TWO-DIMENSIONAL PATTERN LANGUAGES

Henning Fernau$^{(A)}$ Markus L. Schmid$^{(A)}$
K. G. Subramanian$^{(B)}$

$^{(A)}$Fachbereich 4 – Abteilung Informatikwissenschaften,
Universität Trier, D-54296 Trier, Germany
`{Fernau, MSchmid}@uni-trier.de`

$^{(B)}$School of Computer Sciences, Universiti Sains Malaysia, 11800 Penang, Malaysia
`kgsmani1948@yahoo.com`

Abstract
We introduce several classes of array languages obtained by generalising Angluin's pattern languages to the two-dimensional case. These classes of two-dimensional pattern languages are compared with respect to their expressive power and their closure properties are investigated.

1. Introduction

Several methods of generation of *two-dimensional languages* (also called *array languages* or *picture languages*) have been proposed in the literature, extending the techniques and results of formal string language theory. A picture is considered as a rectangular array of terminal symbols in the two-dimensional plane. Models based on grammars or automata as well as those based on theoretical properties of the string languages are well-known and have been extensively investigated. We refer the interested readers to books and surveys like the ones by Rosenfeld [11], Wang [14], Rosenfeld and Siromoney [12], Giammarresi and Restivo [6], or Morita [10]. For example, regular *string* languages (also known as recognizable string languages) can be characterized in terms of local languages and projections. Based on a similar idea, the class REC of recognizable *picture* languages (see Giammarresi and Restivo [5]) was proposed as a two dimensional counterpart of regular string languages. In this work, we attempt to generalise a class of string languages to the two-dimensional case, which also provides several desirable features and has therefore attracted considerable interest over the last three decades in the formal language theory community as well as in the learning theory community: Angluin's pattern languages (see [1]).

In this context, a *pattern* is a string over an alphabet $\{x_1, x_2, x_3, \ldots\}$ of *variables*, e. g., $\alpha := x_1\, x_1\, x_2\, x_2\, x_1$. For some finite alphabet Σ of *terminal symbols*, the *pattern language* described by α (with respect to Σ) is the set of all words over Σ that can be derived from α by uniformly substituting the variables in α by (non-empty) terminal words. For example, if $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, then $u :=$ `bcbbcbccaccabcb` and $v :=$ `ababababaabaab` are words of the pattern language given by α, since replacing x_1 by `bcb` and x_2 by `cca` turns α into u and replacing x_1 by `ab` and x_2 by `aba` turns α into v. On the other hand, the word `cabacababababa` is not a member of the pattern language of α.

One of the most notable features of pattern languages is that they have natural and compact human readable descriptors (or generators), namely the patterns. In particular, this advantage becomes evident when patterns are compared to other language descriptors as, e. g., grammars or automata, which are usually quite involved even though the language they describe is rather simple. Nevertheless, patterns can compete with common automata models and grammars in terms of expressive power and their practical relevance is demonstrated by the widespread use of so-called extended regular expressions with backreferences, which implicitly use the concept of patterns and are capable of defining all pattern languages.[1]

The main goal of this paper is to generalise the concept of patterns as language descriptors to the two-dimensional case, while preserving the desirable features of (one-dimensional) pattern languages, i. e., the simplicity and compactness of their descriptors. The work done so far on two-dimensional languages demonstrates that there are difficulties that seem to be symptomatic for the task of generalising a class of string languages to the two-dimensional case. Firstly, such a generalisation is usually accompanied with a substantial increase in complexity of the descriptors (e. g., when extending context-free or contextual grammars to the two-dimensional case (see Fernau et al. [2], Freund et al. [3])) and, secondly, there are often many competing and seemingly different ways to generalise a specific class of string languages, which all can be considered natural (e. g., it is still on debate what the appropriate two-dimensional counterpart of the class of regular languages might be (see Giammarresi et al. [7], Matz [9])). Our two-dimensional patterns, to be introduced in this work, are as simple and compact as their one-dimensional counterparts. Although there are several different possibilities of how these two-dimensional patterns can describe two-dimensional languages, one of these sticks out as the intuitively most natural one. Hence, the model of Angluin's pattern languages seems to be comparatively two-dimensional friendly.

Besides the conceptional contribution of this paper, we present a comparison between the expressive power of different classes of two-dimensional pattern languages and an investigation of their closure properties. We conclude the paper by outlining further research questions and possible extensions to the model of two-dimensional pattern languages. For reasons of space restrictions, most of the proofs have been moved to an appendix.

2. Preliminaries

In this section, we briefly recall the standard definitions and notations regarding one- and two-dimensional words and languages.

[1]In fact, these extended regular expressions with backreferences are nowadays a standard element of most text editors and programming languages (cf. Friedl [4]).

Let $\mathbb{N} := \{1, 2, 3, \ldots\}$ and let $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For a finite alphabet Σ, a *string* or *word* (*over* Σ) is a finite sequence of symbols from Σ, and ε stands for the *empty string*. The notation Σ^+ denotes the set of all nonempty strings over Σ, and $\Sigma^* := \Sigma^+ \cup \{\varepsilon\}$. For the *concatenation* of two strings w_1, w_2 we write $w_1 \cdot w_2$ or simply $w_1 w_2$. We say that a string $v \in \Sigma^*$ is a *factor* of a string $w \in \Sigma^*$ if there are $u_1, u_2 \in \Sigma^*$ such that $w = u_1 \cdot v \cdot u_2$. If u_1 or u_2 is the empty string, then v is a *prefix* (or a *suffix*, respectively) of w. The notation $|w|$ stands for the length of a string w.

A *two-dimensional word* (or *array*) *over* Σ is a tuple

$$W := ((a_{1,1}, a_{1,2}, \ldots, a_{1,n}), (a_{2,1}, a_{2,2}, \ldots, a_{2,n}), \ldots, (a_{m,1}, a_{m,2}, \ldots, a_{m,n})),$$

where $m, n \in \mathbb{N}$ and, for every i, $1 \leq i \leq m$, and j, $1 \leq j \leq n$, $a_{i,j} \in \Sigma$. We define the *number of columns* (or *width*) and *number of rows* (or *height*) of W by $|W|_c := n$ and $|W|_r := m$, respectively. The *empty array* is denoted by λ, i. e., $|\lambda|_c = |\lambda|_r = 0$. For the sake of convenience, we also denote W by $[a_{i,j}]_{m,n}$ or by a matrix in a more pictorial form. If we want to refer to the j^{th} symbol in row i of the array W, then we use $W[i, j] = a_{i,j}$. By Σ^{++}, we denote the set of all nonempty arrays over Σ, and $\Sigma^{**} := \Sigma^{++} \cup \{\lambda\}$. Every subset $L \subseteq \Sigma^{**}$ is an *array language*.

Let $W := [a_{i,j}]_{m,n}$ and $W' := [a'_{i,j}]_{m',n'}$ be two non-empty arrays over Σ. The *column concatenation* of W and W', denoted by $W \mathbin{⦶} W'$, is undefined if $m \neq m'$ and is the array

$$\begin{array}{cccccccc} a_{1,1} & a_{1,2} & \dots & a_{1,n} & b_{1,1} & b_{1,2} & \dots & b_{1,n'} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} & b_{2,1} & b_{2,2} & \dots & b_{2,n'} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} & b_{m',1} & b_{m',2} & \dots & b_{m',n'} \end{array}$$

otherwise. The *row concatenation* of W and W', denoted by $W \ominus W'$, is undefined if $n \neq n'$ and is the array

$$\begin{array}{cccc} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \\ b_{1,1} & b_{1,2} & \dots & b_{1,n'} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n'} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m',1} & b_{m',2} & \dots & b_{m',n'} \end{array}$$

otherwise. In order to denote that, e. g., $U \ominus V$ is undefined, we also write $U \ominus V = \texttt{undef}$.

Example 2.1 *Let*

$$W_1 := \begin{bmatrix} \mathtt{a} & \mathtt{b} & \mathtt{a} \\ \mathtt{b} & \mathtt{c} & \mathtt{a} \\ \mathtt{a} & \mathtt{b} & \mathtt{b} \end{bmatrix}, W_2 := \begin{bmatrix} \mathtt{b} & \mathtt{c} \\ \mathtt{b} & \mathtt{a} \\ \mathtt{c} & \mathtt{a} \end{bmatrix}, W_3 := \begin{bmatrix} \mathtt{a} & \mathtt{b} & \mathtt{c} \\ \mathtt{c} & \mathtt{b} & \mathtt{b} \end{bmatrix} \text{ and } W_4 := \begin{bmatrix} \mathtt{a} & \mathtt{a} \\ \mathtt{a} & \mathtt{b} \end{bmatrix}.$$

Then $W_1 \ominus W_2$ *and* $W_1 \mathbin{⦶} W_3 = \texttt{undef}$, *but*

$$W_1 \mathbin{⦶} W_2 = \begin{bmatrix} \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{c} \\ \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{b} & \mathtt{a} \\ \mathtt{a} & \mathtt{b} & \mathtt{b} & \mathtt{c} & \mathtt{a} \end{bmatrix} \text{ and } W_1 \ominus W_3 = \begin{bmatrix} \mathtt{a} & \mathtt{b} & \mathtt{a} \\ \mathtt{b} & \mathtt{c} & \mathtt{a} \\ \mathtt{a} & \mathtt{b} & \mathtt{b} \\ \mathtt{a} & \mathtt{b} & \mathtt{c} \\ \mathtt{c} & \mathtt{b} & \mathtt{b} \end{bmatrix}.$$

Next, we define some operations for array languages. The row and column concatenation for array languages L_1 and L_2 is defined by $L_1 \ominus L_2 := \{U \ominus V \mid U \in L_1, V \in L_2, U \ominus V \neq \mathtt{undef}\}$ and $L_1 \obar L_2 := \{U \obar V \mid U \in L_1, V \in L_2, U \obar V \neq \mathtt{undef}\}$, respectively. For an array language L and $k \in \mathbb{N}$, $L^{\ominus k}$ denotes the k-fold row concatenation of L, i. e., $L^{\ominus k} := L_1 \ominus L_2 \ominus \ldots \ominus L_k$, $L_i = L$, $1 \leq i \leq k$. The k-fold column concatenation, denoted by $L^{\obar k}$, is defined analogously. The *row* and *column concatenation closure* of an array language L is defined by $L^{\ominus *} := \bigcup_{k=1}^{\infty} L^{\ominus k}$ and $L^{\obar *} := \bigcup_{k=1}^{\infty} L^{\obar k}$, respectively. Obviously, the row and column concatenation closure of an array language correspond to the Kleene closure of a string language.

Now, we turn our attention to some geometric operations for arrays. The *transposition* of an array U, denoted by $U^{\mathtt{T}}$, is obtained by reflecting U along the main diagonal. The $\ominus$-*reflection* and $\obar$-*reflection* of U, denoted by $U^{\ominus\mathtt{R}}$ and $U^{\obar\mathtt{R}}$, respectively, are obtained by reflecting U along the horizontal and vertical axis, respectively. The *right turn* and *left turn* of U, denoted by $U^{\curvearrowright}$ and $U^{\curvearrowleft}$, respectively, is obtained by turning U through 90 degrees to the right and to the left, respectively. For example, if $U := \begin{bmatrix} \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{d} \\ \mathtt{e} & \mathtt{f} & \mathtt{g} & \mathtt{h} \end{bmatrix}$, then

$$U^{\mathtt{T}} = \begin{bmatrix} \mathtt{a} & \mathtt{e} \\ \mathtt{b} & \mathtt{f} \\ \mathtt{c} & \mathtt{g} \\ \mathtt{d} & \mathtt{h} \end{bmatrix}, U^{\ominus\mathtt{R}} = \begin{bmatrix} \mathtt{e} & \mathtt{f} & \mathtt{g} & \mathtt{h} \\ \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{d} \end{bmatrix}, U^{\obar\mathtt{R}} = \begin{bmatrix} \mathtt{d} & \mathtt{c} & \mathtt{b} & \mathtt{a} \\ \mathtt{h} & \mathtt{g} & \mathtt{f} & \mathtt{e} \end{bmatrix}, U^{\curvearrowright} = \begin{bmatrix} \mathtt{e} & \mathtt{a} \\ \mathtt{f} & \mathtt{b} \\ \mathtt{g} & \mathtt{c} \\ \mathtt{h} & \mathtt{d} \end{bmatrix}, U^{\curvearrowleft} = \begin{bmatrix} \mathtt{d} & \mathtt{h} \\ \mathtt{c} & \mathtt{g} \\ \mathtt{b} & \mathtt{f} \\ \mathtt{a} & \mathtt{e} \end{bmatrix}, (U^{\curvearrowright})^{\curvearrowright} = \begin{bmatrix} \mathtt{h} & \mathtt{g} & \mathtt{f} & \mathtt{e} \\ \mathtt{d} & \mathtt{c} & \mathtt{b} & \mathtt{a} \end{bmatrix}.$$

We address left and right turn also as *quarter-turns* below. Moreover, the twofold right turn (displayed right-most in the example above) is also known as a *half-turn.* All these operations for arrays are extended to array languages in the obvious way.

Next, we briefly summarise the concept of (one-dimensional) pattern languages as introduced in [1] by Angluin. Technically, the version of pattern languages used here are called *nonerasing terminal-free* pattern languages (for an overview of different versions of one-dimensional pattern languages, the reader is referred to [8] by Mateescu and Salomaa).

A (one-dimensional) *pattern* is a string over an alphabet $X := \{x_1, x_2, x_3, \ldots\}$ of *variables*, e. g., $\alpha := x_1\, x_1\, x_2\, x_2\, x_1$. In Section 1., we have seen an intuitive definition of the language described by a pattern α. This intuition can be formalised in an elegant way by using the concept of (*word*) *morphisms*, i. e., mappings $h : \Sigma_1^+ \to \Sigma_2^+$, which satisfy $h(u\, v) = h(u)\, h(v)$, for all $u, v \in \Sigma_1^+$. In this regard, for some finite alphabet Σ, the (one-dimensional) *pattern language* of α (with respect to Σ) is the set $L_{\Sigma}^{\mathtt{1D}}(\alpha) := \{h(\alpha) \mid h : X^+ \to \Sigma^+ \text{ is a morphism}\}$. An alternative, yet equivalent, way to define pattern languages is by means of factorisations. To this end, let $\alpha := y_1\, y_2 \ldots y_n$, $y_i \in X$, $1 \leq i \leq n$. Then $L_{\Sigma}^{\mathtt{1D}}(\alpha)$ is the set of all words $w \in \Sigma^+$ that have a *characteristic* factorisation for α, i. e., a factorisation $w = u_1\, u_2 \cdots u_n$, such that, for every i, $1 \leq i \leq j \leq n$, $y_i = y_j$ implies $u_i = u_j$. It can be easily seen, that these two definitions are equivalent. However, for the two-dimensional case, we shall see that a generalisation of these two approaches will lead to different versions of two-dimensional pattern languages. The class of all one-dimensional pattern languages over the alphabet Σ is denoted by $\mathcal{L}_{\Sigma}^{\mathtt{1D}}$. We recall the example pattern $\alpha = x_1\, x_1\, x_2\, x_2\, x_1$ and the words $u :=$ bcbbcbccaccabcb and $v :=$ ababababaabaab of Section 1. Since $h(\alpha) = u$ and $g(\alpha) = v$, where h and g are the morphisms induced by $h(x_1) :=$ bcb, $h(x_2) :=$ cca and $g(x_1) :=$ ab, $g(x_2) :=$ aba, we can conclude that $u, v \in L_{\Sigma}^{\mathtt{1D}}(\alpha)$, where $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

3. Two-Dimensional Pattern Languages

As already mentioned, this work deals with the task of generalising pattern languages from the one-dimensional to the two-dimensional case. In order to motivate our approach to solve this task, we first spent some effort on illustrating the general difficulties and obstacles that arise.

Abstractly speaking, a pattern language for a given pattern α is the collection of all elements that satisfy α. Thus, a sound definition of how elements satisfy patterns directly entails a sound definition of a class of pattern languages. In the one-dimensional case, the situation that a word satisfies a pattern is intuitively clear and it can be defined in several equivalent ways, i. e., a word w satisfies the pattern α if and only if

- w can be derived from α by uniformly substituting the variables in α,
- w is a morphic image of α,
- w has a characteristic factorisation for α.

We shall now demonstrate that for a two-dimensional pattern, i. e., a two-dimensional word over the set of variables X, e. g., $\alpha := \left[\begin{smallmatrix} x_1 & x_2 \\ x_3 & x_1 \end{smallmatrix}\right]$, these concepts do not work anymore or they describe fundamentally different situations. For instance, the basic operation of substituting a single symbol in a word by another word cannot that easily be extended to the two-dimensional case. For example, the replacements $x_1 \mapsto [\,\mathtt{a}\ \mathtt{a}\,], x_2 \mapsto \left[\begin{smallmatrix} \mathtt{c} \\ \mathtt{c} \end{smallmatrix}\right]$ and $x_3 \mapsto [\,\mathtt{b}\,]$ may turn α into one of the following objects,

$$\begin{bmatrix} & & \mathtt{c} & \\ \mathtt{a} & \mathtt{a} & \mathtt{c} & \\ & \mathtt{b} & \mathtt{a} & \mathtt{a} \end{bmatrix}, \begin{bmatrix} \mathtt{a} & \mathtt{a} & \mathtt{c} \\ & & \mathtt{c} \\ \mathtt{b} & \mathtt{a} & \mathtt{a} \end{bmatrix}, \begin{bmatrix} & & \mathtt{c} \\ \mathtt{a} & \mathtt{a} & \mathtt{c} \\ \mathtt{b} & \mathtt{a} & \mathtt{a} \end{bmatrix}, \begin{bmatrix} & & & \mathtt{c} \\ \mathtt{a} & \mathtt{a} & & \mathtt{c} \\ & \mathtt{b} & \mathtt{a} & \mathtt{a} \end{bmatrix},$$

which are not two-dimensional words, since they all contain holes or are not of rectangular shape and, most importantly, are not uniquely defined. On the other hand, it is straightforward to generalise the concept of a morphism to the two-dimensional case:

Definition 3.1 *A mapping $h : \Sigma_1^{++} \to \Sigma_2^{++}$ is a two-dimensional morphism if it satisfies $h(V \oplus W) = h(V) \oplus h(W)$ and $h(V \ominus W) = h(V) \ominus h(W)$ for all $V, W \in \Sigma_1^{++}$.*

Hence, we may say that a two-dimensional word W satisfies a two-dimensional pattern α if and only if there exists a two-dimensional morphism which maps α to W. Unfortunately, this definition seems to be too strong as demonstrated by the following example. From an intuitive point of view, the two-dimensional word $W := \begin{bmatrix} \mathtt{a\,a\,b\,a\,a\,b} \\ \mathtt{a\,a\,b\,a\,a\,b} \\ \mathtt{c\,c\,c\,c\,c\,c} \end{bmatrix}$ should satisfy the two-dimensional pattern $\alpha := \left[\begin{smallmatrix} x_1 & x_1 \\ x_2 & x_2 \end{smallmatrix}\right]$, but there is no two-dimensional morphism mapping α to W. This is due to the fact that, as pointed out by the following proposition (which has also been mentioned by Siromoney et al. in [13]), a two-dimensional morphism is a mapping with a surprisingly strong condition.

Proposition 3.2 *Let $\Sigma_1 := \{a_1, a_2, \ldots, a_k\}$ and Σ_2 be alphabets. If a mapping $h : \Sigma_1^{++} \to \Sigma_2^{++}$ is a two-dimensional morphism, then*

$$|h([\,a_1\,])|_c = |h([\,a_2\,])|_c = \ldots = |h([\,a_k\,])|_c \text{ and } |h([\,a_1\,])|_r = |h([\,a_2\,])|_r = \ldots = |h([\,a_k\,])|_r\,.$$

Similarly as in the string case, homomorphisms $h : \Sigma_1^{++} \to \Sigma_2^{++}$ are uniquely defined by giving the images $h(\Sigma_1)$. If in particular $h(\Sigma_1) \subseteq \Sigma_2$, we term the resulting morphism a *letter-to-letter morphism*, while in the even more restricted case when the restriction h_{Σ_1} of h to Σ_1 yields a surjective mapping $h_{\Sigma_1} : \Sigma_1 \to \Sigma_2$, h is referred to as a *projection*.

We can conclude that the existence of a two-dimensional morphism seems to be a reasonable sufficient criterion for the situation that a two-dimensional word satisfies a two-dimensional pattern, but not a necessary one.

In fact, it turns out that characteristic factorisations provide the most promising approach to formalise how a two-dimensional word satisfies a two-dimensional pattern. Recall the example pattern $\alpha = \left[\begin{smallmatrix} x_1 & x_1 \\ x_2 & x_2 \end{smallmatrix}\right]$ from above. Since $\alpha = ([\,x_1\,] \obar [\,x_1\,]) \ominus ([\,x_2\,] \obar [\,x_2\,])$, a characteristic factorisation of a two-dimensional word U for α is a factorisation of the form $U = (V_1 \obar V_1) \ominus (V_2 \obar V_2)$. On the other hand, since $\alpha = ([\,x_1\,] \ominus [\,x_2\,]) \obar ([\,x_1\,] \ominus [\,x_2\,])$, we could as well regard a factorisation $U = (V_1 \ominus V_2) \obar (V_1 \ominus V_2)$ as characteristic for α. For the sake of convenience, we say that the former factorisation is of *column-row type* and the latter one is of *row-column type*. Obviously, the two-dimensional word W from above has a characteristic factorisation of column-row type and a characteristic factorisation of row-column type (with respect to α):

$$W = (V_1 \obar V_1) \ominus (V_2 \obar V_2) = \left(\left[\begin{smallmatrix} \mathtt{a} & \mathtt{a} & \mathtt{b} \\ \mathtt{a} & \mathtt{a} & \mathtt{b} \end{smallmatrix}\right] \obar \left[\begin{smallmatrix} \mathtt{a} & \mathtt{a} & \mathtt{b} \\ \mathtt{a} & \mathtt{a} & \mathtt{b} \end{smallmatrix}\right]\right) \ominus \left(\left[\begin{smallmatrix} \mathtt{c} & \mathtt{c} & \mathtt{c} \end{smallmatrix}\right] \obar \left[\begin{smallmatrix} \mathtt{c} & \mathtt{c} & \mathtt{c} \end{smallmatrix}\right]\right) = \begin{bmatrix} \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} \\ \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} \\ \mathtt{c} & \mathtt{c} & \mathtt{c} & \mathtt{c} & \mathtt{c} & \mathtt{c} \end{bmatrix},$$

$$W = (V_1 \ominus V_2) \obar (V_1 \ominus V_2) = \left(\left[\begin{smallmatrix} \mathtt{a} & \mathtt{a} & \mathtt{b} \\ \mathtt{a} & \mathtt{a} & \mathtt{b} \end{smallmatrix}\right] \ominus \left[\begin{smallmatrix} \mathtt{c} & \mathtt{c} & \mathtt{c} \end{smallmatrix}\right]\right) \obar \left(\left[\begin{smallmatrix} \mathtt{a} & \mathtt{a} & \mathtt{b} \\ \mathtt{a} & \mathtt{a} & \mathtt{b} \end{smallmatrix}\right] \ominus \left[\begin{smallmatrix} \mathtt{c} & \mathtt{c} & \mathtt{c} \end{smallmatrix}\right]\right) = \begin{bmatrix} \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} \\ \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} \\ \mathtt{c} & \mathtt{c} & \mathtt{c} & \mathtt{c} & \mathtt{c} & \mathtt{c} \end{bmatrix}.$$

As a matter of fact, for every two-dimensional word U there exists a characteristic factorisation for $\alpha = \left[\begin{smallmatrix} x_1 & x_1 \\ x_2 & x_2 \end{smallmatrix}\right]$ of column-row type if and only if there exists a characteristic factorisation for α of row-column type. However, this is a particularity of α and, e. g., for $\alpha' = \left[\begin{smallmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_1 \end{smallmatrix}\right]$ and $W' := \left[\begin{smallmatrix} \mathtt{a} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} \\ \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{a} & \mathtt{a} \end{smallmatrix}\right]$, there exists a characteristic factorisation of column-row type $W' = (V_1 \obar V_2 \obar V_3) \ominus (V_2 \obar V_3 \obar V_1) = ([\,\mathtt{a}\ \mathtt{a}\ \mathtt{a}\,] \obar [\,\mathtt{b}\,] \obar [\,\mathtt{c}\,]) \ominus ([\,\mathtt{b}\,] \obar [\,\mathtt{c}\,] \obar [\,\mathtt{a}\ \mathtt{a}\ \mathtt{a}\,])$, but no characteristic factorisation of row-column type. Furthermore, the column-row factorisation of W' is somewhat at odds with our intuitive understanding of what it means that a two-dimensional word satisfies a two-dimensional pattern. This is due to the fact that factorising W' into $([\,\mathtt{a}\ \mathtt{a}\ \mathtt{a}\,] \obar [\,\mathtt{b}\,] \obar [\,\mathtt{c}\,]) \ominus ([\,\mathtt{b}\,] \obar [\,\mathtt{c}\,] \obar [\,\mathtt{a}\ \mathtt{a}\ \mathtt{a}\,])$ means that we associate the two-dimensional factors $[\,\mathtt{a}\ \mathtt{a}\ \mathtt{a}\,]$, $[\,\mathtt{b}\,]$ and $[\,\mathtt{c}\,]$ with the variables x_1, x_2 and x_3, respectively, but in the pattern α' the vertical neighbourship relation between the occurrence of x_2 in the first row and the occurrence of x_3 in the second row is not preserved in W' with respect to the corresponding two-dimensional factors $[\,\mathtt{b}\,]$ and $[\,\mathtt{c}\,]$. More precisely, while a column-row factorisation preserves the horizontal neighbourship relation of the variables, it may violate their vertical neighbourship relation, where for row-column factorisations it is the other way around. Consequently, if we want both the vertical as well as the horizontal neighbourship relation to be preserved, we should require that the two-dimensional word U can be disassembled into two-dimensional factors that induce both a column-row as well as a row-column factorisation. More precisely, we say that U satisfies $\alpha' = \left[\begin{smallmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_1 \end{smallmatrix}\right]$ if and only if there exist two-dimensional words V_1, V_2 and V_3, such that $U = (V_1 \obar V_2 \obar V_3) \ominus (V_2 \obar V_3 \obar V_1) = (V_1 \ominus V_2) \obar (V_2 \ominus V_3) \obar (V_3 \ominus V_1)$, which we call a *proper* characteristic factorisation of U.

We are now ready to formalise the ideas developed so far and we can finally give a sound definition of two-dimensional pattern languages. Although we consider the class of two-dimensional pattern languages that results from the proper characteristic factorisations as the natural two-dimensional counterpart of the class of one-dimensional pattern languages, we shall also define the other classes of two-dimensional pattern languages which were sketched above.

For the definition of two-dimensional patterns, we use the same set of variables X that has already been used in the definition of one-dimensional pattern languages. An *array pattern* is a non-empty two-dimensional word over X and a *terminal array* is a non-empty two-dimensional word over a *terminal* alphabet Σ. If it is clear from the context that we are concerned with array patterns and terminal arrays, then we simply say pattern and array, respectively. Any mapping $h : X \to \Sigma^{++}$ is called a *substitution*. For any substitution h, by $h_{\obar,\ominus}$, we denote the mapping $X^{++} \to \Sigma^{++}$ defined in the following way. For any $\alpha := [y_{i,j}]_{m,n} \in X^{++}$, we define

$$\begin{aligned} h_{\obar,\ominus}(\alpha) := {} & (h(y_{1,1}) \obar h(y_{1,2}) \obar \ldots \obar h(y_{1,n}))\ominus \\ & (h(y_{2,1}) \obar h(y_{2,2}) \obar \ldots \obar h(y_{2,n})) \ominus \ldots \ominus \\ & (h(y_{m,1}) \obar h(y_{m,2}) \obar \ldots \obar h(y_{m,n}))\,. \end{aligned}$$

Similarly, $h_{\ominus,\obar} : X^{++} \to \Sigma^{++}$ is defined. Intuitively speaking, both mappings $h_{\obar,\ominus}$ and $h_{\obar,\ominus}$, when applied to an array pattern α, first substitute every variable occurrence of α by a terminal array according to the substitution h and then these $m \times n$ individual terminal arrays are assembled to one terminal array by either first column-concatenating all the n terminal arrays in every individual row and then row-concatenating the resulting m terminal arrays, or by first row-concatenating all the m terminal arrays in every individual column and then column-concatenating the resulting n terminal arrays.

Let $\alpha \in X^{++}$, $W \in \Sigma^{++}$ and let $h : X \to \Sigma^{++}$. The array W is a (1) *column-row image* of α (*with respect to* h), (2) a *row-column image* of α (*with respect to* h) or (3) a *proper image* of α (*with respect to* h) if and only if (1) $h_{\obar,\ominus}(\alpha) = W$, (2) $h_{\ominus,\obar}(\alpha) = W$ or (3) $h_{\obar,\ominus}(\alpha) = h_{\ominus,\obar}(\alpha) = W$, respectively. The mapping h is called a *column-row substitution for* α *and* W, a *row-column substitution for* α *and* W or a *proper substitution for* α *and* W, respectively. We say that W is a column-row, a row-column or a proper image of α if there exists a column-row, a row-column or a proper substitution, respectively, for α and W.

A nice and intuitive way to interpret the different kinds of images of array patterns is to imagine a grid to be placed over the terminal array. The vertical lines of the grid represent a column concatenation and the horizontal lines of the grid represent a row concatenation of the corresponding factorisation. This means that every rectangular area of the grid corresponds to an occurrence of a variable x in the array pattern or, more precisely, to the array $h(x)$ substituted for x. The fact that an array satisfies a pattern is then represented by the situation that each two rectangular areas of the grid that correspond to occurrences of the same variable must have identical content. In Figure 1, an example for each a morphic image, a proper image, a column-row image and a row-column image of a 5×4 pattern is represented in this illustrative way.

Alternatively, we can interpret the property that a terminal array W is a certain type of image of an array pattern as a tiling of W. More precisely, W satisfies a given array pattern α with n different variables if and only if n tiles can be allocated to the n variables of α such that

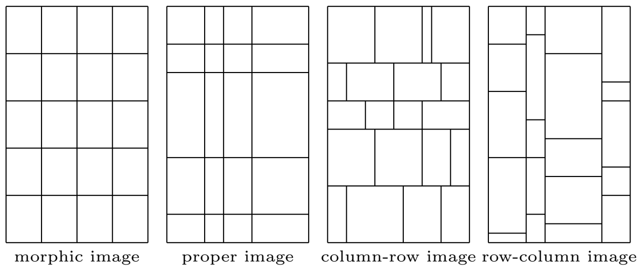

Figure 1: Illustrating possible image partitions

combining the tiles as indicated by the structure of α yields W. The grids depicted in Figure 1 then illustrate the structure of such a tiling.

The definitions of the corresponding classes of pattern languages are now straightforward:

Definition 3.3 *Let $\alpha \in X^{++}$ be an array pattern. We define the following variants of two-dimensional pattern languages:*

- $L_{\Sigma,h}(\alpha) := \{W \in \Sigma^{**} \mid W$ *is a morphic image of* $\alpha\}$,
- $L_{\Sigma,p}(\alpha) := \{W \in \Sigma^{**} \mid W$ *is a proper image of* $\alpha\}$,
- $L_{\Sigma,r}(\alpha) := \{W \in \Sigma^{**} \mid W$ *is a column-row image of* $\alpha\}$,
- $L_{\Sigma,c}(\alpha) := \{W \in \Sigma^{**} \mid W$ *is a row-column image of* $\alpha\}$,
- $L_{\Sigma,rc}(\alpha) := L_{\Sigma,r}(\alpha) \cup L_{\Sigma,c}(\alpha)$.

For a pattern α, we also denote the above languages by Z pattern language of α, *where $Z \in \{h, p, r, c, rc\}$. For every $x \in \{r, c, rc, p, h\}$, we define $\mathcal{L}_{\Sigma,x} := \{L_{\Sigma,x}(\alpha) \mid \alpha \in X^{++}\}$ and $\mathcal{L}_x := \{\mathcal{L}_{\Sigma,x} \mid \Sigma$ is some alphabet$\}$.*

Since, for a fixed array pattern α, every morphic image is a proper image and every proper image is a row-column image as well as a column-row image, the following subset relations between the different types of pattern languages hold (in the following diagram, an arrow denotes a subset relation):

$$\begin{tikzcd}
 & & L_{\Sigma,r}(\alpha) \arrow[dr] & \\
L_{\Sigma,h}(\alpha) \arrow[r] & L_{\Sigma,p}(\alpha) \arrow[ur] \arrow[dr] & & L_{\Sigma,rc}(\alpha) \\
 & & L_{\Sigma,c}(\alpha) \arrow[ur] &
\end{tikzcd}$$

Remark 3.4 *As indicated in the introductory part of this section, we consider the class of p pattern languages as the most natural class of two-dimensional pattern languages. Another observation that supports this claim is that the p pattern languages are compatible, in a certain sense, to the one-dimensional pattern languages. More precisely, for a* one-dimensional *(i. e., $1 \times n$) array pattern α the set $L_{\Sigma,p}(\alpha) \cap \{W \in \Sigma^{++} \mid |W|_r = 1\}$ coincides with the one-dimensional pattern language of α. This does not hold for the h pattern languages (since in the one-dimensional case the words variables are mapped to can differ in length), but holds for the r, c and rc pattern languages. However, as pointed out above, the r, c and rc pattern language of a given pattern α may contain arrays that, from an intuitive point of view, do not satisfy α.*

4. General Observations

In this section, we state some general lemmas about two-dimensional morphisms and array pattern languages, which shall be important for proving the further results presented in this paper. First, we refine Proposition 3.2, by giving a convenient characterisation for the morphism property for mappings on arrays. To this end, we define a substitution h to be *(m, n)-uniform* if, for every $x \in X$, $|h(x)|_r = m$ and $|h(x)|_c = n$ and a substitution is *uniform* if it is (m, n)-uniform, for some $m, n \in \mathbb{N}$.

Lemma 4.1 *A mapping $h : \Sigma^{**} \to \Gamma^{**}$ is a two-dimensional morphism if and only if $h = g_{\ominus,\oplus} = g_{\oplus,\ominus}$, where $g : \Sigma \to \Gamma^{**}$ is a uniform substitution.*

The next lemma states that the composition of two two-dimensional morphisms is again a two-dimensional morphism.

Lemma 4.2 *Let $h_1 : \Gamma^{**} \to \Sigma^{**}$ and $h_2 : \Sigma^{**} \to \Delta^{**}$ be two-dimensional morphisms. Then, the composition $h_{1,2} := h_1 \circ h_2 : \Gamma^{**} \to \Delta^{**}$ is also a two-dimensional morphism.*

It is intuitively clear that the structure of a pattern fully determines the corresponding pattern language and the actual names of the variables are irrelevant, e. g., the patterns $\left[\begin{smallmatrix} x_1 & x_2 & x_1 \\ x_2 & x_3 & x_3 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} x_7 & x_3 & x_7 \\ x_3 & x_5 & x_5 \end{smallmatrix}\right]$ should be considered identical. In the following we formalise this intuition. Two array patterns $\alpha := [y_{i,j}]_{m,n}$ and $\beta := [z_{i,j}]_{m',n'}$ are *equivalent up to a renaming*, denoted by $\alpha \sim \beta$, if and only if $m = m'$, $n = n'$ and, for every i, j, i', j', $1 \le i, i' \le m$, $1 \le j, j' \le n$, $y_{i,j} = y_{i',j'}$ if and only if $z_{i,j} = z_{i',j'}$.

Lemma 4.3 *Let $z, z' \in \{h, p, r, c, rc\}$, let Σ be an alphabet with $|\Sigma| \ge 2$ and let $\alpha, \beta \in X^{++}$. If $L_{\Sigma,z}(\alpha) = L_{\Sigma,z'}(\beta)$, then $\alpha \sim \beta$.*

For every $z, z' \in \{h, p, r, c, rc\}$, $z \neq z'$, $\alpha \sim \beta$ does not necessarily imply $L_{\Sigma,z}(\alpha) = L_{\Sigma,z'}(\beta)$, as pointed out by, e. g., $L_{\Sigma,h}(\left[\begin{smallmatrix} x_1 & x_2 \end{smallmatrix}\right]) \neq L_{\Sigma,p}(\left[\begin{smallmatrix} x_1 & x_2 \end{smallmatrix}\right])$ or $L_{\Sigma,p}(\left[\begin{smallmatrix} x_1 & x_2 \\ x_3 & x_1 \end{smallmatrix}\right]) \neq L_{\Sigma,c}(\left[\begin{smallmatrix} x_1 & x_2 \\ x_3 & x_1 \end{smallmatrix}\right])$. On the other hand, since, for every $z, \in \{h, p, r, c, rc\}$, $\alpha \sim \beta$ obviously implies $L_{\Sigma,z}(\alpha) = L_{\Sigma,z}(\beta)$, two z pattern languages are equivalent if and only if they are described by two patterns that are equivalent up to a renaming.

In the remainder of this work, we do not distinguish anymore between patterns that are equivalent up to a renaming, i. e., from now on we say that α and β are equivalent, denoted by $\alpha = \beta$ for simplicity, if they are actually the same arrays or if they are equivalent up to a renaming.

5. Comparison of Array Pattern Language Classes

In this section, we provide a pairwise comparison of our different classes of array pattern languages and, furthermore, we compare them with the class of recognisable array languages, denoted by REC, which is one of the most prominent classes of array languages. For a detailed description of REC, the reader is referred to the survey [6] by Giammarresi and Restivo. Next, we show that, for every alphabet Σ with $|\Sigma| \geq 2$, the language classes REC, $\mathcal{L}_{\Sigma,h}$, $\mathcal{L}_{\Sigma,p}$, $\mathcal{L}_{\Sigma,r}$, $\mathcal{L}_{\Sigma,c}$ and $\mathcal{L}_{\Sigma,rc}$ are pairwise incomparable. More precisely, for every $\mathcal{L}_1, \mathcal{L}_2 \in \{\mathrm{REC}, \mathcal{L}_{\Sigma,h}, \mathcal{L}_{\Sigma,p}, \mathcal{L}_{\Sigma,r}, \mathcal{L}_{\Sigma,c}, \mathcal{L}_{\Sigma,rc}\}$ with $\mathcal{L}_1 \neq \mathcal{L}_2$, we show that $\mathcal{L}_1 \setminus \mathcal{L}_2 \neq \emptyset$, $\mathcal{L}_2 \setminus \mathcal{L}_1 \neq \emptyset$ and $\mathcal{L}_1 \cap \mathcal{L}_2 \neq \emptyset$. The non-emptiness of the pairwise intersections of these language classes can be easily seen:

Proposition 5.1 *For every $z \in \{h, p, r, c, rc\}$, $L_{z,\Sigma}([\, x_1 \,]) = \Sigma^{++}$ and $\Sigma^{++} \in$ REC.*

It remains to find, for every $\mathcal{L}_1, \mathcal{L}_2 \in \{\mathrm{REC}, \mathcal{L}_{\Sigma,h}, \mathcal{L}_{\Sigma,p}, \mathcal{L}_{\Sigma,r}, \mathcal{L}_{\Sigma,c}, \mathcal{L}_{\Sigma,rc}\}$, a separating language $L_1 \in \mathcal{L}_1 \setminus \mathcal{L}_2$ and a separating language $L_2 \in \mathcal{L}_2 \setminus \mathcal{L}_1$. We first present all these separating languages in a table and then we formally prove their separating property. In rows 2 to 6 of the following table, if a pattern α is the entry that corresponds to the row labeled by class $\mathcal{L}_{\Sigma,z}$ and the column labeled by class $\mathcal{L}_{\Sigma,z'}$, where $z, z' \in \{h, p, r, c, rc\}$, $z \neq z'$, then this means that $L \in \mathcal{L}_{\Sigma,z} \setminus \mathcal{L}_{\Sigma,z'}$. Row 1, on the other hand, contains recognisable array languages that are not array pattern languages.

	REC	$\mathcal{L}_{\Sigma,h}$	$\mathcal{L}_{\Sigma,p}$	$\mathcal{L}_{\Sigma,r}$	$\mathcal{L}_{\Sigma,c}$	$\mathcal{L}_{\Sigma,rc}$
REC	–	$\{[\mathtt{a}]\}$	$\{[\mathtt{a}]\}$	$\{[\mathtt{a}]\}$	$\{[\mathtt{a}]\}$	$\{[\mathtt{a}]\}$
$\mathcal{L}_{\Sigma,h}$	$\begin{bmatrix} x_1 \\ x_1 \end{bmatrix}$	–	$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$
$\mathcal{L}_{\Sigma,p}$	$\begin{bmatrix} x_1 \\ x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$	–	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$
$\mathcal{L}_{\Sigma,r}$	$\begin{bmatrix} x_1 \\ x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$	–	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$
$\mathcal{L}_{\Sigma,c}$	$\begin{bmatrix} x_1 \\ x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$	–	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$
$\mathcal{L}_{\Sigma,rc}$	$\begin{bmatrix} x_1 \\ x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$	$\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$	–

Lemma 5.2 *$L_{\Sigma,h}(\begin{bmatrix} x_1 \\ x_1 \end{bmatrix}) \notin$ REC.*

It can be easily verified that, for every $z \in \{p, r, c, rc\}$, $L_{\Sigma,z}(\alpha) = L_{\Sigma,h}(\alpha)$, where $\alpha := \begin{bmatrix} x_1 \\ x_1 \end{bmatrix}$. Hence, for every $z \in \{h, p, r, c, rc\}$, $L_{\Sigma,z}(\alpha) \notin$ REC, which implies the first column of the table. Furthermore, for every $z \in \{h, p, r, c, rc\}$, $\{[\mathtt{a}]\} \notin L_{\Sigma,z}(\alpha)$, but $\{[\mathtt{a}]\} \in$ REC, which implies the first row of the table.

We point out that, by Lemma 4.3, for every $z, z' \in \{h, p, r, c, rc\}$, $z \neq z'$, if there exists a pattern β with $L_{\Sigma,z}(\beta) \neq L_{\Sigma,z'}(\beta)$, then $L_{\Sigma,z}(\beta) \in \mathcal{L}_{\Sigma,z} \setminus \mathcal{L}_{\Sigma,z'}$ and $L_{\Sigma,z'}(\beta) \in \mathcal{L}_{\Sigma,z'} \setminus \mathcal{L}_{\Sigma,z}$. Consequently, in order to prove the remaining entries of the table, it is sufficient to identify, for every $z, z' \in \{h, p, r, c, rc\}$, $z \neq z'$, a pattern β with $L_{\Sigma,z}(\beta) \neq L_{\Sigma,z'}(\beta)$, which is done by the following two lemmas.

Lemma 5.3 *For every $z \in \{p, c, r, cr\}$, $L_{\Sigma,h}(\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}) \neq L_{\Sigma,z}(\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix})$.*

Lemma 5.4 *For every $z, z' \in \{p, c, r, cr\}$, $z \neq z'$, $L_{\Sigma,z}(\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}) \neq L_{\Sigma,z'}(\begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix})$.*

Proof. Let $\gamma := \begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$ and let $W_1 := \begin{bmatrix} \mathtt{a} & \mathtt{a} \\ \mathtt{a} & \mathtt{a} \\ \mathtt{a} & \mathtt{a} \end{bmatrix}, W_2 := \begin{bmatrix} \mathtt{a} & \mathtt{a} & \mathtt{a} \\ \mathtt{a} & \mathtt{a} & \mathtt{a} \end{bmatrix}$. We observe that $g_{\ominus,\obar}(\gamma) = W_1$, $g'_{\obar,\ominus}(\gamma) = W_2$, where g, g' are defined by

$$g(x_1) := [\,\mathtt{a}\,],\ g(x_2) := \begin{bmatrix} \mathtt{a} \\ \mathtt{a} \end{bmatrix},\ g'(x_1) := [\,\mathtt{a}\,],\ g'(x_2) := [\,\mathtt{a}\ \mathtt{a}\,].$$

Thus, $W_1 \in L_{\Sigma,c}(\gamma)$, $W_2 \in L_{\Sigma,r}(\gamma)$ and $W_1, W_2 \in L_{\Sigma,rc}(\gamma)$. On the other hand, $W_1, W_2 \notin L_{\Sigma,p}(\gamma)$, since every proper image of γ must have an even number of columns and an even number of rows. Consequently, for every $z \in \{r, c, rc\}$, $L_{\Sigma,p}(\gamma) \neq L_{\Sigma,z}(\gamma)$. Similarly, $W_1 \notin L_{\Sigma,r}$ and $W_2 \notin L_{\Sigma,c}$, since every column-row image of γ must have an even number of rows and every row-column image of γ must have an even number of columns. This implies that, for every $z, z' \in \{c, r, cr\}$, $z \neq z'$, $L_{\Sigma,z}(\gamma) \neq L_{\Sigma,z'}(\gamma)$, which concludes the proof. □

6. Closure Properties of Array Pattern Languages

The research of closure properties of classes of formal languages is a classical topic in this area. However, the number of natural properties is richer in the case of arrays compared to the more conventional string case. Thus, in this section, we classify the operations that shall be investigated in this regard according to whether or not they correspond to string language operations.

6.1. String Language Operations

We first point out that, due to Lemma 6.1 below, whenever a non-closure result is known for terminal-free non-erasing string pattern languages, this would straightforwardly transfer to the array case. We will therefore focus on finding proofs for the string case for non-closure properties, and conversely, we will try to give proofs for the array case for closure properties. Interestingly enough, (non-)closure properties have not been studied for the (classical) terminal-free non-erasing string pattern languages, all published proofs that we are aware of for this topic use terminals or erasing. So, our study also contributes to the theory of string pattern languages. Conversely, if we do not manage to find proofs or examples as required for the mentioned approach, this implicitly always raises an open classical string language question.

For any mode $z \in \{h, p, r, c, rc\}$ and any pattern π, let $L^{\mathtt{1D}}_{\Sigma,z}(\pi)$ denote those arrays from $L_{\Sigma,z}(\pi)$ that have just one row, i. e., $L^{\mathtt{1D}}_{\Sigma,z}(\pi) := \{W \in L_{\Sigma,z}(\pi) \mid |W|_r = 1\}$. Clearly, such arrays can be interpreted as strings and vice versa. In this sense, $L^{\mathtt{1D}}_{\Sigma,z}(\pi)$ and the string language $L^{\mathtt{1D}}_{\Sigma}(\pi)$ generated by the pattern π coincide, as long as $z \neq h$. For $z = h$, we encounter the special case that all inserted words have to be of the same length. Let us formulate this more formally:

Lemma 6.1 *Let π be an array pattern of height one. Then, π is, at the same time, a string pattern. Moreover, for any $z \in \{p, r, c, rc\}$, $L^{1D}_{\Sigma,z}(\pi) = L^{1D}_{\Sigma}(\pi)$, while $L^{1D}_{\Sigma,h}(\pi) \subseteq L^{1D}_{\Sigma}(\pi)$.*

We shall now prove non-closure properties for $L^{\mathtt{1D}}_{\Sigma}(\pi)$, which directly carry over to the classes $L_{\Sigma,z}(\pi)$, $z \in \{h, p, r, c, rc\}$ (for some operations, however, the class $L_{\Sigma,h}(\pi)$ constitutes a special case, which is treated separately). To this end, we will mostly focus on two patterns: $\alpha = xyx$ and $\beta = xxy$. The next lemma states an immediate observation for these patterns.

Lemma 6.2 *Over the terminal alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, let $L_s(\alpha)$ ($L_s(\beta)$) denote the shortest words that can be described by α and β, respectively, disallowing erasing. Then,*

$$L_s(\alpha) = \{\mathtt{aaa}, \mathtt{aba}, \mathtt{bab}, \mathtt{bbb}\},$$
$$L_s(\beta) = \{\mathtt{aaa}, \mathtt{aab}, \mathtt{bba}, \mathtt{bbb}\}.$$

Proposition 6.3 *For any non-unary alphabet Σ, $\mathcal{L}^{\mathtt{1D}}_{\Sigma}$ is not closed under union.*

Now if there was a pattern γ such that $L_{\Sigma,z}(\gamma) = L_{\Sigma,z}(\alpha) \cup L_{\Sigma,z}(\beta)$, $z \in \{p, r, c, rc\}$, then, by Lemma 6.1, this would imply $L^{\mathtt{1D}}_{\Sigma}(\gamma) = L^{\mathtt{1D}}_{\Sigma}(\alpha) \cup L^{\mathtt{1D}}_{\Sigma}(\beta)$, contradicting Proposition 6.3. We point out that in the proof of Proposition 6.3, we do not use any replacements by words of different lengths to obtain our contradiction. Hence, this argument is also valid in the case when $z = h$.

Corollary 6.4 *None of the array pattern language classes under consideration (over some non-unary alphabet) is closed under union.*

We proceed with the intersection operation.

Proposition 6.5 *For any non-unary alphabet Σ, $\mathcal{L}^{\mathtt{1D}}_{\Sigma}$ is not closed under intersection.*

Proof. The argument resembles the previous proof. Assume that γ describes $L^{\mathtt{1D}}_{\Sigma}(\alpha) \cap L^{\mathtt{1D}}_{\Sigma}(\beta)$. Notice that $L_s(\gamma) = \{\mathtt{aaa}, \mathtt{bbb}\}$, which clearly implies that $\gamma = xxx$. However, $\mathtt{aabaa} \in (L^{\mathtt{1D}}_{\Sigma}(\alpha) \cap L^{\mathtt{1D}}_{\Sigma}(\beta)) \setminus L^{\mathtt{1D}}_{\Sigma}(\gamma)$. □

Notice that the replacement words we used for deriving a contradiction are of different lengths, meaning that $\mathtt{aabaa} \in L_{\Sigma}(\alpha)$ because of the replacement $x \mapsto \mathtt{aa}$ and $y \mapsto \mathtt{b}$, but $\mathtt{aabaa} \in L_{\Sigma}(\beta)$ because of $x \mapsto \mathtt{a}$ and $y \mapsto \mathtt{baa}$. Hence, we cannot conclude non-closure for the h-mode in the following corollary:

Corollary 6.6 *None of the array pattern language classes under consideration (over some non-unary alphabet and apart from the h-case) is closed under intersection.*

Indeed, the h-mode plays a special rôle, as can be seen by the following result.

Proposition 6.7 *Let Σ be some alphabet. Then, $\mathcal{L}_{\Sigma,h}$ is closed under intersection.*

Arguments as in Propositions 6.3 and 6.5 can be given for any non-trivial binary set operation, for instance, symmetric difference or set difference. This also gives the according result for complementation, but there is also an easier argument in that case. Notice that, as non-erasing pattern languages or array patterns cannot reasonably cope with the empty word or the empty array, we disregard this in the complement operation.

Proposition 6.8 *For any alphabet Σ, $\mathcal{L}^{\mathtt{1D}}_{\Sigma}$ is not closed under complementation.*

Corollary 6.9 *None of the array pattern language classes under consideration (over any alphabet) is closed under complementation.*

Notice that in the other cases (but complementation), we cannot cope with unary alphabets. This might need some different arguments.

We shall now turn to operations that are described by different kinds of morphisms. For the array case, codings (or projections) is a common such operation.

Theorem 6.10 *Any of our array pattern language classes (over arbitrary alphabets) is closed under projections.*

The result does not generalize to (string) morphisms where each image is of the same length.

Proposition 6.11 *For any non-unary alphabet* Σ*,* $\mathcal{L}_{\Sigma}^{1D}$ *is not closed under morphisms that map every letter to a word of length two.*

Corollary 6.12 *None of the array pattern language classes under consideration (over some non-unary alphabet) is closed under two-dimensional morphisms.*

This is also true for the more general operation of substitution, with the same examples.

Let us also remark that Proposition 6.11 did not rely on the fact that we restricted our attention to one specific non-unary alphabet Σ. However, if we have a specific alphabet, then we can even state:

Proposition 6.13 *Any of our array pattern language classes (over some fixed alphabet* Σ*) is closed under some projection* $\pi : \Sigma \to \Sigma$ *if and only if* π *is a bijection.*

This immediately implies the following

Corollary 6.14 *None of our array pattern language classes (over some fixed alphabet* Σ*) is closed under all letter-to-letter morphisms.*

Alternatively, we could also look at inverse morphisms. Here, we already get negative results for inverse codings.

Proposition 6.15 *For any alphabet* Σ *with at least four letters,* $\mathcal{L}_{\Sigma}^{1D}$ *is not closed under inverse letter-to-letter morphisms.*

Corollary 6.16 *None of the array pattern language classes under consideration (over some sufficiently large alphabet) is closed under inverse (two-dimensional) morphisms.*

6.2. Operations Similar to String Language Operations

Let us first turn to the concatenation operation. As a warm-up, we first consider the string case.

Lemma 6.17 *For any alphabet Σ, $\mathcal{L}^{\mathit{1D}}_{\Sigma}$ is closed under concatenation.*

The first thing one should note is that the concatenation of two arrays could be undefined (i. e., if their dimensions do not match), even though the concatenation of the two according languages need not be empty. However, we can prove:

Theorem 6.18 *Fix some alphabet Σ.*

- *$\mathcal{L}_{\Sigma,r}$ is closed under row concatenation $\ominus$;*
- *$\mathcal{L}_{\Sigma,c}$ is closed under column concatenation $\mathbin{⦶}$;*
- *$\mathcal{L}_{\Sigma,p}$ and $\mathcal{L}_{\Sigma,h}$ are closed both under row and under column concatenation.*

It is not a coincidence that for $\mathcal{L}_{\Sigma,r}$ and $\mathcal{L}_{\Sigma,c}$, we had to focus on the "correct" concatenation operation in the preceding theorem. More precisely, we can show:

Theorem 6.19 *Fix some non-unary alphabet Σ.*

- *$\mathcal{L}_{\Sigma,r}$ is not closed under column concatenation $\mathbin{⦶}$;*
- *$\mathcal{L}_{\Sigma,c}$ is not closed under row concatenation $\ominus$;*
- *$\mathcal{L}_{\Sigma,rc}$ is neither closed under row nor under column concatenation.*

Notice that the proofs of negative closure properties necessitate a non-unary alphabet to work.

Lemma 6.20 *Let Σ be a non-unary alphabet. Consider $\alpha = xx$. Then, $(L^{\mathit{1D}}_{\Sigma}(\alpha))^{+} \notin \mathcal{L}^{\mathit{1D}}_{\Sigma}$.*

Proposition 6.21 *Let Σ be a non-unary alphabet. Then, none of the array language families $\mathcal{L}_{\Sigma,x}$ with $x \in \{r, c, rc, p, h\}$ is closed under column concatenation closure nor under row concatenation closure.*

6.3. Operations Special to Arrays

Recall that the transposition operation is first defined for arrays (or patterns) and can then be lifted to languages and even to language classes. Nearly by definition, we find:

Lemma 6.22 *Let Σ be some alphabet. Let α be a pattern. Then, $L_{\Sigma,r}(\alpha)^{T} = L_{\Sigma,c}(\alpha^{T})$ and $L_{\Sigma,c}(\alpha)^{T} = L_{\Sigma,r}(\alpha^{T})$.*

Corollary 6.23 *Let Σ be some alphabet. Then, ${\mathcal{L}_{\Sigma,r}}^{T} = \mathcal{L}_{\Sigma,c}$ and ${\mathcal{L}_{\Sigma,c}}^{T} = \mathcal{L}_{\Sigma,r}$.*

Since $\alpha := \left[\begin{smallmatrix} x_1 & x_2 \\ x_2 & x_1 \end{smallmatrix}\right]$ is identical to its transposition and, as shown in the proof of Lemma 5.4, describes an r pattern language (a c pattern language), which is not a c pattern language (not an r pattern language, respectively), we can conclude the following:

Proposition 6.24 *Let Σ be an alphabet. Neither $\mathcal{L}_{\Sigma,c}$ nor $\mathcal{L}_{\Sigma,r}$ are closed under transposition.*

Proposition 6.25 *For any alphabet Σ and $x \in \{h, p, rc\}$, $\mathcal{L}_{\Sigma,x}$ is closed under transposition.*

With respect to purely geometric operations as turns and reflections, we find the following:

Proposition 6.26 *Let Σ be some alphabet.*

- *$\mathcal{L}_{\Sigma,rc}$, $\mathcal{L}_{\Sigma,p}$ and $\mathcal{L}_{\Sigma,h}$ are closed under quarter-turn.*
- *For every $x \in \{r, c, rc, p, h\}$, $\mathcal{L}_{\Sigma,x}$ is closed under half-turn and reflections.*
- *$\mathcal{L}_{\Sigma,r}$ and $\mathcal{L}_{\Sigma,c}$ are closed neither under left nor under right turn.*

The positive closure properties can be easily observed by applying the geometric operation directly on the array pattern. In order to show non-closure of $\mathcal{L}_{\Sigma,r}$ and $\mathcal{L}_{\Sigma,c}$ with respect to left and right turn, it is again sufficient to observe that the pattern α from above is identical to its left or right turn and then apply a similar argument as in the proof of Lemma 5.4.

Due to symmetry, it does not matter if we consider horizontal or vertical reflections. Notice that both half-turns and reflections coincide in the string case in any meaningful, non-trivial interpretation; in that case, the operation is also known as mirror image.

7. Future Research Directions

A thorough investigation of the typical decision problems for two-dimensional pattern languages like the membership, inclusion and equivalence problem is left for future research. It can be easily seen that the NP-completeness of the membership problem for string pattern languages carries over to $\mathcal{L}_{\Sigma,x}$, $x \in \{p, r, c, rc\}$. On the other hand, for a given array pattern α and a terminal array W, the question whether or not $W \in L_{\Sigma,h}(\alpha)$ can be decided in polynomial time by checking whether W is a morphic image of α with respect to a $(\frac{|W|_r}{|\alpha|_r}, \frac{|W|_c}{|\alpha|_c})$-uniform substitution. As shown by Lemma 4.3, the equivalence problem for all the classes $\mathcal{L}_{\Sigma,x}$ with $x \in \{h, p, r, c, rc\}$ and $|\Sigma| \geq 2$ can be easily solved by simply comparing the patterns. However, for every $z, z' \in \{h, p, r, c, rc\}$, $z \neq z'$, the problem to decide for given patterns α and β whether or not $L_{\Sigma,z}(\alpha) = L_{\Sigma,z'}(\beta)$ might be worth investigating. The inclusion problem for terminal-free nonerasing string pattern languages is still open. Hence, with respect to the inclusion problem, a positive decidability result for two-dimensional pattern languages implies a positive decidability result for terminal-free nonerasing string pattern languages.

For string pattern languages it is common to use terminal symbols in the patterns as well as to consider the *erasing* case, i. e., variables can be replaced by the empty word. The p pattern languages can be adapted to the erasing case by allowing variables to be substituted by the empty array. Furthermore, the situation of having a terminal symbol at position (i, j) of an array pattern simply forces all the variables in the i^{th} row to be substituted by arrays of height 1 and all the variables in the j^{th} column to be substituted by arrays of width 1. As in the string case, it is likely that in the two-dimensional case the difference between erasing and nonerasing substitutions and patterns with and without terminal symbols lead to different language classes with different decidability properties, too.

Finally, we wish to point out that it is straightforward to generalise our different classes of two-dimensional pattern languages to the three-dimensional or even n-dimensional case.

References

[1] D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[2] H. Fernau, R. Freund, and M. Holzer. The generative power of d-dimensional #-context-free array grammars. In M. Margenstern, editor, *Proceedings of MCU'98, Volume 2*, pages 43–56. University of Metz, 1998.

[3] R. Freund, G. Păun, and G. Rozenberg. Chapter 8: Contextual array grammars. In C. Martín-Vide, V. Mitrana, and G. Păun, editors, *Series in Machine Perception and Artificial Intelligence: Volume 66 - Formal Models, Languages and Applications*, pages 112–136. World Scientific, 2007.

[4] J. E. F. Friedl. *Mastering Regular Expressions.* O'Reilly, Sebastopol, CA, third edition, 2006.

[5] D. Giammarresi and A. Restivo. Recognizable picture languages. *International Journal of Pattern Recognition and Artificial Intelligence*, 6:31–46, 1992.

[6] D. Giammarresi and A. Restivo. Two-dimensional languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 4, pages 215–267. Springer, 1997.

[7] D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Information and Computation (formerly Information and Control)*, 125:32–45, 1996.

[8] A. Mateescu and A. Salomaa. Patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 230–242. Springer, 1997.

[9] O. Matz. Recognizable vs. regular picture languages. In *Proc. 2nd International Conference on Algebraic Informatics, CAI 2007*, volume 4728 of *Lecture Notes in Computer Science*, pages 75–86, 2007.

[10] K. Morita. Two-dimensional languages. In C. Martín-Vide, V. Mitrana, and G. Păun, editors, *Studies in Fuzziness and Soft Computing - Formal Languages and Applications*, pages 427–437. Springer, 2004.

[11] A. Rosenfeld. *Picture Languages: Formal Models for Picture Recognition.* Academic Press, Inc., Orlando, 1979.

[12] A. Rosenfeld and R. Siromoney. Picture languages – a survey. *Languages of Design*, 1:229–245, 1993.

[13] G. Siromoney, R. Siromoney, and K. Krithivasan. Picture languages with array rewriting rules. *Information and Control*, 22:447–470, 1973.

[14] P. S. P. Wang. *Array Grammars, Patterns and Recognizers.* World Scientific Publishing Co., Inc., NJ, USA, 1989.