RAIRO-Theor. Inf. Appl. 52 (2018) 235–251 https://doi.org/10.1051/ita/2018018

AUTOMATA WITH CYCLIC MOVE OPERATIONS FOR PICTURE LANGUAGES[☆],☆☆

Friedrich $\mathrm{Otto}^{1,*}$ and František $\mathrm{Mr}\mathrm{\acute{a}z}^2$

Abstract. Here, we study the cyclic extensions of Sgraffito automata and of deterministic twodimensional two-way ordered restarting automata for picture languages. Such a cyclically extended automaton can move in a single step from the last column (or row) of a picture to the first column (or row). For Sgraffito automata, we show that this cyclic extension does not increase the expressive power of the model, while for deterministic two-dimensional two-way restarting automata, the expressive power is strictly increased by allowing cyclic moves. In fact, for the latter automata, we take the number of allowed cyclic moves in any column or row as a parameter, and we show that already with a single cyclic move per column (or row) the deterministic two-dimensional extended two-way restarting automaton can be simulated. On the other hand, we show that two cyclic moves per column or row already give the same expressive power as any finite number of cyclic moves.

Mathematics Subject Classification. 68Q45.

Accepted November 21, 2018.

1. INTRODUCTION

In the literature, one finds many different types of grammars and automata for defining classes of picture languages (for a survey see, *e.g.*, [8]). In particular, a lot of work has been devoted to defining and characterizing classes of picture languages that would correspond to the class of regular "string" languages. Eventually, an agreement was reached that the class REC of *recognizable languages* of Giammarresi and Restivo [7] is such a class. Unfortunately, this class contains some NP-complete languages [12], which means that in general the membership problem for a recognizable picture language can be quite complex.

Motivated by this observation, the current authors initiated a research program in cooperation with Daniel Průša for finding a two-dimensional automaton that, on the one hand, is conceptually simple and more powerful than the class DREC of deterministic recognizable languages of [1, 2], but that, on the other hand, only accepts the regular languages when restricted to the one-dimensional case (that is, string languages), that has a membership problem that is decidable in polynomial time, and that has nice closure properties.

 $^{^{\}circ}$ The results of this paper have been announced at NCMA 2016 in Debrecen, Hungary, August 2016. An extended abstract appeared in the proceedings of that conference [18].

Keywords and phrases: Picture language, Sgraffito automaton, restarting automaton, cyclic move operation.

¹ Fachbereich Elektrotechnik/Informatik, Universität Kassel, 34109 Kassel, Germany.

² Charles University, Faculty of Mathematics and Physics, Malostranské nám. 25, 118 25 Prague 1, Czech Republic.

^{*} Corresponding author: f.ottoQuni-kassel.de

As a first such model, the Sgraffito automaton was introduced and studied in [20, 22-24]. The nondeterministic Sgraffito automaton is quite powerful, as it accepts all recognizable picture languages, but the deterministic Sgraffito automaton meets most of the properties above, and it is more expressive than the four-way alternating automaton [11] and the deterministic four-way one-marker automaton [3], and it accepts the sudoku-deterministically recognizable picture languages [4].

In the next step, the restarting automaton, which was introduced in [10] as a formal device to model the linguistic technique of *analysis by reduction*, has been extended to models that process two-dimensional inputs. The first such model is the *restarting tiling automaton*, which is a stateless device with a two-by-two window [19, 21] that works in cycles. In each cycle it scans the current picture based on a given scanning strategy until, at some place, it performs a rewrite step and restarts, which completes the current cycle. If no rewrite operation can be performed, then the automaton halts after scanning the current picture completely. It is said to accept if at that point the current picture satisfies certain local conditions, similar to a tiling automaton (see, e.g., [8]).

Then in [14] we introduced the *deterministic two-dimensional three-way ordered restarting automaton* (or det-2D-3W-ORWW-automaton, for short) that works more in the spirit of the restarting automata on strings. Such an automaton has a finite-state control and a window of size three-by-three. It also works in cycles, where in each cycle it scans the current rectangular picture starting at the top left corner. Based on the current state and the contents of its window, it can change its state and move either to the right, down, or up, but not to the left. It keeps on moving until it either halts, accepting or rejecting, or until it performs a rewrite step, in which it replaces the symbol in the middle of its window by a symbol that is strictly smaller with respect to a given ordering on its tape alphabet. After performing such a rewrite, the automaton restarts immediately, which completes the current cycle. When restricted to one-dimensional inputs (that is, strings), then this device just accepts the regular languages. For two-dimensional inputs, however, it is quite powerful, as it can simulate the deterministic Sgraffito automaton. However, because it can perform up and down movements, while it can only perform move-right but no move-left steps, this automaton clearly favours vertical operations over horizontal operations. Hence, it is not surprising that it can accept picture languages consisting of one-column pictures such that the corresponding string languages are not regular.

Next, in [13] we restricted these automata by considering a two-way variant that can only perform move-right and move-down steps. However, such an automaton is not able to scan a given rectangular picture completely within a single cycle, and accordingly, it appears that it is quite weak. In fact, it has been shown in [15] that the deterministic two-dimensional two-way ordered restarting automaton (or det-2D-2W-ORWW-automaton) is equivalent to its stateless variant, and that it can be simulated by the deterministic Sgraffito automaton. Accordingly, we introduced an extended variant, the deterministic two-dimensional extended two-way ordered restarting automaton (det-2D-x2W-ORWW-automaton), in [13]. For these automata the move operations are generalized, using an idea that has also been employed for the returning finite automata studied in [6]: when a move-right step is executed, while the central position of the window is placed in row *i* of the last column, then the window is moved such that its central position of the window is placed in column *j* of the bottommost row, then the window is moved such that its central position is in column j + 1 of the topmost row. In order to avoid infinite sequences of move operations, it is required that in any cycle, the automaton can either use extended move-right or extended move-down steps, but not both.

As it turned out, this automaton is quite expressive, as it can accept the two-dimensional variant of the copy language (see [17]) and already its stateless variant can simulate the deterministic Sgraffito automaton. However, when restricted to picture languages that only consist of one-column pictures, then the det-2D-x2W-ORWW-automaton can only accept languages that are obtained by the operation of rotation from regular (string) languages. Hence, it follows that these automata cannot accept all those picture languages that are accepted by det-2D-3W-ORWW-automata. In fact, the class of picture languages that these automata accept is incomparable under inclusion to the class of picture languages that are accepted by det-2D-3W-ORWW-automata.

Here, we introduce and discuss the extensions of the Sgraffito automaton and of the deterministic twodimensional two-way ordered restarting automaton that we obtain by admitting *cyclic* move operations, called *jumps*. Whenever an automaton scans the rightmost symbol in a row (the bottommost symbol in a column) of a picture, then by performing a jump the automaton moves in a single step to the leftmost symbol in that row (the topmost symbol in that column). In fact, we take the number of jumps that the automaton may execute on any column or row as a parameter, and we show that already with a single jump (per column or row) the det-2D-2W-ORWW-automaton can simulate the det-2D-x2W-ORWW-automaton. In fact, the 1-jump det-2D-2W-ORWW-automaton is already more expressive than the det-2D-x2W-ORWW-automaton, as the former can already accept some string languages that are not even growing context-sensitive. On the other hand, for bounded languages (see Sect. 4 for the definition) the det-2D-x2W-ORWW-automaton is equivalent to the det-2D-2W-ORWW-automaton extended by jumps.

Further, we will see that for Sgraffito automata, jumps do not help, as they can be simulated by ordinary Sgraffito automata. This is true in the nondeterministic as well as in the deterministic setting. Further, for det-2D-2W-ORWW-automata, we will show that two jumps per column or row suffice to simulate any bounded number of jumps, and for string languages, already one jump is as powerful as any bounded number of jumps.

This paper is structured as follows. In Section 2, we restate in short the basic notions on pictures and picture languages and we recall the definitions of the Sgraffito automaton, the det-2D-2W-ORWW-automaton and its extended variant and the relationship between the corresponding classes of picture languages. Then, in Section 3, we introduce and study the Sgraffito automaton extended by jumps, and in Section 4, we consider the det-2D-2W-ORWW-automaton extended by jumps. The paper closes with a short summary and some open problems.

2. Picture languages, Sgraffito automata, and two-dimensional restarting automata

Here, we use the common notation and terms on strings, pictures, and picture languages (see, e.g., [8]). Let Σ be a finite alphabet. Then Σ^* is the set of strings over Σ . For $w \in \Sigma^*$, |w| denotes the *length* of the string w, λ denotes the empty string, which is of length 0, and for a letter $a \in \Sigma$, $|w|_a$ denotes the *a*-length of w, which is the number of occurrences of the symbol a in w. Further, $\Sigma^{*,*}$ is the set of (rectangular) pictures over Σ , that is, an element $P \in \Sigma^{*,*}$ is a two-dimensional array of symbols from Σ . We say that P is of dimension (m, n), written as $P \in \Sigma^{m,n}$, to express the fact that P has m rows and n columns, and we take $\ell_1(P) = m$ and $\ell_2(P) = n$. Here, in order to simplify the definitions of the various types of automata for processing pictures, we will only consider non-empty pictures, that is, for each picture P considered we will assume that $\ell_1(P) > 0$ and $\ell_2(P) > 0$. Further, P(i, j) denotes the symbol at row i and in column j for all $1 \le i \le m$ and $1 \le j \le n$. We introduce a set of five special markers (sentinels) $S = \{\vdash, \dashv, \top, \bot, \#\}$, and we assume that $\Sigma \cap S = \emptyset$ for any alphabet Σ considered. In order to enable an automaton to detect the border of a picture P easily, we define the boundary picture \hat{P} over $\Sigma \cup S$ of dimension (m+2, n+2). It is illustrated by the following schema:

#	Т	Т		Т	Т	#
+ : +			Р			Т : Т
#	\perp	\perp	•••	\perp	\perp	#

First we recall the definition of the Sgraffito automaton from [20]. However, in order to obtain a uniform way of presenting the various types of automata considered in the current paper, we modify the definition slightly. The Sgraffito automaton is a two-dimensional extension of the one-dimensional *constant-visit machine* studied by Hennie [9].

Let $\mathcal{H} = \{R, L, D, U, Z\}$ be the set of possible *head movements*, where the first four elements denote directions (right, left, down, up), while Z stands for zero (no) movement. Furthermore, let $\nu : (S \setminus \{\#\}) \to \mathcal{H}$ denote the mapping that is defined by $\nu(\vdash) = R$, $\nu(\dashv) = L$, $\nu(\top) = D$, and $\nu(\perp) = U$, that is, for each occurrence of a

sentinel from $\{\vdash, \dashv, \top, \bot\}$ within a boundary picture \hat{P} , ν yields the direction to the nearest field of the proper picture P.

Definition 2.1. A two-dimensional Sgraffito automaton (2SA) is given by a 7-tuple $A = (Q, \Sigma, \Gamma, \delta, q_0, Q_F, >)$, where

- -Q is a finite, nonempty set of states,
- Σ is a finite input alphabet,
- Γ is a finite working alphabet such that $\Sigma \subseteq \Gamma$,
- $-q_0 \in Q$ is the initial state,
- $-Q_F \subseteq Q$ is the set of final states,
- $\delta: (Q \setminus Q_F) \times (\Gamma \cup S) \to 2^{Q \times (\Gamma \cup S) \times \mathcal{H}}$ is a transition relation such that the following properties are satisfied for each pair $(q, a) \in (Q \setminus Q_F) \times (\Gamma \cup S)$ and each transition $(q', a', d) \in \delta(q, a)$:
 - if $a \in S$, then $d = \nu(a)$ and a' = a, and
 - if $a \notin S$, then $a' \notin S$,
- and > is a partial ordering on Γ such that a > a' holds for all transitions $(q', a', d) \in \delta(q, a)$ satisfying $a \in \Gamma$.

The Sgraffito automaton A is deterministic, that is, a 2DSA, if $|\delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Gamma \cup S$.

The definition above differs from the original definition in that the partial ordering has replaced the weight function used in [20], but it is rather obvious that the two are equivalent.

The notions of configuration and computation of the 2SA A are defined as usual. Let $P \in \Sigma^{*,*}$ be a given input picture for A. In the initial configuration of A on input P, the working tape contains the boundary picture \hat{P} , A is in its initial state q_0 , and its head scans the top-left corner of P, that is, cell (2, 2) of \hat{P} . Now Amoves across this boundary picture, and it cannot leave the space covered by \hat{P} . Furthermore, when executing a transition step at a position (i, j) of P, then it must replace the current symbol, say a, at that position by a symbol, say a', that is strictly smaller with respect to the partial ordering >. It follows that A can visit each position of P, and therewith of \hat{P} , at most $|\Gamma|$ many times. The automaton A is said to accept P if there is a computation of A that starts in the initial configuration on input P and that finishes in a state from Q_F . By L(A) we denote the picture language that consists of all pictures that are accepted by A, and $\mathcal{L}(2SA)$ ($\mathcal{L}(2DSA)$)) denotes the class of all picture languages that are accepted by (deterministic) Sgraffito automata.

Next we restate in short the definition of the det-2D-2W-ORWW-automaton. This automaton has a read/write window of size three-by-three, which it can move across a given bordered picture \hat{P} . For doing so, it uses the set $\mathcal{H} = \{R, D\}$ of possible *window movements*, where R denotes a step to the *right* and D a step *down*. Observe that no movement to the left or up is allowed.

Definition 2.2. A deterministic two-dimensional two-way ordered restarting automaton, a det-2D-2W-ORWWautomaton for short, is given through a 6-tuple $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$, where

- -Q is a finite, nonempty set of states,
- $-\Sigma$ is a finite input alphabet,
- Γ is a finite tape alphabet containing Σ ,
- m $q_0 \in Q$ is the initial state,
- > is a partial ordering on Γ , and
- $-\delta: Q \times (\Gamma \cup S)^{3,3} \to (Q \times \mathcal{H}) \cup \Gamma \cup \{\text{Accept}\}\$ is a partial function, called transition function, that satisfies the following three restrictions for all $q \in Q$ and all $C \in (\Gamma \cup S)^{3,3}$:
 - if $C(2,3) = \dashv$, then $\delta(q,C) \neq (q',\mathbf{R})$ for all $q' \in Q$,
 - if $C(3,2) = \bot$, then $\delta(q,C) \neq (q',D)$ for all $q' \in Q$,
 - if $\delta(q, C) = b \in \Gamma$, then C(2, 2) > b with respect to the partial ordering >.

We say that the window of M is at position (i, j) to express the fact that the field in the center of the window is at row i and column j. Given a picture $P \in \Sigma^{m,n}$ as input, M begins its computation in state q_0 with its read/write window reading the subpicture of dimension (3,3) of \hat{P} at the upper left corner, that

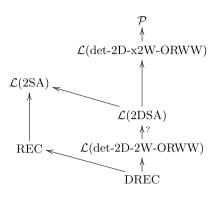


FIGURE 1. Hierarchy of classes of picture languages accepted by various types of two-dimensional automata. The question mark indicates an inclusion not known to be proper.

is, the window is at position (1,1) of P. Applying its transition function, M now moves through \hat{P} until it reaches a state p and a position with current contents C of the read/write window such that either $\delta(p, C)$ is undefined, or $\delta(p, C) = \text{Accept}$, or $\delta(p, C) = b$ for some letter $b \in \Gamma$. In the first case, M gets stuck, and so the current computation ends without accepting, in the second case, M halts and accepts, and in the third case, M replaces the symbol C(2, 2) by the symbol b, which by definition is smaller than C(2, 2) with respect to the partial ordering >, moves its read/write window back to the upper left corner, and reenters its initial state q_0 . This latter step is therefore called a *combined rewrite/restart step*. A subcomputation that begins in an initial configuration or immediately after a rewrite/restart step, that consists of move operations, and that ends with a rewrite/restart step is called a *cycle*. Thus, a computation of M on a given input picture consists of a sequence of cycles that is followed by a *tail computation*, which is defined like a cycle but without the final rewrite/restart step. Hence, a tail computation either ends with an accept step or with M getting stuck. A picture $P \in \Sigma^{*,*}$ is *accepted* by M, if the computation of M on input P ends with an accept step. By L(M) we denote the language consisting of all pictures over Σ that M accepts.

Finally, we recall the deterministic two-dimensional *extended* two-way ordered restarting automaton that is obtained from the det-2D-2W-ORWW-automaton by extending the move-right and move-down steps in the same way as the move-right steps are generalized in the *returning finite automaton*, which reads a rectangular picture row by row from top to bottom [6].

Definition 2.3. A deterministic two-dimensional extended two-way ordered restarting automaton, a det-2D-x2W-ORWW-automaton for short, is given through a 6-tuple $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$, where all components are defined as for a det-2D-2W-ORWW-automaton. However, the move-right and move-down steps are extended as follows:

- 1. As long as the window does neither contain the right border marker nor the bottom marker, move-right and move-down steps can be used freely.
- 2. When the window contains the right border marker, but not the bottom marker, then an *extended moveright step* shifts the window to the beginning of the next row, that is, if the central position of the window is on the last field of row *i* for some $i < \ell_1(P)$, then it is now placed on the first field of row i + 1.
- 3. When the window contains the bottom marker, but not the right border marker, then an *extended move*down step shifts the window to the top of the next column, that is, if the central position of the window is on the bottommost field of column j for some $j < \ell_2(P)$, then it is now placed on the topmost field of column j + 1.
- 4. In any cycle, as soon as *M* executes an extended move-right (move-down) step, then for the rest of this cycle, it cannot execute any extended move-down (move-right) step.

Acceptance is defined in the same way as for det-2D-2W-ORWW-automata, and the language consisting of all pictures accepted by M is denoted as L(M).

Concerning the expressive power of the various types of two-dimensional automata introduced so far, the inclusions presented in the diagram in Figure 1 have been obtained. Here \mathcal{P} denotes the class of all picture languages for which the membership problem is decidable in deterministic polynomial time. We would like to point out that the class of picture languages that are accepted by the aforementioned returning finite automata is also properly contained in the class $\mathcal{L}(2DSA)$, as these automata can be simulated by deterministic four-way automata.

3. Cyclically extended Sgraffito Automata

A cyclically extended Sgraffito automaton is a Sgraffito automaton that, on the rightmost (leftmost) cell of a row, can make a cyclic move-right (move-left) step, which takes it to the leftmost (rightmost) cell of that very row. Analogously, on the topmost (bottommost) cell of a column, it can make a cyclic move-up (move-down) step, which will take it to the bottommost (topmost) cell of that very column. Formally, it is defined as follows.

Definition 3.1. A cyclically extended Sgraffito automaton is given by a 7-tuple $A = (Q, \Sigma, \Gamma, \delta, q_0, Q_F, >)$, where the components $Q, \Sigma, \Gamma, q_0, Q_F$, and > are defined as for a two-dimensional Sgraffito automaton, while the transition relation $\delta : (Q \setminus Q_F) \times (\Gamma \cup S) \rightarrow 2^{Q \times (\Gamma \cup S) \times \mathcal{H}}$ satisfies the following properties for each pair $(q, a) \in (Q \setminus Q_F) \times (\Gamma \cup S)$ and each transition $(q', a', d) \in \delta(q, a)$:

- if $a \in \Gamma$, then $a' \in \Gamma$ and a > a',
- if $a = \dashv$, then a' = a and either d = L or d = R, that is, either A moves back to the left neighboring tape field or it executes a cyclic move-right step, called a *right jump*, in which case it moves its head to the field immediately to the right of the left sentinel ⊢ on the current row;
- if $a = \vdash$, then a' = a and either $d = \mathbb{R}$ or $d = \mathbb{L}$, that is, either A makes a single move-right step or it makes a *left jump* which takes its head to the field immediately to the left of the right sentinel ⊣ on the current row;
- if $a = \top$, then a' = a and either d = D or d = U, that is, either A moves down to the neighboring field immediately below the sentinel or it executes a cyclic move-up step, called an *up jump*, in which case it moves its head to the field immediately above the sentinel \perp in the current column;
- if $a = \bot$, then a' = a and either d = U or d = D, that is, either A makes a single move-up step or it makes a *down jump* which takes its head to the field immediately below the sentinel \top in the current column.

Observe that a right jump involves a move from the right sentinel \dashv to the right neighboring field of the left sentinel \vdash on the same row, and analogously for the other jump operations. Although these jump operations may seem to be quite powerful, we actually have the following result.

Theorem 3.2. From a cyclically extended Sgraffito automaton A, one can construct a Sgraffito automaton B that accepts the same picture language as A. If A is deterministic, then so is B.

Proof. First, we describe the idea of how a Sgraffito automaton B can simulate a given cyclically extended Sgraffito automaton A. Let $A = (Q, \Sigma, \Gamma, \delta, q_0, Q_F, >)$ be a cyclically extended Sgraffito automaton, and let $n = |\Gamma|$. The Sgraffito automaton $B = (Q_B, \Sigma, \Gamma_B, \delta_B, p_0, P_F, >_B)$ has tape alphabet

$$\Gamma_B = \{(a, i, j, k, l) \mid a \in \Gamma, 0 \le i, j, k, l \le n\},\$$

that is, for each letter a of the tape alphabet Γ of A, Γ_B contains symbols of the form (a, i, j, k, l), where $0 \le i, j, k, l \le n$. The partial ordering $>_B$ is defined through

240

that is, either a > b or at least one of the integers i, j, k, l has been replaced by an integer that is strictly larger.

The integers i, j, k, l will be used to count the number of cyclic move operations (jumps) that have been executed on any given row or column, where i (j, k, l) counts the right (left, up, down) jumps. In addition, each input letter $a \in \Sigma$ is identified with the letter (a, 0, 0, 0, 0).

Let P be an input picture for A. Then B simulates the computation of A on P as follows. While inside the picture P, B behaves just like A, that is, if A replaces a symbol a by a symbol b (where a > b), then B replaces the symbol (a, i, j, k, l) at that position by the symbol (b, i, j, k, l). If, however, A is at a border cell, for example, at the rightmost cell of a row, and if A performs a right jump, which will take it to the leftmost cell on that row, then B will simply move left across the corresponding row, thereby replacing each symbol (a, i, j, k, l)encountered by the symbol (a, i + 1, j, k, l). Analogously, a left jump of A from the leftmost cell on a row is simulated by a sweep to the right by B, in which (a, i, j, k, l) is replaced by (a, i, j + 1, k, l). The up and down jumps of A are analogously simulated by down-sweeps or up-sweeps of B, replacing each symbol (a, i, j, k, l) by (a, i, j, k + 1, l) or, respectively, by (a, i, j, k, l + 1).

To define B in detail, we take $Q_B = Q \cup \{(q, \rho) \mid q \in Q, \rho \in \mathcal{H}\}$, that is, for each state $q \in Q$ of A, B has six states: the state q itself and its five copies $(q, \mathbf{R}), (q, \mathbf{L}), (q, \mathbf{D}), (q, \mathbf{U})$, and (q, \mathbf{Z}) . The transition relation δ_B is now defined as follows:

1. For all $q \in Q$, $a \in \Gamma$, and $i, j, k, l \in \{0, 1, ..., n\}$,

$$\delta_B(q, (a, i, j, k, l)) = \{ (q', (b, i, j, k, l), \rho) \mid (q', b, \rho) \in \delta(q, a) \},\$$

that is, each operation $(q', b, \rho) \in \delta(q, a)$ is simply replaced by the operations $(q', (b, i, j, k, l), \rho) \in \delta_B(q, (a, i, j, k, l))$ for all $0 \le i, j, k, l \le n$ and $\rho \in \mathcal{H}$.

2. For all $q \in Q$,

$$\delta_B(q, \dashv) = \{(q', \dashv, \mathbf{L}) \mid (q', \dashv, \mathbf{L}) \in \delta(q, \dashv)\} \cup \{((q', \mathbf{R}), \dashv, \mathbf{L}) \mid (q', \dashv, \mathbf{R}) \in \delta(q, \dashv)\},\$$

that is, a right jump $(q', \dashv, \mathbf{R}) \in \delta(q, \dashv)$ is replaced by the operation $((q', \mathbf{R}), \dashv, \mathbf{L}) \in \delta_B(q, \dashv)$. To complete the simulation of the right jump, we introduce the following additional transitions for all $j, k, l \in \{0, 1, \ldots, n\}$ and i < n:

$$\delta_B((q', \mathbf{R}), (a, i, j, k, l)) = \{((q', \mathbf{R}), (a, i+1, j, k, l), \mathbf{L})\},\\\delta_B((q', \mathbf{R}), \vdash) = \{(q', \vdash, \mathbf{R})\},\$$

that is, in state (q', \mathbf{R}) , B moves from right to left across the current row, increasing the index i for each symbol from Γ encountered. On reaching the left sentinel \vdash , state q' is entered and the head is moved to the field immediately to the right of this sentinel.

3. The cases that B sees one of the other sentinels, that is, the left sentinel \vdash , the top sentinel \top or the bottom sentinel \perp , are dealt with analogously using states of the form (q', L), (q', U), or (q', D).

Observe that A can enter each position on the border of a boundary picture only $|\Gamma|$ many times, as it must do so from its left (right, bottom, top) neighbor. Hence, it can execute at most $|\Gamma|$ many right (left, up, down) jumps within any row or column, which shows that the extended symbols of the form (a, i, j, k, l) of B suffice to correctly replace jumps by row or column sweeps. It follows that L(B) = L(A) holds, and we see from the definition of B that B is deterministic, if A is.

Thus, we see that for (deterministic and nondeterministic) Sgraffito automata, the extension by cyclic move operations does not affect the class of accepted picture languages. It may, however, simplify Sgraffito automata for particular picture languages, allowing simpler descriptions.

4. Cyclically extended det-2D-2W-ORWW-Automata

Here, we generalize the det-2D-2W-ORWW-automaton by extending its move-right and move-down steps into cyclic operations, called jumps, similar to the way in which we generalized the Sgraffito automaton above. Then we compare the resulting automaton to the det-2D-x2W-ORWW-automaton, and we study the question of whether the number of jumps that are allowed on any row or column yields an increasing hierarchy of language classes for cyclically extended det-2D-2W-ORWW-automata.

4.1. Definition and example

While the Sgraffito automaton replaces the current symbol by a smaller symbol in each step, which restricts the number of operations that it can perform on a picture of dimension (m, n) to $O(m \cdot n)$, a cyclically extended det-2D-2W-ORWW-automaton could get stuck on a row (a column) of a given picture by just keeping on executing move-right steps and right jumps (move-down steps and down jumps). Thus, we must be careful to define the cyclic extension of the det-2D-2W-ORWW-automaton in such a way that this cannot happen.

Definition 4.1. Let $d \ge 1$ be a constant. A *d*-cyclic deterministic two-dimensional two-way ordered restarting automaton, a *d*-cyc-2D-2W-ORWW-automaton for short, is given through a 6-tuple $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$, where all components are defined as for a det-2D-2W-ORWW-automaton. However, the move-right and move-down steps are cyclically extended as follows, where $P \in \Gamma^{m,n}$ denotes the current picture:

- 1. As long as the window does neither contain the right border marker nor the bottom marker, that is, if the window is at position (i, j) for some i < m and j < n, move-right and move-down steps can be used freely.
- 2. When the window contains the right border marker, then a cyclic move-right step, called a *right jump*, shifts the window to the beginning of the current row, that is, if the central position of the window is on the last field P(i, n) of row *i* for some $i \leq m$, then it is now placed on the first field P(i, 1) of row *i*.
- 3. When the window contains the bottom marker, then a cyclic move-down step, called *down jump*, shifts the window to the top of the current column, that is, if the central position of the window is on the bottommost field P(m, j) of column j for some $j \leq n$, then it is now placed on the topmost field P(1, j) of column j.
- 4. In any cycle, as soon as *M* executes a right (down) jump, then for the rest of this cycle, it cannot execute any down (right) jump.
- 5. Finally, during a cycle, M can execute at most d right jumps (down jumps) on any given row (column).

The conditions given in (4) and (5) above look quite technical. However, they can easily be ensured by structuring the set of states Q in a specific way as explained in the following remark.

Remark 4.2. In order to ensure conditions (4) and (5) of Definition 4.1, we can take a set of states Q for M that has the form

$$Q = \{(p, \alpha, c) \mid p \in \{p_0, p_1, \dots, p_m\}, \alpha \in \{\Box, \rightarrow, \downarrow\}, 0 \le c \le d\},\$$

where

- a state of the form $(p, \Box, 0)$ indicates that in the current cycle, M has not yet executed any right or down jump,
- a state of the form (p, \rightarrow, c) indicates that in the current cycle, M has already executed c right jumps on the current row, and
- a state of the form (p, \downarrow, c) indicates that in the current cycle, M has already executed c down jumps on the current column.

Accordingly, the initial state would then be $q_0 = (p_0, \Box, 0)$. A right jump can only be executed in a state of the form $(p, \Box, 0)$ or (p, \rightarrow, c) , where c < d, and through this step a state of the form $(p', \rightarrow, 1)$, respectively, $(p', \rightarrow, c+1)$, is reached. Analogously, a down jump can only be executed in a state of the form $(p, \Box, 0)$ or (p, \downarrow, c) , where c < d, and through this step a state of the form $(p', \downarrow, c+1)$, is reached.

Finally a move-down step from a state of the form (p, \rightarrow, c) yields a state of the form $(p', \rightarrow, 0)$, and a move-right step from a state of the form (p, \downarrow, c) yields a state of the form $(p', \downarrow, 0)$.

The following example illustrates the expressive power of the right jump operation in the special case of one-row pictures, that is, strings.

Example 4.3. Let $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$ be the 1-cyc-2D-2W-ORWW-automaton over $\Sigma = \{a, b, \$\}$ and $\Gamma = \Sigma \cup \Sigma_1 \cup \Sigma_2$, where $\Sigma_1 = \{a_1, b_1\}$ and $\Sigma_2 = \{a_2, b_2\}$, that only processes one-row pictures and that is described below. We will show that it accepts the language $L(M) = \{u\$u \mid u \in \Sigma_0^+\}$, where $\Sigma_0 = \{a, b\}$.

Let $w = u\$v \in \Sigma^+$ be the given input picture of dimension (1, n), where $u, v \in \Sigma_0^+$. In each cycle, M moves from left to right across row 1, checking the form of the current string in that row. Based on the form of this string, it then executes a rewrite operation, possibly after first performing a right jump. The main idea consists in comparing the two factors u and v from Σ_0^+ from left to right, using index 1 to mark the actual symbols to be compared and using index 2 to mark the prefixes of u and v that have already been checked successfully. Accordingly, M proceeds as follows.

- 1. If the current string is from the set $\Sigma_2^* \cdot \Sigma_0^+ \cdot \$ \cdot \Sigma_2^* \cdot \Sigma_0^+$, then *M* makes a right jump at the end of this row, it continues to move to the right, and then it replaces the leftmost symbol of the form $c \in \Sigma_0$ by the corresponding symbol $c_1 \in \Sigma_1$ and restarts.
- 2. If the current string is from the set $\Sigma_2^* \cdot \Sigma_1 \cdot \Sigma_0^* \cdot \$ \cdot \Sigma_2^* \cdot \Sigma_0^+$, then M moves the information on the letter $c_1 \in \Sigma_1$ to the right and compares it to the first letter of the second factor from Σ_0^* . If that factor starts with the letter c, then this occurrence of c is also rewritten into c_1 and M restarts; otherwise, M halts without accepting.
- 3. If the current string is from the set $\Sigma_2^* \cdot \Sigma_1 \cdot \Sigma_0^* \cdot \$ \cdot \Sigma_2^* \cdot \Sigma_1 \cdot \Sigma_0^*$, then the letters with index 1 are replaced by the corresponding letters with index 2. Accordingly, M executes a right jump at the end of this row, it continues to move right, and then it replaces the leftmost symbol of the form $c_1 \in \Sigma_1$ by the corresponding symbol $c_2 \in \Sigma_2$ and restarts.
- 4. If the current string is from the set $\Sigma_2^+ \cdot \Sigma_0^* \cdot \$ \cdot \Sigma_2^* \cdot \Sigma_1 \cdot \Sigma_0^*$, then *M* replaces the symbol $c_1 \in \Sigma_1$ by the corresponding symbol $c_2 \in \Sigma_2$ and restarts.

Finally, if the current string is from the set $\Sigma_2^+ \cdot \$ \cdot \Sigma_2^+$, that is, each letter $c \in \Sigma_0$ has been replaced by the corresponding auxiliary symbol $c_2 \in \Sigma_2$, then M accepts. In this case it follows that u and v coincide, that is, w has the form w = u\$u. Hence, $L(M) = \{u\$u \mid u \in \Sigma_0^+\}$, that is, it is the marked copy language which is known to be not even growing context-sensitive (see, e.g., [5]).

4.2. Cyclically extended det-2D-2W-ORWW-automata versus det-2D-x2W-ORWW-automata

By executing a move-down step immediately after a right jump, and by executing a move-right step immediately after a down jump, a 1-cyc-2D-2W-ORWW-automaton can simulate a det-2D-x2W-ORWW-automaton. Thus, we have the following inclusion, which is proper by Example 4.3, as det-2D-x2W-ORWW-automata do only accept string languages that are regular [13, 17].

Theorem 4.4. $\mathcal{L}(det-2D-x2W-ORWW) \subseteq \mathcal{L}(1-cyc-2D-2W-ORWW).$

Example 4.3 shows that 1-cyc-2D-2W-ORWW-automata can accept some string languages that are not accepted by det-2D-x2W-ORWW-automata. However, for picture languages with a fixed finite number of rows $m \geq 2$, the situation is quite different.

Lemma 4.5. Let $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$ be a 1-cyc-2D-2W-ORWW-automaton such that $L(M) \subseteq \Sigma^{m,*}$ for some integer $m \ge 2$. Then there exists a det-2D-x2W-ORWW-automaton M' such that L(M') = L(M).

Proof. Let $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$ be a 1-cyc-2D-2W-ORWW-automaton and let $m \ge 2$ be an integer such that $\ell_1(P) = m$ for all pictures $P \in L(M)$. We will simulate M by a det-2D-x2W-ORWW-automaton

 $M' = (Q', \Sigma, \Gamma', q'_0, \delta', >')$ that first encodes the given input picture $P \in \Sigma^{m,n}$ in the second row and that then simulates each cycle of M on P through a cycle that just reads the first two rows and records the rewrite of M in the corresponding field in row 2. To this end, we define M' as follows, where we distinguish between two phases of M'.

In the first phase M' encodes the contents of the given picture

$$P = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

in the second row. Accordingly, the set of states Q' of M' consists of a set Q_1 of states that are used during the encoding and a set Q_2 of states that are used during the simulation of M. The set Q_1 and the tape alphabet Γ' are defined through $Q_1 = \{q'_0, q'_1, \ldots, q'_{m-1}, q'_m\}$ and

$$\Gamma' = \Sigma \cup \{a' \mid a \in \Sigma\} \cup \left\{ \begin{bmatrix} a_1 \\ \vdots \\ a_i \end{bmatrix} \mid 1 \le i \le m, a_1, a_2, \dots, a_i \in \Gamma \right\},$$

that is, each letter from Γ' is either an input letter from Σ , a marked copy a' of it, or a column of 1 to m letters from Γ . The latter symbols, which will be denoted as $[\alpha]$ in what follows, will be used to encode the bottom parts of columns of P.

The transition function δ' is defined to perform the intended encoding as follows:

1. M' moves down the first column until it sees a symbol of the form $[\alpha]$ or the bottom marker \perp in position (3, 2) of its window. In the latter case it moves to the right until it finds a symbol of the form [b] or until it reaches the right endmaker \dashv , and it rewrites the left neighbor of that symbol, say $a \in \Sigma$, into the symbol [a]. In the other case, it moves to the right, scanning the row above the symbol $[\alpha]$ from left to right until it finds a symbol of the form $[\beta]$ or the right endmarker \dashv in position (2,3) of its window, and

it then replaces the left neighbor of that symbol, say $a \in \Sigma$, by the combined symbol $\begin{bmatrix} a \\ \gamma \end{bmatrix}$, where $[\gamma]$ is

the combined symbol below the symbol a. This process ends as soon as row 2 contains the code symbols $\begin{bmatrix} a_{2,1} \end{bmatrix} \begin{bmatrix} a_{2,2} \end{bmatrix} \begin{bmatrix} a_{2,n} \end{bmatrix}$

- $\begin{bmatrix} a_{2,1} \\ \cdots \\ a_{m,1} \end{bmatrix}, \begin{bmatrix} a_{2,2} \\ \cdots \\ a_{m,2} \end{bmatrix}, \cdots, \begin{bmatrix} a_{2,n} \\ \cdots \\ a_{m,n} \end{bmatrix}$
- 2. Then M^7 moves again to the second row and, proceeding from right to left, it adds the symbols from the first row to the code symbols in the second row. Also it replaces the symbols in row 1, from right to left, by their marked copies. After this the picture P has been rewritten into the form

$$P' = \begin{bmatrix} a'_{1,1} & a'_{1,2} & \cdots & a'_{1,n} \\ a_{1,1} \\ a_{2,1} \\ \cdots \\ a_{m,1} \end{bmatrix} \begin{bmatrix} a_{1,2} \\ a_{2,2} \\ \cdots \\ a_{m,2} \end{bmatrix} \begin{bmatrix} a_{1,n} \\ a_{2,n} \\ \cdots \\ a_{m,n} \end{bmatrix} \\ \begin{bmatrix} a_{3,1} \\ \cdots \\ a_{m,1} \end{bmatrix} \begin{bmatrix} a_{3,2} \\ \cdots \\ a_{m,2} \end{bmatrix} \begin{bmatrix} a_{3,n} \\ \cdots \\ a_{m,n} \end{bmatrix} \\ \begin{bmatrix} a_{3,n} \\ \cdots \\ a_{m,n} \end{bmatrix} \end{bmatrix}$$

When M' sees a symbol a' in position (2, 2) of its window after performing a rewrite/restart step, then it realizes that it is in phase 2 of the simulation. During the remaining cycles, it will only visit rows 1 and 2 of P'. In fact, starting at the top left corner, M' now simulates, simultaneously for all $i \in \{1, 2, ..., m\}$ and all states $q \in Q$, the computation of M that starts in state q with the central position of its window on the first field of row i. Actually, apart from the computation of M that starts in the initial state q_0 with the window at the top left corner, all other computations are only simulated up to a certain point. Accordingly, the set of states Q_2 that M' uses for this part of the simulation is defined as

$$Q_2 = \{ (\langle q, i; q', j \rangle)_{q \in Q, 1 \le i \le m} \mid q' \in Q \cup \{+, -\} \text{ and } j \in \{1, 2, \dots, m\} \},\$$

that is, each state from Q_2 is a sequence of four-tuples $\langle q, i; q', j \rangle \in Q \times \{1, 2, \dots, m\} \times (Q \cup \{+, -\}) \times \{1, 2, \dots, m\}$ such that there is one such tuple for each $q \in Q$ and each $i \in \{1, 2, \dots, m\}$. These states will be used as follows. If M' is in state $(\langle q, i; q', j \rangle)_{q \in Q, 1 \leq i \leq m}$ while scanning cell (2, k) for some $k \geq 1$, then, for each $q \in Q$ and each $i \in \{1, 2, \dots, m\}$, if M starts in state q scanning the first symbol in row i, then it will eventually reach cell (j, k) while being in state q', if $q' \in Q$. If q' = +, then the above state means that M, starting in state q scanning the first symbol in row i, executes a rewrite/restart step or an accept step before reaching column k, and if q' = -, then the above state means that M has already performed a down jump before reaching column k.

For the simulation, M' proceeds as follows. The initial state q'_0 is identified with the state $(\langle q, i; q, i \rangle)_{q \in Q, 1 \leq i \leq m}$ expressing the situation that, for all states $q \in Q$ and all $i \in \{1, 2, \ldots, m\}$, we are looking at all computations of M that start in state q with visiting the first field of row i. Now while moving its window to the right along the first row of P', M' sees the contents of each column of P in the bottom row of its window. Hence, it can simulate each one of the above partial computations of M. For each $q \in Q$ and each $i \in \{1, 2, \ldots, m\}, q \neq q_0$ or i > 1, this is done as follows. Assume that the four-tuple $\langle q, i; q', j \rangle$ is the component of the current state of M'that corresponds to q and i. Thus, the computation of M that starts in state q while scanning the first symbol of row i eventually reaches the jth symbol in the current column with M being in state q'.

- If in state q' with its window on the *j*th item of the current column, M would perform $l, 0 \le l \le m j$, move-down steps followed by a move-right step, which would take M into state p, then in the above four-tuple the state q' is updated to p and the current value of j is updated to j + l.
- If in state q' with its window on the *j*th item of the current column, M would perform $l, 0 \le l \le m j$, move-down steps that are followed by a rewrite/restart step or an accept step, then q' is replaced by the symbol +.
- If in state q' with its window on the *j*th item of the current column, M would make m j move-down steps that are followed by a down jump, then q' is replaced by the symbol -.

In addition, the computation of M that starts in state q_0 on row 1 is simulated. Let $\langle q_0, 1; q', j \rangle$ be the corresponding four-tuple of the current state of M', that is, the computation of M on the given input picture P eventually reaches the *j*th symbol in the current column with M being in state q'.

- If in state q' with its window on the *j*th item of the current column, M would perform some move-down steps (including possibly a down jump) to the *l*-th row followed by a move-right step, which would take M into state p, then in the four-tuple above the state q' is replaced by p and the value of j is updated to l.
- If in state q' with its window on the *j*th item of the current column, M would execute a number of move-down steps (including possibly a down jump) that are followed by a rewrite/restart step, then M' moves to row 2, performs the corresponding rewrite operation on the code symbol in row 2, and restarts.
- If in state q' with its window on the *j*th item of the current column, M would execute a number of move-down steps (possibly including a down jump) that are followed by an accept step, then M' simply accepts.

It remains the case that, starting in state q_0 on row 1, M will eventually execute a right jump at the end of some row j. In this case, the state s that M' enters on reaching the last item on row 1 will contain the four-tuple $\langle q_0, 1; \hat{q}, j \rangle$ such that $\delta(\hat{q}, C) = (q, \mathbb{R})$ for some state $q \in Q$, where C is the current subjecture of dimension (3, 3) with the central position at field (j, n). Thus, after this right jump the computation of M will continue with Mbeing in state q scanning the first symbol in row j. Let $\langle q, j; p, t \rangle$ be the component of s that corresponds to qand j, that is, it corresponds to the aforementioned computation of M. We now have to distinguish several cases.

- If t = +, then the computation of M that starts with state q on the first field of row j ends with a rewrite/restart step or an accept step. Accordingly, M' executes an extended move-right step and simply simulates this computation while scanning row 2 from left to right.
- If $t \ge j$, and if M would perform a number of move-down steps followed by a rewrite/restart step or an accept step when starting from state p with its window on the last field of row t, then again M' executes an extended move-right step and simply simulates this computation while scanning row 2 from left to right.
- If t > j, and if M would perform $l \ge 0$ move-down steps followed by a right jump entering state p' when starting from state p with its window on the last field of row t, then this consideration is repeated with the four-tuple $\langle p', t+l; p'', t' \rangle$, where this is the unique four-tuple in the current state s of M' that corresponds to the state p' and the integer t+l.

Observe that the case t = - expresses the fact that the computation of M that starts with state q on the first field of row j contains a down jump. However, as we are dealing with a cycle of M that contains a right jump, no down jump must occur in this cycle, which means that the case t = - need not be considered here. Thus, we see that M' does indeed simulate a cycle (or a tail) of the computation of M by first scanning the first row from left to right and by then executing an extended move-right step after which, scanning row 2 from left to right, it executes that part of this cycle (or tail) of the computation of M that follows after the last right jump. \Box

Note that the construction used in the proof of Lemma 4.5 easily extends to the case that M may execute up to d right jumps on any row for some fixed integer $d \ge 1$. Also it is easily seen that it would suffice that all pictures of the language L(M) have at least two and at most m rows for a fixed bound $m \ge 2$. Further, by symmetry, we can also bound the number of columns instead of the number of rows. Accordingly, we obtain the following general result, where a picture language $L \subseteq \Sigma^{*,*}$ is called *bounded* if there exists an integer $m \ge 2$ such that, for each picture $P \in L$, $2 \le \ell_1(P) \le m$ holds, or such that, for each picture $P \in L$, $2 \le \ell_2(P) \le m$ holds.

Theorem 4.6. Let $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$ be a d-cyc-2D-2W-ORWW-automaton for some integer $d \ge 1$ such that the language $L(M) \subseteq \Sigma^{*,*}$ is bounded. Then there exists a det-2D-x2W-ORWW-automaton M' such that L(M') = L(M).

However, the following question remains open.

Open problem 1. For unbounded picture languages that do not contain single-row or single-column pictures, are det-2D-x2W-ORWW-automata equivalent to 1-cyc-2D-2W-ORWW-automata with respect to their expressive power?

4.3. The number of jumps allowed does not induce a hierarchy

By cyc-2D-2W-ORWW we denote the class of det-2D-2W-ORWW-automata that are *d*-cyclic for some $d \ge 1$. Obviously, the parameter *d* induces a hierarchy of classes of picture languages:

246

where the proper inclusion $\mathcal{L}(\text{det-2D-x2W-ORWW}) \subseteq \mathcal{L}(1\text{-cyc-2D-2W-ORWW})$ has been shown using one-row pictures (Example 4.3), while for bounded picture languages, all these classes coincide. Concerning the general case we have the following additional result.

Theorem 4.7. For each $d \geq 3$ and each d-cyc-2D-2W-ORWW-automaton M, there exists a 2-cyc-2D-2W-ORWW-automaton M' such that L(M') = L(M).

Proof. Let $d \geq 3$ and let $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$ be a d-cyc-2D-2W-ORWW-automaton. We will simulate M by a 2-cyc-2D-2W-ORWW-automaton $M' = (Q', \Sigma, \Gamma, q'_0, \delta', >)$. On each row (or column) of a given input picture $P \in \Sigma^{m,n}$, M' will first behave just like M until the first right (or down) jump is executed. Thereafter, M' will simulate. simultaneously for all states q of M, the computation of M that starts in state q with its window on the first field of the current row (or column). This simulation contains all those computations that M would execute on the current row (or column) after each of its up to d right (or down) jumps. Hence, as in the proof of Lemma 4.5, M' can then determine the last right (or down) jump that M will execute on the current row (or column), execute this step, and continue with the simulation of M on that row (or column), which will not lead to any further right (down) jump on that row (column).

For realizing this simulation, the set of states of M' is chosen as

$$\begin{array}{lll} Q' &=& \{(q,i) \mid q \in Q, i \in \{1,2,3\}\} \cup \\ && \{(q^{(\mu)}, \langle p, p' \rangle_{p \in Q}) \mid q \in Q, p' \in Q \cup \{+\}, \mu \in \{\mathrm{R},\mathrm{D}\}\}, \end{array}$$

and $q'_0 = (q_0, 1)$. For $q \in Q$, the states of the form (q, 1) will be used to indicate that a part of a computation of M is simulated, where M is in state q, those of the form (q, 2) will be used to indicate in addition that it has been detected that the current computation of M does not involve another right jump on the current row, and those of the form (q,3) will be used to indicate that it has been detected that the current computation of M does not involve another down jump in the current column. Further, the states of the form $(q^{(R)}, \langle p, p' \rangle_{v \in O})$ will be used to simulate all computations of M that start at the first field of the current row in any state $p \in Q$ as long as M performs move-right steps in that computation. In that case p' is the corresponding actual state of M, while p' = + expresses the fact that the corresponding computation of M has encountered a rewrite/restart step, a move-down step, or an accept step. Analogously, the states of the form $(q^{(D)}, \langle p, p' \rangle_{p \in Q})$ will be used to simulate all computations of M that start on the first field of the current column in any state $p \in Q$ as long as M performs move-down steps in that computation. In that case p' is the corresponding actual state of M, while p = + expresses the fact that the corresponding computation of M has encountered a rewrite/restart step, a move-right step, or an accept step.

The transition function δ' is defined as follows, where C denotes the current window content:

- 1. If $\delta(q,C) = (q_1, \mathbb{R})$ and $C(2,3) \neq \exists$, then $\delta'((q,1),C) = ((q_1,1),\mathbb{R}), \ \delta'((q,2),C) = ((q_1,2),\mathbb{R}),$ and $\delta'((q,3),C) = ((q_1,1),R).$
- 2. If $\delta(q,C) = (q_1,D)$ and $C(3,2) \neq \bot$, then $\delta'((q,1),C) = ((q_1,1),D), \ \delta'((q,2),C) = ((q_1,1),D)$, and $\delta'((q,3),C) = ((q_1,3),D).$
- 3. If $\delta(q, C) \in \Gamma \cup \{\text{Accept}\}$, then $\delta'((q, 1), C) = \delta(q, C)$, $\delta'((q, 2), C) = \delta(q, C)$, and $\delta'((q, 3), C) = \delta(q, C)$.
- 4. If $\delta(q,C) = (q_1,R)$ and $C(2,3) = \dashv$, that is, M performs a right jump, then $\delta'((q,1),C) =$
- $((q_1^{(R)}, \langle p, p \rangle_{p \in Q}), R)$, that is, M' simulates the right jump of M by entering the state $(q_1^{(R)}, \langle p, p \rangle_{p \in Q})$. 5. If $\delta(q, C) = (q_1, D)$ and $C(3, 2) = \bot$, that is, M performs a down jump, then $\delta'((q, 1), C) =$ $((q_1^{(D)}, \langle p, p \rangle_{p \in Q}), D)$, that is, M' simulates the down jump of M by entering the state $(q_1^{(D)}, \langle p, p \rangle_{p \in Q})$.
- 6. In a state of the form $(q^{(R)}, \langle p, p' \rangle_{p \in Q})$ with its window over the k-th field of the current row, M' will simulate the computation of M that starts in state q at the current position and, for each $p' \in Q$, also the computation of M that starts in state p' at the current position.

If the current computation of M that is being simulated in the first component of the state of M' leads to a rewrite/restart step, a move-down step, or an accept step without encountering another right jump,

then M' simply continues with the simulation of that computation, forgetting about the other simulated partial computations of M (see (a) and (b)):

- (a) If $\delta(q, C) \in \Gamma \cup \{\text{Accept}\}\$, then $\delta'((q^{(R)}, \langle p, p' \rangle_{p \in Q}), C) = \delta(q, C)$.
- (b) If $\delta(q, C) = (q_1, D)$, then $\delta'((q^{(R)}, \langle p, p' \rangle_{p \in Q}), C) = ((q_1, 1), D)$.
- (c) If $\delta(q, C) = (q_1, \mathbb{R})$ and $C(2, 3) \neq \exists$, then

$$\delta'((q^{(\mathbf{R})}, \langle p, p' \rangle_{p \in Q}), C) = ((q_1^{(\mathbf{R})}, \langle p, \tilde{p} \rangle_{p \in Q}), \mathbf{R})_{q \in Q}$$

where, for each $p' \in Q$, the corresponding \tilde{p} is determined as follows:

- i. If $\delta(p', C) \in \Gamma \cup \{\text{Accept}\} \cup (Q \times \{D\}), \text{ then } \tilde{p} = +.$
- ii. If $\delta(p', C) = (p_1, \mathbf{R})$, then $\tilde{p} = p_1$.

It remains to consider the case that the current computation of M leads to another right jump (or down jump) on the current row (or column). Assume that M' reaches the last field on the current row in state $s = (q^{(R)}, \langle p, p' \rangle_{p \in Q})$, and that $\delta(q, C) = (q_1, R)$, where $C(2, 3) = \dashv$. Then the pair (q_1, q'_1) of s contains the precomputed information on the computation that M will perform after this right jump on the current row.

- (a) If $q'_1 = +$, then this computation leads to a rewrite/restart step, to an accept step, or to a move-down step without an additional right jump on the current row. Accordingly, we take $\delta'(s, C) = ((q_1, 2), R)$, that is, M' executes a right jump entering state $(q_1, 2)$.
- (b) If $q'_1 = q_2 \in Q$ and $\delta(q_2, C) \in \Gamma \cup \{\text{Accept}\}\)$, then we take $\delta'(s, C) = \delta(q_2, C)\)$, that is, M' simply executes the operation that M will perform when it reaches the last field of the current row again.
- (c) If $q'_1 = q_2 \in Q$ and $\delta(q_2, C) = (q_3, D)$, then $\delta'(s, C) = ((q_3, 1), D)$, that is, also in this case M' simply executes the operation that M will perform when it reaches the last field of the current row again.
- (d) If $q'_1 = q_2 \in Q$ and $\delta(q_2, C) = (q_3, \mathbf{R})$, that is, M will perform another right jump on the current row, then we repeat the above considerations with the corresponding pair (q_3, q'_3) from s.
- 7. In a state of the form $(q^{(D)}, \langle p, p' \rangle_{p \in Q})$ the computation continues in a way that is symmetric to 6.

It follows that M' simulates the up to d right jumps of M on the current row through at most two right jumps of its own. The case that the current computation of M leads to another down jump on the current column is dealt with symmetrically.

We see that M' is just 2-cyclic and that it accepts the same picture language as M, since it simulates the computations of M. This completes the proof of Theorem 4.7.

Thus, we obtain the following result for general picture languages showing that the above hierarchy consists of at most three levels.

Corollary 4.8. $\mathcal{L}(2$ -cyc-2D-2W-ORWW) = $\mathcal{L}(cyc-2D-2W-ORWW)$.

However, it remains open whether the second jump operation is actually necessary.

Open problem 2. Are 2-cyc-2D-2W-ORWW-automata really more expressive than 1-cyc-2D-2W-ORWW-automata?

The proof of Theorem 4.7 can easily be adjusted to show the following result, where a *d*-cyc-ORWWautomaton is just a *d*-cyc-2D-2W-ORWW-automaton that only works on one-row pictures, that is, strings. Just notice that each computation of a *d*-cyc-2D-2W-ORWW-automaton on a one-row picture always begins at the leftmost field of that row, which means that the first right jump in the simulation presented in the proof of Theorem 4.7 is not needed.

Corollary 4.9. For each $d \ge 2$ and each d-cyc-ORWW-automaton M, there exists a 1-cyc-ORWW-automaton M' such that L(M') = L(M).

Thus, for strings we have the following hierarchy:

$$\text{REG} = \mathcal{L}(\text{det-ORWW}) \subseteq \mathcal{L}(1\text{-cyc-ORWW}) = \mathcal{L}(\text{cyc-ORWW}),$$

where the properness of the inclusion $\mathcal{L}(\text{det-ORWW}) \subsetneq \mathcal{L}(1\text{-cyc-ORWW})$ is an immediate consequence of Example 4.3.

4.4. Closure under inverse homogeneous morphisms

It is shown in [17] that the classes of picture languages $\mathcal{L}(det-2D-2W-ORWW)$ and $\mathcal{L}(det-2D-x2W-ORWW)$ are both closed under the operations of transposition, complementation, union, and intersection. For $\mathcal{L}(1-cyc-2D-2W-ORWW)$ and $\mathcal{L}(2-cyc-2D-2W-ORWW)$, closure under transposition follows immediately from the fact that these types of automata are symmetric with respect to row and column operations, closure under complementation follows from the fact that these types of automata are deterministic, and closure under union and intersection can be proved by using the same arguments as in the corresponding proof for det-2D-x2W-ORWW-automata in [17].

Further, it is shown in [15] (see also [16]) that the class $\mathcal{L}(\text{det-2D-2W-ORWW})$ is closed under inverse homogeneous morphisms. Here a homogeneous morphism φ from $\Delta^{*,*} \to \Sigma^{*,*}$ is defined by two integers $b, h \ge 1$ and a mapping φ that associates, with each letter $a \in \Delta$, a picture $\varphi(a) \in \Sigma^{h,b}$ of dimension (h,b). Then φ extends naturally to a morphism $\varphi : \Delta^{*,*} \to \Sigma^{*,*}$ that maps a picture $P \in \Delta^{m,n}$ into a picture $\varphi(P) \in \Sigma^{m \cdot h, n \cdot b}$ (see, *e.g.*, [15]). However, the class $\mathcal{L}(\text{det-2D-x2W-ORWW})$ is not closed under inverse homogeneous morphisms [16]. Concerning the cyclically extended det-2D-2W-ORWW-automata, we have the following positive result.

Theorem 4.10. The class of picture languages $\mathcal{L}(2$ -cyc-2D-2W-ORWW) is closed under inverse homogeneous morphisms, that is, if $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$ is a 2-cyc-2D-2W-ORWW-automaton and $\varphi : \Delta^{*,*} \to \Sigma^{*,*}$ is a homogeneous morphism, then there exists a 2-cyc-2D-2W-ORWW-automaton M' which accepts $P \in \Delta^{*,*}$ if and only if M accepts $\varphi(P)$.

Proof. Let $M = (Q, \Sigma, \Gamma, q_0, \delta, >)$ be a 2-cyc-2D-2W-ORWW-automaton and $\varphi : \Delta^{*,*} \to \Sigma^{*,*}$ be a homogeneous morphism that maps each letter $a \in \Delta$ to a picture $\varphi(a) \in \Sigma^{h,b}$. We first describe a cyc-2D-2W-ORWWautomaton $M' = (Q', \Delta, \Gamma', q'_0, \delta', >')$ which simulates the computation of M on input $\varphi(P)$. We take $\Gamma' = \Delta \cup \Gamma^{h,b}$, where a symbol $B \in \Gamma^{h,b}$ will be used to represent the contents of a block of dimension (h, b) of the tape of M.

Given a picture $P = [a_{i,j}]_{1 \le i \le m, 1 \le j \le n}$, M' first rewrites P into a compressed representation of the picture $\varphi(P)$ by replacing each letter $a_{i,j}$ by the symbol $\varphi(a_{i,j}) \in \Gamma^{h,b}$. This is easily achieved by proceeding row by row, from the bottom to the top, processing each row from right to left. Once the letter $a_{1,1}$ has been rewritten by the symbol $\varphi(a_{1,1}) \in \Gamma^{h,b}$, then M' starts to simulate the computation of M on the picture $\varphi(P)$. While the window of M only contains a subpicture of dimension (3,3) of $\varphi(P)$, the window of M' contains a subpicture of dimension (3,3) of $\varphi(P)$, the window of M' contains a subpicture of dimension (3,3) of $\varphi(P)$, the window of M' contains a subpicture of dimension (h,b), ending when M leaves this block or when M ends the current cycle inside this block. When M leaves a block, the automaton M' must perform a corresponding block, the state of M' must record the position at the block perimeter in which M enters it. Since a cyc-2D-2W-ORWW-automaton moves its head only to the right and down, there are h + b - 1 possible such positions in the first row and the first column of each block. Accordingly, we take $Q' = Q \times \{1, 2, \ldots, h + b - 1\}$ and $q'_0 = (q_0, 1)$, assuming that the number 1 represents the top-left corner of a block. The partial ordering >' is defined in accordance with > by taking the transitive closure of the relation that is defined as follows:

$$\begin{array}{ll} a & >' & B & \text{ for all } a \in \Gamma \text{ and } B \in \Gamma^{h,b}, \\ B & >' & B' & \text{ if } B, B' \in \Gamma^{h,b} \text{ and there exist } i \in \{1,2,\ldots,h\}, j \in \{1,2,\ldots,b\}: \\ & B(i,j) > B'(i,j) \text{ and } B(r,s) = B'(r,s) \text{ for all pairs } (r,s) \neq (i,j). \end{array}$$

It is easily seen that M' accepts on input P iff M accepts on input $\varphi(P)$, that is, $L(M') = \varphi^{-1}(L(M))$.

As M can execute up to two right (or down) jumps on any row (or column) of $\varphi(P)$, M' may execute up to 2h (2b) right (or down) jumps on any row (or column) of P. Thus, M' is a k-cyc-2D-2W-ORWW-automaton for $\varphi^{-1}(L(M))$, where $k = 2 \cdot \max\{h, b\}$. From Theorem 4.7 we now obtain a 2-cyc-2D-2W-ORWW-automaton for the language $\varphi^{-1}(L(M))$, which completes the proof of Theorem 4.10.

However, the following question remains open.

Open problem 3. Is the class of picture languages $\mathcal{L}(1$ -cyc-2D-2W-ORWW) closed under inverse homogeneous morphisms?

5. Concluding Remarks

We have introduced cyclic move operations, called jumps, for Sgraffito automata and for det-2D-2W-ORWWautomata. While for the former, these operations do not increase the expressive power, we have seen that for the latter, already a single jump operation (per column or row) increases the power of the det-2D-2W-ORWWautomaton beyond that of the det-2D-x2W-ORWW-automaton, at least for one-row pictures. For bounded picture languages, det-2D-x2W-ORWW-automata have the same expressive power as cyclically extended det-2D-2W-ORWW-automata, but for general picture languages, it remains open whether 1-cyc-2D-2W-ORWWautomata are really more expressive than det-2D-x2W-ORWW-automata. Further, for picture languages, two jump operations (per column or row) are already sufficient to simulate any bounded number of jump operations, and for string languages, already a single jump operation suffices. However, it also remains open whether the second jump operation is really necessary for picture languages in general. Finally, we have seen that the class of picture languages $\mathcal{L}(2\text{-cyc-2D-2W-ORWW})$ is closed under inverse homogeneous morphisms, while for the class $\mathcal{L}(1\text{-cyc-2D-2W-ORWW})$, it remains open whether it is closed under this operation.

References

- A. Anselmo, D. Giammarresi and M. Madonia, From determinism to non-determinism in recognizable two-dimensional languages, in *Proc. of DLT 2007*, edited by T. Harju, J. Karhumäki and A. Lepistö. Vol. 4588 of *Lect. Notes Comput. Sci.* Springer, Heidelberg (2007) 36–47.
- [2] A. Anselmo, D. Giammarresi and M. Madonia, Deterministic and unambiguous families within recognizable two-dimensional languages. Fund. Inform. 98 (2010) 143–166.
- [3] M. Blum and C. Hewitt, Automata on a 2-dimensional tape, in Proc. of SWAT 1967. IEEE Computer Society, Washington, DC, USA (1967) 155–160.
- [4] B. Borchert and K. Reinhardt, Deterministically and sudoku-deterministically recognizable picture languages, in Preproc. of LATA 2007, Report 35/07, edited by R. Loos, S.Z. Fazekas and C. Martín-Vide. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona (2007) 175–186.
- [5] G. Buntrock and F. Otto, Growing context-sensitive languages and Church-Rosser languages. Inform. Comput. 141 (1998) 1–36.
- [6] H. Fernau, M. Paramasivan, M.L. Schmid and D.G. Thomas, Scanning pictures the boustrophedon way, in Proc. of 17th Intern. Workshop IWCIA – Combinatorial Image Analysis, edited by R.P. Berneva, B.B. Bhattacharya and V.E. Brimkov. Vol. of 9448 Lect. Notes Comput. Sci. Springer, Heidelberg (2015) 202–216.
- [7] D. Giammarresi and A. Restivo, Recognizable picture languages. Int. J. Pattern Recogn. Artif. Intell. 6 (1992) 241-256.
- [8] D. Giammarresi and A. Restivo, Two-dimensional languages. Vol. 3 of Handbook of Formal Languages, edited by G. Rozenberg and A. Salomaa. Springer, New York, NY, USA (1997) 215–267.
- [9] F.C. Hennie, One-tape, off-line Turing machine computations. Inform. Cont. 8 (1965) 553-578.
- [10] P. Jančar, F. Mráz, M. Plátek and J. Vogel, Restarting automata, in Proc. of FCT'95, edited by H. Reichel. Vol. 965 of Lect. Notes Comput. Sci. Springer, Heidelberg (1995) 283–292.
- [11] J. Kari and C. Moore, New results on alternating and non-deterministic two-dimensional finite-state automata, in *Proc. of STACS 2001*, edited by A. Ferreira and H. Reichel. Vol. 2010 of *Lect. Notes Comput. Sci.* Springer, Heidelberg (2001) 396–406.
 [12] V. Li and C. Moore, New results on alternating and non-deterministic two-dimensional finite-state automata, in *Proc. of STACS 2001*, edited by A. Ferreira and H. Reichel. Vol. 2010 of *Lect. Notes Comput. Sci.* Springer, Heidelberg (2001) 396–406.
- [12] K. Lindgren, C. Moore and M. Nordahl, Complexity of two-dimensional patterns. J. Stat. Phys. 91 (1998) 909–951.
- [13] F. Mráz and F. Otto, Extended two-way ordered restarting automata for picture languages, in Proc. of LATA 2014, edited by A.-H. Dediu, C. Martín-Vide, J.-L. Sierra-Rodríguez, and B. Truthe. Vol. 8370 of Lect. Notes Comput. Sci. Springer, Heidelberg (2014) 541–552.
- [14] F. Mráz and F. Otto, Ordered restarting automata for picture languages, in *Prof. of SOFSEM 2014*, edited by V. Geffert, B. Preneel, B. Rovan, J. Štuller and A. Min Tjoa. Vol. 8327 of *Lect. Notes Comput. Sci.* Springer, Heidelberg (2014) 431–442.

250

- [15] F. Mráz, F. Otto and D. Průša, On a class of rational functions for pictures, in Proc. of Seventh Workshop on Non-Classical Models of Automata and Applications (NCMA 2015), edited by R. Freund, M. Holzer, N. Moreira and R. Reis. Vol. 318 of books@ocg.at. Oesterreichische Computer Gesellschaft, Wien (2015) 159–176.
- [16] F. Mráz, F. Otto and D. Průša, Some classes of rational functions for pictures. RAIRO: ITA 50 (2016) 351-369.
- [17] F. Otto and F. Mráz, Deterministic ordered restarting automata for picture languages. Acta Inform. 52 (2015) 593-623.
- [18] F. Otto and F. Mráz, Cyclically extended variants of Sgraffito and restarting automata for picture languages, in Proc. of Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016), edited by H. Bordihn, R. Freund, B. Nagy and G. Vaszil. Vol. 321 of books@ocg.at. Oesterreichische Computer Gesellschaft, Wien (2016) 259–273.
- [19] D. Průša and F. Mráz, Restarting tiling automata, in Proc. of CIAA 2012, edited by N. Moreira and R. Reis. Vol. 7381 of Lect. Notes Comput. Sci. Springer, Heidelberg (2012) 289–300.
- [20] D. Průša and F. Mráz, Two-dimensional Sgraffito automata, in Proc. of DLT 2012, edited by H.C. Yen and O.H. Ibarra. Vol. 7410 of Lect. Notes Comput. Sci. Springer, Heidelberg (2012) 251–262.
- [21] D. Průša and F. Mráz, Restarting tiling automata. Int. J. Found. Comput. Sci. 24 (2013) 863-878.
- [22] D. Průša, F. Mráz and F. Otto, Comparing two-dimensional one-marker automata to Sgraffito automata, in Proc. of CIAA 2013, edited by S. Konstantinidis. Vol. 7982 of Lect. Notes Comput. Sci. Springer, Heidelberg (2013) 268–279.
- [23] D. Průša, F. Mráz and F. Otto, New results on deterministic Sgraffito automata, in Proc. of DLT 2013, edited by M.P. Béal and O. Carton. Vol. 7907 of Lect. Notes Comput. Sci. Springer, Heidelberg (2013) 409–419.
- [24] D. Průša, F. Mráz and F. Otto, Two-dimensional Sgraffito automata. RAIRO: ITA 48 (2014) 505-539.