately the effective closures of the language families accepted under complementation. In Section 3 this observation is turned in a characterization. Moreover, the strong speed-up by a multiplicative constant is derived for any time-computable time complexity. In Section 4 we explore the computational capacity of realtime SVIAs. In particular, its is shown that even realtime SVIAs are as powerful as lineartime self-verifying cellular automata and vice versa. So they are strictly more powerful than deterministic iterative arrays. Closure properties of the family of languages accepted by realtime SVIAs are studied in Section 5. The family is closed under the Boolean operations, reversal, concatenation, and inverse homomorphisms, while it is not closed under arbitrary homomorphisms. Finally, decidability problems are considered in Section 6. In particular, by a reduction of the emptiness problem it is shown that the property of being self-verifying is non-semidecidable.

2 Preliminaries and Definitions

We denote the non-negative integers by $\mathbb{N}$. Let Σ denote a finite set of letters. Then we write Σ^* for the *set of all finite words* (strings) consisting of letters from Σ. The *empty word* is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. For the *reversal of a word* w we write w^R and $|w|$ denotes its *length*. A subset of Σ^* is called a *language* over Σ. In general, we use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*.

A one-dimensional iterative array is a linear, semi-infinite array of finite state machines (sometimes called cells) that are identical except for the leftmost one. All but the leftmost cells are connected to their both nearest neighbors, respectively (see Figure 1). For convenience we identify the cells by their coordinates, that is, by non-negative integers. The leftmost cell is distinguished. This so-called communication cell is connected to its right neighbor and, additionally, to the input supply which feeds the input sequentially. We assume that once the whole input is consumed an end-of-input symbol is supplied permanently. At the outset of a computation all cells are in the so-called quiescent state. The cells work synchronously at discrete time steps. Here we assume that the communication cell is a nondeterministic finite automaton while all the other cells

are deterministic ones (cf. [1]). Although this is a very restricted case, for easier writing we call such devices nondeterministic.

Formally, a *nondeterministic iterative array* (NIA, for short) is a system $M = \langle S, \Sigma, F_+, s_0, \lhd, \delta_{nd}, \delta_d \rangle$, where S is the finite, nonempty set of *cell states*, Σ is the finite, nonempty set of *input symbols*, $F_+ \subseteq S$ is the set of *accepting states*, $s_0 \in S$ is the *quiescent state*, $\lhd \notin \Sigma$ is the *end-of-input symbol*, $\delta_{nd} \colon (\Sigma \cup \{\lhd\}) \times S \times S \to (2^S \setminus \emptyset)$ is the *nondeterministic local transition function for the communication cell*, $\delta_d \colon S \times S \times S \to S$ is the *deterministic local transition function for non-communication cells* satisfying $\delta_d(s_0, s_0, s_0) = s_0$.

A configuration of M at time $t \geq 0$ is a pair (w_t, c_t), where $w_t \in \Sigma^*$ is the remaining input sequence and $c_t \colon \mathbb{N} \to S$ is a mapping that maps the single cells to their current states. The initial configuration (w_0, c_0) is defined by the given input $w_0 \in \Sigma^*$ and the mapping $c_0(i) = s_0$, $i \geq 0$. Subsequent configurations are computed by the *global transition function* Δ that is induced by δ_d and δ_{nd} as follows: Let (w_t, c_t), $t \geq 0$, be a configuration. Then the set of its possible successor configurations (w_{t+1}, c_{t+1}) is defined as follows:

$$(w_{t+1}, c_{t+1}) \in \Delta((w_t, c_t)) \iff \begin{cases} c_{t+1}(0) \in \delta_{nd}(a, c_t(0), c_t(1)) \\ c_{t+1}(i) = \delta_d(c_t(i-1), c_t(i), c_t(i+1)) \end{cases}$$

for all $i \geq 1$, where $a = \lhd$, $w_{t+1} = \lambda$ if $w_t = \lambda$, and $a = a_1$, $w_{t+1} = a_2 a_3 \cdots a_n$ if $w_t = a_1 a_2 \cdots a_n$.

An input w is accepted by an NIA M if at some time step during the course of at least one computation for w the communication cell enters an accepting state. The *language accepted by* M is denoted by $L(M) = \{\, w \in \Sigma^* \mid w \text{ is accepted by } M \,\}$. Let $t \colon \mathbb{N} \to \mathbb{N}$, $t(n) \geq n + 1$ be a mapping. If for each $w \in L(M)$ there is an accepting computation with at most $t(|w|)$ time steps, then M and $L(M)$ are said to be of time complexity t.

In general, the family of all languages which are accepted by some type of device X with time complexity t is denoted by $\mathscr{L}_t(X)$. If t is the function $n + 1$, acceptance is said to be in *realtime*. Since for nontrivial computations an iterative array has to read at least one end-of-input symbol, realtime has to be defined as $(n + 1)$-time. We write $\mathscr{L}_{rt}(X)$ for realtime and $\mathscr{L}_{lt}(X)$ for lineartime.

Now we turn to *self-verifying* iterative arrays (SVIA). Basically, an SVIA is an NIA, but the definition of acceptance is different. There are now three disjoint sets of states representing answers *yes*, *no*, and *neutral*. Moreover, for every input word, at least one computation path must give either the answer *yes* or *no*, and the answers given must not be contradictory. In order to implement the three possible answers the state set is partitioned into three disjoint subsets

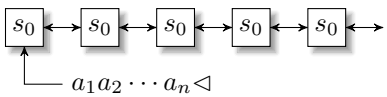

Fig. 1. Initial configuration of an iterative array.

$S = F_+ \dot\cup F_- \dot\cup F_0$, where F_+ is the set of accepting states, F_- is the set of rejecting states, and $F_0 = S \setminus (F_+ \cup F_-)$ is referred to as the set of neutral states. If $M = \langle S, \Sigma, F_+, F_-, s_0, \triangleleft, \delta_{nd}, \delta_d\rangle$ is an SVIA, for each input word $w \in \Sigma^*$ and for a corresponding computation $\bar c$ let $S_{w,\bar c}$ denote the set of states entered by the communication cell during computation $\bar c$. For the "self-verifying property" it is required that for each $w \in \Sigma^*$ and each corresponding $\bar c$, $S_{w,\bar c} \cap F_+$ is empty if and only if $S_{w,\bar c} \cap F_-$ is nonempty.

If all $w \in L(M)$ are accepted and all $w \notin L(M)$ are rejected after at most $t(|w|)$ time steps, then the self-verifying iterative array M is said to be of time complexity t.

In the sequel we will often utilize the possibility of iterative arrays to simulate the data structures pushdown stores (stacks) [2, 4], queues, and rings [9] without any loss of time. Here a ring is a queue that can write and erase at the same time. For pushdown stores the communication cell simulates the top of the store, for queues it simulates the front, and for rings the front and the end of the store.

We illustrate the definitions with an example.

Example 1. The nondeterministic context-free language $\{\, w \in \{a,b\}^* \mid w = w^R \,\}$ is accepted by the SVIA $M = \langle S, \{a,b\}, F_+, F_-, s_0, \triangleleft, \delta_{nd}, \delta_d\rangle$. The basic idea is to simulate a stack whose top is the communication cell.

We set $S = (\{s_1, s_2, s_3, s_+, s_-, s_?\} \times S_{pd}) \cup \{s_0\} \cup \hat S_{pd}$ with $F_+ = \{s_+\} \times S_{pd}$ and $F_- = \{s_-\} \times S_{pd}$. Here S_{pd} are the register contents used by the communication cell to manage the top entries of the stack, while $\hat S_{pd}$ are the non-quiescent states of all but the communication cell, that realize the stack. So, the transition function δ_d just realizes the interior of the stack and is omitted here.

The idea of the construction of δ_{nd} is summarized in the following table and described below. Since we didn't make S_{pd} explicit in detail and since the state of the right neighbor of the communicating cell is only needed for updating its part of the stack, the right neighbor state is left out in the table. The current state of the communication cell is indicated as $(s_i, y\cdots)$ meaning that at the top of the stack is a symbol $y \in \{a,b\}$. A transition to $xy\cdots$ means that x has been pushed onto, and a transition to $\cdots$ means that y has been popped from the stack.

(1)	$\delta_{nd}(\triangleleft, (s_0, \bot), \_) \ni (s_+, \bot)$	accept λ
(2)	$\delta_{nd}(x, (s_0, \bot), \_) \ni (s_1, x)$	push first symbol
(3)	$\delta_{nd}(x, (s_1, y\cdots), \_) \ni (s_1, xy\cdots)$	in s_1 continue pushing input symbols
(4)	$\delta_{nd}(x, (s_1, y\cdots), \_) \ni (s_2, y\cdots)$	switch to s_2, dropping symbol x
(5)	$\delta_{nd}(x, (s_1, y\cdots), \_) \ni (s_2, xy\cdots)$	switch to s_2, without dropping x
(6)	$\delta_{nd}(x, (s_2, x\cdots), \_) \ni (s_2, \cdots)$	continue in s_2 for matching symbols
(7)	$\delta_{nd}(\triangleleft, (s_2, \bot), \_) \ni (s_+, \bot)$	accept if everything matched
(8)	$\delta_{nd}(x, (s_2, \bar x\cdots), \_) \ni (s_3, \cdots)$	switch to s_3 if mismatch ($\bar a = b, \bar b = a$)
(9)	$\delta_{nd}(x, (s_3, y\cdots), \_) \ni (s_3, \cdots)$	drop remaining input symbols
(10)	$\delta_{nd}(\triangleleft, (s_3, \bot), \_) \ni (s_-, \bot)$	reject since there was a mismatch

Consider the point in time after the SVIA is in state s_2 for the first time. Let v denote the part of the input that has been pushed to the stack at that time and let u denote the part of the input that still has not been read. Then the input is vxu if (4) has been used to switch to s_2 and the input is vu if (5) was used.

If the communication cell switched from s_1 to s_2 "at the right time", that is $|u| = |v|$, then it will for the first time see input $\lhd$ and the empty stack $\bot$ simultaneously. Obviously, the decision to accept using (7) or to reject using (10) is the correct one.

If the communication cell switched from s_1 to s_2 "at the wrong time", that is $|u| \neq |v|$, then it cannot decide whether the input is a palindrome. This can be recognized by the SVIA since either all input symbols have been consumed but the stack is not empty or the stack is empty but not all input symbols have been consumed. In this case a correct behavior is obtained by entering state $s_?$ and rules that make the SVIA never leave it again. ■

3 Structural Properties and Speed-Up

Though we are mainly interested in fast computations, that is, realtime and lineartime computations, we allowed general time complexities in the definition of the devices (see [10] for a discussion of this general treatment of time complexity functions). However, it seems to be reasonable to consider only time complexities t that allow the communication cell to recognize the time step $t(n)$. Such functions are said to be *time-computable*. For example, the function $t(n) = n+1$ is trivially a time-computable time complexity for IAs.

Other examples are time complexities $\lfloor \frac{y}{x} \cdot n \rfloor$, for any positive integers $x < y$, polynomials $t(n) = n^k$, and exponential time complexities $t(n) = k^n$, for any integer $k \geq 2$. More details can be found in [13].

In general, the symmetric conditions for acceptance/rejection of self-verifying devices imply immediately the effective closures of the language families accepted under complementation. In order to turn this observation in a characterization, we first give evidence that self-verifying iterative arrays are in fact a generalization of deterministic iterative arrays. The proof of a corresponding result for cellular automata [12] applies here almost literally.

Lemma 2. *Any deterministic iterative array with a time-computable time complexity t can effectively be converted into an equivalent self-verifying iterative array with the same time complexity t.*

The proper inclusion $\mathscr{L}_{rt}(\text{IA}) \subset \mathscr{L}_{rt}(\text{NIA})$ is well known [1]. So, nondeterminism strengthens the computational capacity of iterative arrays. On the other hand, it is an open problem whether the family $\mathscr{L}_{rt}(\text{NIA})$ is closed under complementation. Therefore, the question whether the family $\mathscr{L}_{rt}(\text{SVIA})$ is properly included in $\mathscr{L}_{rt}(\text{NIA})$, or whether both families coincide, is of natural interest. Next we turn to relate it to the open complementation closure of $\mathscr{L}_{rt}(\text{NIA})$.

Proposition 3. *Let t be a time-computable time complexity. The family of languages $L \in \mathscr{L}_t(NIA)$, such that the complement $\overline{L}$ belongs to $\mathscr{L}_t(NIA)$ as well, coincides with the family $\mathscr{L}_t(SVIA)$.*

Proof. Given a t-time SVIA M, it is straightforward to construct an NIA that accepts the complement of $L(M)$ with the same time complexity t.

Conversely, let M_1 be an NIA accepting L and M_2 be an NIA accepting $\overline{L}$ with time complexity t. Now a t-time self-verifying iterative array M simulates M_1 and M_2 on different tracks, that is, it uses the same two channel technique of [6, 14].

Then it remains to define the set of accepting states as $F_+ = \{ (s, s') \mid s \in F_1 \}$ and the set of rejecting states as $F_- = \{ (s, s') \mid s' \in F_2 \}$, where F_1 is the set of accepting states of M_1 and F_2 is the set of accepting states of M_2. □

Proposition 3 implies that $\mathscr{L}_{rt}$(SVIA) is properly included in $\mathscr{L}_{rt}$(NIA) if and only if $\mathscr{L}_{rt}$(NIA) is not closed under complementation; otherwise both families coincide.

Next, we turn to strong speed-up results for self-verifying iterative arrays from which follows that realtime is as powerful as lineartime.

Theorem 4. *Let $k \geq 1$ be a constant and t be a time complexity. Then the families $\mathscr{L}_{k \cdot t}(SVIA)$ and $\mathscr{L}_t(SVIA)$ coincide.*

Proof. A given $(k \cdot t)$-time SVIA M is simulated by a t-time SVIA M' as follows. Basically, M' performs two tasks in parallel on different tracks.

For the first task, assume that the input is fed to the communication cell of M' in k-symbol blocks, that is, k input symbols in each step. Then each k cells of M' are grouped together into one cell. In this well-known way the iterative array M' can simulate k steps of M in one step. That is, this task of M' has time complexity t and the self-verifying property, since M has time complexity $k \cdot t$ and the self-verifying property.

The second task of M' is to make the assumption for the first task true. To this end, it simulates a ring store whose front (and end) is the communication cell. Now the communication cell starts to guess k input symbols in every step. These symbols are fed to the first task. Additionally, the communication step guesses when the end-of-input symbol appears. From that time step on no further input symbols are guessed. In order to verify that the guesses are correct, the k symbols are entered at the end of the ring store respectively. In each step, the symbol at the front of the ring is removed and compared with the actual input symbol. If both match, the guessed symbol is correct, otherwise it is not. In case of a mismatch or a wrongly guessed number of input symbols the second task remains in a neutral state. If it has guessed the input correctly, it enters a positive state.

Finally, M' accepts if and only if the second task guesses the input correctly and the first task accepts. That is, if the actual input is accepted by M. The iterative array M' rejects if and only if the second task guesses the input correctly and the first task rejects. That is, if the actual input is rejected by M. So, M' has the self-verifying property. □

Corollary 5. *The families $\mathscr{L}_{rt}(SVIA)$ and $\mathscr{L}_{lt}(SVIA)$ coincide.*

4 Computational Capacity

The first question in connection with the computational capacity of realtime SVIA is the impact of the (restricted) nondeterminism. Does it increase the capacity? More precisely, we are interested in the question whether the computing power of realtime SVIA is strictly stronger than that of realtime IA.

Example 1 shows that the mirror language is accepted by a realtime SVIA. However, by using a completely different algorithm the language is accepted by some deterministic realtime IA as well [3]. So, it cannot be used as a witness for the strictness of the inclusion $\mathscr{L}_{rt}(\text{IA}) \subset \mathscr{L}_{rt}(\text{SVIA})$. Nevertheless, the strictness follows from a more general result below and is stated in Proposition 10.

Corollary 6. *The family of languages accepted by self-verifying pushdown automata is strictly included in the family* $\mathscr{L}_{rt}(SVIA)$.

In order to discuss further comparisons we now turn to results that show the strong computational capacity of realtime SVIA.

While iterative arrays fetch their input sequentially through the communication cell, so-called cellular automata obey a parallel input mode. In a preinitial step their cells fetch an input symbol. That is, there are as many cells as input symbols. So, a two-way cellular automaton (CA) is a linear array of identical finite automata which are numbered $1, 2, \ldots, n$. Except for border cells the state transition depends on the current state of a cell itself and those of its both nearest neighbors. Border cells receive a boundary symbol # on their free input lines. An input w is accepted by a cellular automaton if at some time step during some computation the leftmost cell enters an accepting state. For cellular automata, realtime is defined to be $t(n) = n$. Cellular automata whose first state transitions are nondeterministic, whose further transitions are deterministic, and that have the self-verifying property (SVCA) are studied in [12].

Lemma 7. *The family* $\mathscr{L}_{lt}(SVIA)$ *is included in* $\mathscr{L}_{rt}(SVCA)$.

Lemma 7 gives an upper bound for languages accepted by lineartime SVIA. It shows that a sequential input mode and one nondeterministic cell can be traded for parallel input mode and all cells nondeterministic once only. In fact, the upper bound is sharp, that is, the converse is also true.

Lemma 8. *The family* $\mathscr{L}_{lt}(SVCA)$ *is included in* $\mathscr{L}_{rt}(SVIA)$.

Proof. The possibility to speed-up SVCAs by a constant factor is shown in [12]. That is, $\mathscr{L}_{lt}(\text{SVCA}) = \mathscr{L}_{rt}(\text{SVCA})$. So, given some realtime SVCA M with state set S and set of input symbols Σ, we construct an equivalent realtime SVIA M' as follows. Basically, M' works in three phases. First it guesses and generates a configuration that represents the fourfold packed initial configuration of M. Then this packed part is synchronized. Finally, the synchronized cells simulate M whereby four steps are simulated in one step. The phases are depicted in Figure 2. In parallel, the guesses are verified.

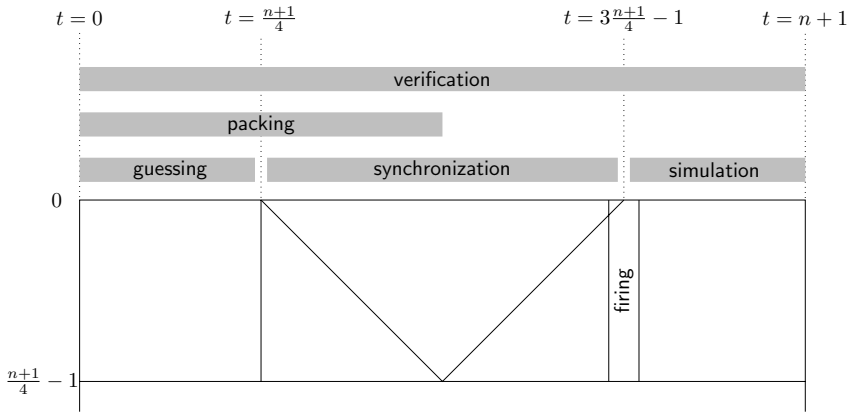

Fig. 2. Simulation phases.

Phase 1: Let n denote the length of the input. In the following we assume that $n+1$ is a multiple of four. The generalization of the simulation to the other cases is a straightforward adaption.

So, during the first $\frac{n+1}{4}$ time steps the communication cell of M' guesses four input symbols in every step. Additionally, the communication cell guesses a mapping $(\Sigma \cup \{\#\}) \times \{a\} \times (\Sigma \cup \{\#\}) \to S$ for each (guessed) input symbol $a \in \Sigma$. These mappings are used for the simulation of the nondeterministic first transitions of the cells of M. The blocks of four input symbols together with the corresponding mappings are shifted to the right such that each of the leftmost $\frac{n+1}{4}$ cells gets one block of symbols and mappings. When the communication cell guesses the end-of-input symbol the first phase ends.

In order to verify that the guesses are correct, the four symbols are entered at the end of a ring store respectively. In each step, the symbol at the front of the ring is removed and compared with the actual input symbol. If both match, the guessed symbol is correct, otherwise it is not. In case of a mismatch or a wrongly guessed number of input symbols the computation blocks in a neutral state.

Phase 2: At time step $\frac{n+1}{4} + 1$ the communication cell initiates an FSSP synchronization of the leftmost $\frac{n}{4}$ cells. The blocks of four input symbols together with the corresponding mappings arrive at their destination cells $0 \leq i \leq \frac{n+1}{4} - 1$ at time $2i+1$. The initial signal for the FSSP arrives at cell i at time $\frac{n+1}{4}+1+i > 2i+1$. So, each cell starts Phase 2 after finishing Phase 1. Altogether, Phase 2 is finished when all cells are synchronized at time $\frac{n+1}{4}+1+2 \cdot \frac{n+1}{4}-2 = 3 \cdot \frac{n+1}{4}-1$.

Phase 3: Due to the compressed representation, M' can simulate M with fourfold speed. In order to simulate the nondeterministic transitions which the cells of M perform during the first time step, the cells of M' apply the nondeterministically guessed mappings to its local configurations. Thus, M' simulates the nth step of M at time step $3 \cdot \frac{n+1}{4} - 1 + \frac{n+1}{4} = n$.

Since the verification of the guessed input takes $n+1$ time steps, we conclude that the total time complexity of M' is $t(n) = n + 1$, that is realtime.

Finally, M' accepts if and only if the input has been guessed correctly and M accepts, and it rejects if and only the input has been guessed correctly and M rejects. Since M has the self-verifying property, M' is self-verifying as well. So, we have $L(M') = L(M)$. □

Lemma 7 and Lemma 8 reveal the equality of the next theorem.

Theorem 9. $\mathscr{L}_{lt}(SVCA) = \mathscr{L}_{rt}(SVCA) = \mathscr{L}_{rt}(SVIA) = \mathscr{L}_{lt}(SVIA)$.

In particular, now we can deduce that even the restricted nondeterminism gained in considering a self-verifying communication cell strictly increases the computational capacity of realtime iterative arrays. That is, the inclusion $\mathscr{L}_{rt}(\text{IA}) \subset \mathscr{L}_{rt}(\text{SVIA})$ is strict.

Proposition 10. *The family $\mathscr{L}_{rt}(IA)$ is strictly included in $\mathscr{L}_{rt}(SVIA)$.*

Proof. The relations $\mathscr{L}_{rt}(\text{IA}) \subset \mathscr{L}_{lt}(\text{IA}) = \mathscr{L}_{lt}(\text{CA})$ are known (see, for example, [10]). Since $\mathscr{L}_{lt}(\text{IA}) \subseteq \mathscr{L}_{lt}(\text{SVIA})$ the assertion follows. □

Theorem 9 shows that a sequential input mode and one nondeterministic cell can be traded for parallel input mode and all cells nondeterministic once only, and vice versa. To this end, it does not matter whether the computations are in realtime or lineartime. But what about the world beyond lineartime? Are self-verifying arrays stronger than deterministic ones? Or weaker than nondeterministic ones? The open question of the strictness of one of the inclusions $\mathscr{L}_{lt}(\text{IA}) \subseteq \mathscr{L}_{rt}(\text{SVIA}) \subseteq \mathscr{L}(\text{SVCA}) \subseteq \mathscr{L}(\text{NCA})$ is strongly related to famous open problems in complexity theory (see [11]). Note that at the top of this hierarchy are devices that may have an exponential time complexity (due to the space bound).

5 Closure Properties

Here we turn to explore the closure properties of the family of realtime SVIA languages. They are summarized in Table 1. We start with the Boolean operations.

Proposition 11. *The family of languages accepted by realtime SVIA is closed under complementation, union, and intersection.*

The closure under reversal is of crucial importance. It is an open problem for $\mathscr{L}_{rt}(\text{CA})$ and, equivalently, for $\mathscr{L}_{lt}(\text{OCA})$ (OCAs are CAs where information can only be passed from right to left, that is, the new state of a cell does not depend on that of its left neighbor). Moreover, it is linked with the open closure property under concatenation for the same family and, hence, with the question whether lineartime CAs are more powerful than realtime CAs. It is known that the family $\mathscr{L}_{rt}(\text{IA})$ is not closed under reversal, while the family $\mathscr{L}_{lt}(\text{IA})$ is closed.

Proposition 12. *The family of languages accepted by realtime SVIA is closed under reversal.*

Proof. Given some realtime SVIA M, Theorem 9 says that there is an equivalent lineartime SVCA M' with transition functions δ_{nd} and δ_d.

Now, the arguments of the local transition functions are interchanged. That is, $\delta'_d(s_3, s_2, s_1)$ is defined to be $\delta_d(s_1, s_2, s_3)$, and $\delta'_{nd}(s_3, s_2, s_1)$ is defined to be $\delta_{nd}(s_1, s_2, s_3)$. The resulting device M'' with transition functions δ'_{nd} and δ'_d. accepts the reversal $L(M)^R$ at the *rightmost cell*. Then the result is sent as a signal to the leftmost cell. Altogether, M'' is still a lineartime SVCA and, thus, $L(M)^R \in \mathscr{L}_{lt}(\text{SVCA}) = \mathscr{L}_{rt}(\text{SVIA})$. □

Proposition 13. *The family of languages accepted by realtime SVIA is closed under concatenation.*

Proof. Let $L_1, L_2 \in \mathscr{L}_{rt}(\text{SVIA})$. If the empty word belongs to L_1 then language L_2 belongs to the concatenation and vice versa. Since the family of languages accepted by realtime SVIA is closed under union, it remains to consider languages $L_1, L_2 \in \mathscr{L}_{rt}(\text{SVIA})$ that do not contain the empty word. Let M_1 and M_2 be realtime SVIA that accept L_1 and L_2.

Since the family $\mathscr{L}_{rt}(\text{SVIA})$ is closed under reversal, there is a realtime SVIA M_2^R that accepts the reversal L_2^R of L_2.

A realtime SVIA M that accepts the concatenation $L_1 \cdot L_2$ works as follows. First we describe two tasks that are performed by M in parallel.

Basically, the first task is to read the input and to simulate M_1. In addition, the input is stored into a ring whose front is the communication cell. Moreover, in any simulation step, M tests whether it would accept or reject the input prefix read so far by checking if it would accept or reject when the next input symbol were the end-of-input symbol $\lhd$. If the current input prefix is accepted or rejected, the input symbol stored into the ring is marked suitably.

The second task is to guess the reversal of the input symbol by symbol. The guessed reversal is stored into a pushdown store whose top is the communication cell. Additionally, the realtime SVIA M_2^R is simulated on the guessed input. Similarly as for the first task, if the current prefix of the guessed input would be accepted or rejected, the guessed input symbol stored into the pushdown store is marked suitably.

Let $x_1x_2 \cdots x_n$ be the actual input. It is stored in the ring when the end-of-input symbol appears. At that time, let $y_1y_2 \cdot y_n$ be the content of the pushdown store (from top to bottom). Clearly, M has guessed the reversal of the input correctly if and only if $x_1x_2 \cdots x_n = y_1y_2 \cdots y_n$. So, after having read the end-of-input symbol, the SVIA M verifies the guessed reversal of the input by successively removing symbols from the ring and pushdown store and testing whether they match. If M detects any mismatch it blocks in a neutral state.

Now, assume that the reversal of the input has been guessed correctly.

Already while verifying the guesses by successively scanning the ring and the queue, the SVIA M tests whether for some $1 \leq i \leq n-1$ input symbol x_i is

marked by the simulation of M_1 and symbol y_{i+1} is marked by the simulation of M_2^R.

Case 1: $x_1 \cdots x_n \in L_1L_2$, say $x_1 \cdots x_i \in L_1$. Then there are computations by M_1 accepting $x_1 \cdots x_i$ and by M_2^R accepting $y_n y_{n-1} \cdots y_{i+1} = x_n x_{n-1} \cdots x_{i+1} = (x_{i+1} \cdots x_n)^R$. Having M accept an input iff x_i is marked by M_1 and symbol y_{i+1} is marked by M_2^R makes M accept all words in L_1L_2.

Case 2: $x_1 \cdots x_n \notin L_1L_2$. In this case for each i either $x_1 \cdots x_i \notin L_1$ or $x_n \cdots x_{i+1} \notin L_2^R$ or both. That means that for each i there are computations by M_1 and M_2^R for the respective inputs such that at least one of both rejects. Hence, there will be a computation for M which correctly explicitly rejects an input, if for any two adjacent cells always at least one of them is marked rejecting.

In any other case, the leftmost cell remains in a neutral state.

For the computation on input of length n the SVIA M takes $n+1$ steps to read (and guess) the input for the tasks, and further $n+1$ steps to verify the guesses and test the markings. So, M works in lineartime which ca be sped-up to realtime. □

Next, we turn to the operations homomorphism and inverse homomorphism.

Proposition 14. *The family of languages accepted by realtime SVIA is not closed under arbitrary homomorphisms.*

Proof. It is shown in [12] that every recursively enumerable language would be contained in the family $\mathscr{L}_{rt}$(SVOCA) if the family were closed under arbitrary homomorphisms. Since $\mathscr{L}_{rt}(\text{SVOCA}) \subseteq \mathscr{L}_{rt}(\text{SVCA}) = \mathscr{L}_{rt}(\text{SVIA})$ the same is true for the family $\mathscr{L}_{rt}$(SVIA), a contradiction due to the time bound. □

Proposition 15. *The family of languages accepted by realtime SVIA is closed under inverse homomorphisms.*

The closure properties of $\mathscr{L}_{rt}$(SVIA) with respect to iteration (Kleene star) and non-erasing homomorphisms are open problems. They are settled for non-deterministic devices since, basically, for iteration it is sufficient to guess the positions in the input at which words are concatenated, and for non-erasing homomorphism it is sufficient to guess the pre-image of the input. However, self-verifying devices have to reject explicitly if the input does not belong to the language. Intuitively, this means that they have to 'know' that all possible guesses either do not lead to accepting computations or are 'wrong.'

6 Decidability Questions

First we note that the membership problem is obviously decidable for SVIAs obeying a time-computable time complexity.

On the other hand, in [10] it is observed that for any language family that effectively contains $\mathscr{L}_{rt}$(IA), the problems emptiness, universality, finiteness, infiniteness, regularity, and context-freeness are not semidecidable. Since we know $\mathscr{L}_{rt}(\text{IA}) \subset \mathscr{L}_{lt}(\text{IA}) \subseteq \mathscr{L}_{lt}(\text{SVIA}) = \mathscr{L}_{rt}(\text{SVIA})$ we derive the next corollary.

Family	—	$\cup$	$\cap$	R	$\cdot$	$*$	h_λ	h	h^{-1}
$\mathscr{L}_{rt}$(SVIA)	✓	✓	✓	✓	✓	?	?	✗	✓
$\mathscr{L}_{rt}$(IA)	✓	✓	✓	✗	✗	✗	✗	✗	✓
$\mathscr{L}_{lt}$(IA)	✓	✓	✓	✓	?	?	?	✗	✓

Table 1. Closure properties of the language family $\mathscr{L}_{rt}$(SVIA) in comparison with the families $\mathscr{L}_{rt}$(IA) and $\mathscr{L}_{lt}$(IA), where h_λ denotes λ-free homomorphisms.

Corollary 16. *The problems emptiness, universality, finiteness, infiniteness, inclusion, equivalence, regularity, and context-freeness are not semidecidable for realtime IAs and thus for realtime SVIAs.*

In [12] it is shown that the problem to decide whether a given realtime one-way cellular automaton is self-verifying or not is undecidable. Unfortunately, the result has no direct implications for the same question for iterative arrays. However, the undecidability for cellular automata is shown by a reduction of the emptiness problem. We turn to prove the undecidability for iterative arrays as well. Moreover, we use a reduction of the emptiness and universality problem, but the reduction itself is different. Since general iterative arrays do not have neutral or rejecting states (only accepting and non-accepting states), there is no partitioning of the state set. So, the decidability can be asked for a given fixed partitioning or for the existence of a partitioning. We first consider the latter question.

Theorem 17. *Given a realtime (non)deterministic iterative array M with state set S and accepting states F_+, it is not even semidecidable whether there exists $F_- \subseteq (S \setminus F_+)$ such that M is an SVIA with respect to the sets F_+ and F_-.*

Proof. Let $M_0 = \langle S, \Sigma, F_+, s_0, \triangleleft, \delta_{nd}, \delta_d \rangle$ be an arbitrary realtime IA. We safely may assume that a cell which has left the quiescent state will never enter the quiescent state again. This behavior can be implemented by adding a new state that plays the role of the quiescent state. If necessary, the new state can be entered instead of s_0.

We modify M_0 to $M_1 = \langle S', \Sigma', F'_+, s_0, \triangleleft, \delta'_{nd}, \delta'_d \rangle$ by adding a new input symbol \$ and two new states p_+ and p_0. So, we set $S' = S \cup \{p_+, p_0\}$, $\Sigma' = \Sigma \cup \{\$\}$, and $F'_+ = F_+ \cup \{p_+\}$. The intention is that a \$ in the input causes the IA to simulate a step on the end-of-input symbol $\triangleleft$ (in restricted form) and to reinitialize the computation by letting the cells enter the quiescent state again (which is impossible in M_1). Therefore, the transition function δ'_{nd} is basically δ_{nd} extended by transitions for the input symbol \$ and the states p_+ and p_0. When a \$ appears in the input, the communication cell enters state p_+ if it could enter an accepting state on the end-of-input symbol $\triangleleft$. For all $s_1, s_2 \in S$,

$$\delta'_{nd}(\$, s_1, s_2) = \{p_+\} \text{ if } \delta_{nd}(\triangleleft, s_1, s_2) \cap F_+ \neq \emptyset.$$

Otherwise it enters state p_0: $\delta'_{nd}(\$, s_1, s_2) = \{p_0\}$ if $\delta_{nd}(\lhd, s_1, s_2) \cap F_+ = \emptyset$. In state p_+ or p_0 the computation continues as it would from the very beginning. For $p \in \{p_+, p_0\}$, all $a \in \Sigma' \cup \{\lhd\}$, and all $s \in S'$, $\delta'_{nd}(a, p, s) = \delta'_{nd}(a, s_0, s_0)$. In order to implement the reinitialization of the other cells, recall that δ_d drives no non-quiescent cell into the quiescent state. So, we can utilize the quiescent state as a signal sent by the communication cell. The signal causes the reinitialization of the cells passed through. So, the transition function δ'_d is basically δ_d extended as follows. For $p \in \{p_+, p_0\}$ and all $s_1, s_2 \in S$,

$$\delta'_d(p, s_1, s_2) = s_0, \quad \delta'_d(s_0, s_1, s_2) = s_0, \quad \delta'_d(s_1, s_0, s_2) = \delta(s_1, s_0, s_0).$$

Therefore $L(M_1)$ consists of all concatenations of \$ separated words u_i such that at least one u_i is in $L(M_0)$. In particular $L(M_1) \cap \Sigma^* = L(M_0)$.

We claim that there exists $F'_- \subseteq (S' \setminus F'_+)$ such that the iterative array $M_2 = \langle S', \Sigma', F'_+, F'_-, s_0, \lhd, \delta'_{nd}, \delta'_d \rangle$ is self-verifying if and only if $L(M_0)$ is empty or coincides with Σ^*. Observe that $L(M_2) = L(M_1)$ because both have the same set of accepting states.

If $L(M_0)$ is empty then $L(M_1)$ is empty. Therefore, the communication cell will never enter an accepting state from F'_+. So, we safely may set $F'_- = (S' \setminus F'_+)$ and obtain that M_2 is self-verifying. Similarly, if $L(M_0) = \Sigma^*$ then $L(M_1) = \Sigma'^*$, and we safely may set $F'_- = \emptyset$ to obtain a self-verifying IA.

Now assume that $L(M_0)$ and, thus, $L(M_2)$ neither be empty nor contain all words over the input alphabet. Then there exists some $u \in L(M_0)$ and some $v \notin L(M_0)$. We consider the computation of M_2 on input $u\$v$. Since M_0 accepts u, the IA M_2 enters an accepting state while processing the input prefix $u\$$ (its computation is a simulation of M_0 on $u\lhd$). Then the computation of M_2 is reinitialized and continues with a simulation of M_0 on input v. Since $v \notin L(M_0)$, in this phase, M_2 cannot accept v either. However, since it already was in an accepting state and its overall answer is already *yes*, M_2 cannot enter a contradictory rejecting state in this phase either. This implies that the communication cell of M_2 on input v will only assume neutral states and, thus, neither accept nor reject v. That is, M_2 is not self-verifying and the claim follows.

From the construction of M_2 and the claim we conclude that the semidecidability of the problem in question implies the semidecidability of the emptiness or universality problem for realtime IAs contradicting Corollary 16. □

What about the undecidability if we provide a partitioning of its state set? Can we test if this partitioning makes the IA self-verifying? The answer is no, since for a given realtime iterative array with accepting state set F_+ there are only finitely many partitions induced by setting $F_- \subseteq (S \setminus F_+)$. All these could be tested in parallel. Now the problem in question can be semidecided if the test is successful for at least one partitioning.

Corollary 18. *Given a realtime (non)deterministic iterative array M with state set S and partitioning $S = F_+ \dot\cup F_- \dot\cup F_0$, it is not semidecidable whether M is an SVIA with respect to the partitioning.*

By Lemma 2 any deterministic iterative with a time-computable time complexity can effectively be made self-verifying. But it is non-semidecidable whether it already *is* self-verifying. This non-semidecidability carries immediately over to nondeterministic iterative arrays. However, it is an open problem whether any nondeterministic iterative with a time-computable time complexity can effectively be made self-verifying. In fact, it is an open problem whether the family of languages accepted by realtime nondeterministic iterative arrays is closed under complementation or not.

References

1. Buchholz, Th., Klein, A., Kutrib, M.: Iterative arrays with limited nondeterministic communication cell. In: Words, Languages and Combinatorics III. pp. 73–87. World Scientific Publishing (2003)
2. Buchholz, Th., Kutrib, M.: Some relations between massively parallel arrays. Parallel Comput. 23, 1643–1662 (1997)
3. Cole, S.N.: Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE Trans. Comput. C-18, 349–365 (1969)
4. Čulik II, K., Yu, S.: Iterative tree automata. Theoret. Comput. Sci. 32, 227–247 (1984)
5. Duris, P., Hromkovic, J., Rolim, J.D.P., Schnitger, G.: Las Vegas versus determinism for one-way communication complexity, finite automata, and polynomial-time computations. In: Theoretical Aspects of Computer Science (STACS 1997). LNCS, vol. 1200, pp. 117–128. Springer (1997)
6. Dyer, C.R.: One-way bounded cellular automata. Inform. Control 44, 261–281 (1980)
7. Fernau, H., Kutrib, M., Wendlandt, M.: Self-verifying pushdown automata. In: Non-Classical Models of Automata and Applications (NCMA 2017). books@ocg.at, vol. 329, pp. 103–117. Austrian Computer Society, Vienna (2017)
8. Jirásková, G., Pighizzini, G.: Optimal simulation of self-verifying automata by deterministic automata. Inform. Comput. 209, 528–535 (2011)
9. Kutrib, M.: Cellular automata – a computational point of view. In: New Developments in Formal Languages and Applications, chap. 6, pp. 183–227. Springer (2008)
10. Kutrib, M.: Cellular automata and language theory. In: Encyclopedia of Complexity and System Science, pp. 800–823. Springer (2009)
11. Kutrib, M.: Complexity of one-way cellular automata. In: Cellular Automata and Discrete Complex Systems (AUTOMATA 2014). LNCS, vol. 8996, pp. 3–18. Springer (2015)
12. Kutrib, M., Worsch, T.: Self-verifying cellular automata. In: Cellular Automata for Research and Industry (ACRI 2018). LNCS, vol. 11115, pp. 340–351. Springer (2018)
13. Mazoyer, J., Terrier, V.: Signals in one-dimensional cellular automata. Theoret. Comput. Sci. 217, 53–80 (1999)
14. Smith III, A.R.: Real-time language recognition by one-dimensional cellular automata. J. Comput. System Sci. 6, 233–253 (1972)