PICTURE LANGUAGES GENERATED BY SPLICING AND ASSEMBLING TILES

Anthonath Roslin Sagaya Mary

Presented by
Anthonath Roslin Sagaya Mary

Picture Languages Generated by Splicing and Assembling Tiles

Doctoral Thesis

Supervised by
K. G. Subramanian
Paola Bonizzoni

Co-Supervised by
María Dolores Jiménez López

Universitat Rovira i Virgili
Tarragona, 2015

Supervisors :

Professor K G Subramanian
School of Mathematical Sciences,
Universiti Sains Malaysia,
11800 Penang, Malaysia.

Professor Paola Bonizzoni
Dipartimento di Informatica Sistemistica e Comunicazione
Università degli Studi di Milano-Biccoca,
Viale Sarca 336 Milano, 20126 (ITALY).

Co-Supervisor

Assistant Professor. María Dolores Jiménez López
Grup de Recerca de Linguística Matemática
Departament de Filologies Romàniques
Facultat de Lletres, Universitat Rovira i Virgili
Av. Catalunya 35, 43002 Tarragona (SPAIN)

Dr K.G. Subramanian
Visiting Professor (March 2015 – Feb. 2017)
Liverpool Hope University, Liverpool, UK
Formerly (1970-2006), Department of Mathematics,
Madras Christian College, Chennai, India
& Visiting Professor (July 2007- May 2015), UniversitiSains Malaysia,
Malaysia

As one of the supervisors of the present doctoral dissertation entitled "PictureLanguage Generated by Splicing and Assembling Tiles", presented by Ms AnthonathRoslinSagaya Mary, PhD Scholar at theDepartament de FilologiesRomàniques,**I STATE**that this thesis has been prepared by herwithme assupervisor and that it fulfils all therequirements to be defended.

(K.G. Subramanian)
Date: November 28, 2015

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano– Bicocca
Viale Sarca 336 – Edificio U14 – 20126 Milano

As a supervisor of the present doctoral dissertation entitled "Picture Language Generated by Splicing and Assembling Tiles", presented by Ms. Anthonath Roslin Sagaya Mary,

I STATE

That this thesis has been carried out under my co-supervision at the Departament of Informatics, Systems and Communication, and that it fulfills all the requirements to be defended.

Milan, 26 November 2015

Prof. Paola Bonizzoni – Tel. +39 02 64487814 – e-mail: bonizzoni@disco.unimib.it

UNIVERSITAT ROVIRA I VIRGILI
DEPARTAMENT DE FILOLOGIES ROMÀNIQUES

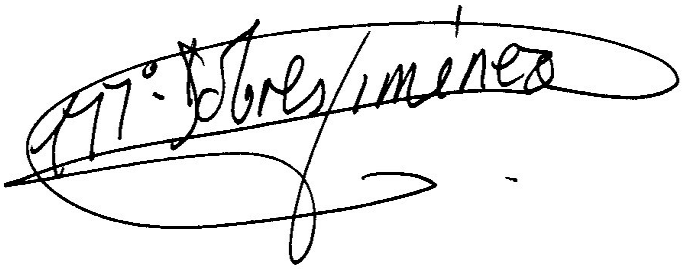
As a co-supervisor of the present doctoral dissertation entitled “Picture Language Generated by Splicing and Assembling Tiles”, presented by Ms. Anthonath Roslin Sagaya Mary,

I STATE

That this thesis has been carried out under my co-supervision at the Departament de Filologies Romàniques, and that it fulfills all the requirements to be defended.

Tarragona, 26 November 2015

Doctoral Thesis Co-supervisor



Dr. M. Dolores Jiménez López
Departament de Filologies Romàniques
Universitat Rovira i Virgili

UNIVERSITAT ROVIRA I VIRGILI
PICTURE LANGUAGES GENERATED BY SPLICING AND ASSEMBLING TILES
Anthonath Roslin Sagaya Mary

Abstract

The extension of the study of formal languages over string case to two dimensional languages or picture languages has been of interest for long for its vast applications.

The objective of this thesis concentrates on the study of generation of Picture language classes by bio-inspired operations namely, 'Splicing' and 'Self-Assembly' of DNA-Computing. Henceforth the study presented is given by two formalisms in which former is an extension of String case H-Splicing with given set(sequence) of Domino Splicing Rules and the later a new formalism emerging again with Tiling rules (in sequence) which assemble tiles. Thus Pictures are generated by applying these rules in columns and rows.

One of the formalism is H-Array Splicing Systems HAS introduced as an extension of string case also including its restriction languages to produce language classes Self-Cross Over Array Languages $\mathcal{L}(A_{SCO})$ and Simple Array Splicing Languages $\mathcal{L}$(SASL) . The Splicing Operation to cut a context of the strings and paste by concatenation is extended similarly to Picture Languages by set of Domino Splicing Rules. These sequence of Domino Splicing Rules generate pictures by cutting and pasting set of initial pictures in columns and rows according to the rules. Various Picture languages enhancing itself by iterative applications of the rules are constructed for each classes. And incomparability results are proved between the classes $\mathcal{L}$(HASL) , $\mathcal{L}(A_{SCO})$, $\mathcal{L}$(SASL) and 2D-RLG, LOC. But the classes intersect since we have proved common picture languages. Also, a Parallel Grammar System based formalism called Splicing Array Grammar Systems is introduced and studied by obtaining, elegant comparisons with string case Regular, CF, CS languages. The comparisons are done based on its component being 2D-RLG.

Then we also study a cell biology inspired formalism called Pictural Networks of Evolutionary Processors. We apply contextual insertion, deletion and substitution to the formalism in the process of Picture generation.

Also, we compare this language class with that of Puzzle grammars. Then we study some properties of extending pictures to three dimensional notions of recognizability given already.

Another main formalism introduced and studied in the thesis is Tiling Rule System TRuS . This formalism generates picture by set of tiling rules, assembling tiles. We have proved that the class of $\mathcal{L}$(TS) (Tiling System, recognizable language) is contained in TRuS . Also, we prove there exist a construct of the formalism based on generating pictures in rows or columns which is equivalent to $\mathcal{L}$(TS) . Thus leading to an interesting notion of bio-inspired (self-assembling) operation to picture generation.

Contents

Chapter 1

Introduction

1.1 Background

One of the extensions of string language theory is to two-dimensional languages (picture languages). There has been a continued interest in adapting the techniques of formal string language theory for developing methods to study the problem of picture generation and description. As early as 1964, Narasimhan [35] suggested a syntactic model for the solution of problems in picture processing where pictures are considered as connected, digitized finite arrays in the two-dimensional plane. All these approaches were initially motivated by problems arising in the framework of pattern recognition and image processing. In syntactic approaches to generation and recognition of images or picture patterns, considered as digitized arrays, several two-dimensional grammars have been proposed and studied in [40, 37, 42, 45, 48, 47]. These studies adapt the techniques of formal string language theory and introduce various types of picture or array grammars. Most of the array grammars developed to handle picture languages, are based on Chomskian string grammars.

The study of picture languages has vast applications, some of which are tiling problems in math, topology, physics and biology too. In the thesis and the most common two-dimensional object studied is a picture which is a rectangular array of symbols taken from a finite alphabet. We restrict ourselves mainly to the study of languages build from such objects. These languages are called Picture languages.

This thesis is titled as “Picture Languages Generated by Splicing and Tile Assembling” which in elaborate are the classes of picture languages

generated by bio-inspired similar operations. i.e. of DNA computing, particularly the nature of recombinant DNA sequences. Splicing and Self Assembling is extended to sequence of dominoes and tile assembling with defined set of domino splicing rules or tiling rules to generate Picture Languages. i.e. it is the construction of the systems by forming rules with dominoes and tiles to apply on the rows and columns of the pictures. The idea in application is analogous to the splicing operation and self-assembling nature of DNA sequences defined for the string case.

Extension of the splicing operation to graphs has been proposed by Freund in [19]. Relationship between graph splicing languages of Freund in [19] and Hyper-edge replacement graph languages in [29] is examined in [23]. Sakakibara and Ferretti in [48] have introduced and studied splicing of tree structures. Krithivasn in [28] has considered a different kind of splicing of graphs. As an extension of the ideas Splicing of arrays structures which could be thought of as graphs on grid structures that has been considered in [29, 25]. A simple but effective method of splicing on images of rectangular arrays is introduced in [25] as an extension of the operation of splicing on strings.

Firstly, the class of Picture languages proposed and studied in the thesis, with the bio-inspired operation Splicing for string case by T. Head in [23] is called H-Array Splicing Systems in [24]. The approach to Array(Picture) splicing considered in subramanian et.al in [24] is different from the one considered by Krithivasan et.al [29].

Dassow and Mitrana in [13] investigated a very simple and natural restriction on the splicing operation, namely cross-over rule applicable only on identical strings in trying to capture features of the recombination of genes in a chromosome. The self cross-over operation on arrays is also introduced in [24] and the resulting language is called *self cross-over array languages* where it is defined by the H-array splicing on identical arrays. This family is referred by $\mathcal{L}$(SASL) in the thesis and with further results of incomparability with main Recognizable picture language class $\mathcal{L}$(TS) and the 2D-Right Linear Grammar class studied in [31].

Mateescu et. al in [32] introduced a special kind of a splicing rule, called Simple Splicing Rule and investigated the effect of this type of rules on words. This notion of simple splicing has been examined in [8] for circular words. As an application of the concept of simple splicing, it is natural to extend this special form of splicing to higher dimensional structures such as graphs, trees and arrays in [43] called Simple Array Splicing

Systems and the language class as $\mathcal{L}$(SASL) . We state the results on various examples and counter examples of the $\mathcal{L}$(SASL) class in the thesis.

The theory of Grammar systems is a well-investigated field of formal language theory, providing a theoretical framework for modeling various kinds of multi-agent systems at the symbolic level [12]. A grammar system consists of several grammars or other language identifying mechanisms, that cooperate according to some well-defined protocol. The components of the system correspond to the agents, the current string(s) in generation to a symbolic environment, and the system's behaviour is represented by the language. Among a variety of grammar system models, Parallel Communicating Grammar Systems, in which the components are generative grammars working on their own sentential forms in parallel and communicating with each other by sending their sentential forms by request, have been of intensive study [12, 11].

A new type of Parallel Communicating grammar systems has been introduced in [13] by replacing communication by splicing of strings. P$\check{a}$un [21] has investigated Splicing Grammar Systems improving the results of [13].
Freund [20] has introduced and investigated cooperating distributed array grammar systems extending the concept of cooperation in string grammar systems and using array grammars. Motivated by the study of Dassow and Mitrana (1996), we consider Grammar Systems that describe Images or Pictures of rectangular arrays in [44]. The components of the Grammar system consist of two-dimensional Grammars in [22] and domino splicing rules [11] with the grammars working in parallel and splicing rules acting on arrays of two components yielding rectangular arrays of symbols. The resulting systems are called Splicing Array Grammar Systems which is given the study of $\mathcal{L}$(HASL) in the thesis. Different component grammars such as Regular Matrix grammars [22, 40], Context-free Matrix grammars [22, 40], are considered and properties such as generative power, comparison etc in [44].

An interesting Computing model inspired by cell biology, called Network of Evolutionary processors, was introduced by Castellanos et al [6] and the investigation of this model continued in Castellanos et al [7] and Martin-Vide et al [30]. This notion has been carried over to pictures and Pictural Networks of Evolutionary Processors (PNEP) have been considered by Mitrana et al [34] . We also include the notion of Contextual Pic-

tural Network of Evolutionary Processors using contextual insertion and deletion rules that we introduced in [18]. These rules are a special case of contextual insertion/deletion studied by Mitrana [33]. We have presented this notion also with some results of extending three dimensional notion for the existing ideas of two dimension, three dimension recognizability being introduced for languages defined by [16].

The other new and the main formalism of the thesis called Tiling Rule System (TRuS) which is based on assembling tiles along the columns and rows of the pictures is introduced and studied in [4, 5]. The idea of tile assembling being inspired by the vast DNA computing area with mainly self-assembly in [49] with its universality result of Turing-Machine. We have considered the Wang Systems to compare with TRuS since Wang Systems is an interesting system derived from Wang Tiles for Tiling problem results. Wang Systems in [47] is proved to be equivalent to Recognizable Picture Language class $\mathcal{L}(\mathrm{TS})$.

1.2 Study and Contribution

1.2.1 Picture languages : Splicing and Assembling Tiles in (rows)columns

H-array Splicing Systems (HAS)

H-array Splicing Systems HAS is a bio-inspired formalism extended from H-Splicing from string case, a vastly investigated study introduced by T. Head. HAS formalism contributes to the aspects of over grammar, automata theory for picture languages. In particular it is structured as a mechanism by studying two-dimensional right linear grammars. In elaborate this formalism is a mechanism which is applied on finite number of pictures called initial pictures I with given set of column and row domino splicing rules. A column or row domino splicing rule is given by $\delta_1; \delta_2 : \delta_3; \delta_4$, where δ_1 and δ_2 are two adjacent columns or rows over a given picture $p_1 \in I$ and similarly δ_3 and δ_4 two adjacent columns or rows over picture $p_2 \in I$. It operates by 'cutting' two given pictures p_1 and p_2 at the identified context of columns and rows with that of the one formed by δ_1 and δ_2 in a sequence by the given set of domino splicing rules. The context site where the two pictures are cut in columns and rows are decided by the sequence of adjacent dominoes in the set of rules. And then the 'pasting' of the first part (or sub-array) of the picture p_1 to the second

part of the picture p_2 is done by column and row concatenations respectively. The column and row concatenation operation, various geometrical rotation over the pictures is proved to be closed for the class of H-Array Splicing languages $\mathcal{L}$(HASL) .

The idea of Self-cross over from string case is applied to HAS .i.e. by restricting the system to apply HAS rules over two identical pictures are called Self-cross over array languages $\mathcal{L}(A_{SCO})$. This class of languages are proved not be closed under union and concatenation operation. Then also Simple Splicing rules are extended which is a restriction over the domino splicing rules for HAS . Simple Domino Splicing rules are given by $\delta_1; \lambda : \delta_2; \lambda$. Thus applying the splicing operation at any column and row or the initial picture. This class of language $\mathcal{L}$(SASL) also seems to be having properties corresponding to that of string case and analogous to $\mathcal{L}$(HASL) .

One of the main classification of the Picture language class is Recognizability by the Tiling Systems $\mathcal{L}$(TS) in [22] defined by LOC languages. i.e. Picture language obtained as projection on alphabets of pictures in LOC, which are pictures identified by a given set of two by two windows otherwise called as tiles. This class of language have been constantly explored for its close relation to regular language of string case, specifically with closure properties. And that the formalism concerns also with this structure on finding an automata for Picture language. Recently there have been approaches introduced as subclasses to this class of language $\mathcal{L}$(TS) to study the deterministic properties and unambiguity properties specially with columns and rows.

As of giving grammar based systems to Picture languages there has been a constant interest and introduction to formalisms that are derived from string case. One of the grammars that has been studied with automata for Pictures is called 2D-Right Linear Grammar (2D-RLG) also in [22]. This grammar has been of interest to compare also since its elegant approach extending the string grammars straight away for constructing pictures. The first horizontal string is generated as intermediate string until generating all the vertical strings from each intermediate alphabet symbol of the string to form the pictures.

In the study of this thesis with examples and counter examples in the list of the results for the classes of HAS and its restriction languages defined, the results are incomparable with the above two described main notions LOC and 2D-RLG, but are not disjoint i.e. in [24, 31, 43] we have $\mathcal{L}$(HASL) , $\mathcal{L}(A_{SCO})$, $\mathcal{L}$(SASL) are incomparable with LOC and

2D-RLG but are not disjoint, since we have obtained common examples of each class. Also, $\mathcal{L}$(HASL) and $\mathcal{L}(A_{SCO})$ seem to be incomparable but not disjoint, which is analogous with the string case formalism.

Although the main study is on picture languages with defined bio-inspired operations over its columns and rows and its properties. The construction of the systems studied in this thesis also concerns on contributing to the formation of grammars and automata for picture languages.

The defined operations in the form of rules which are in accord to apply over columns and rows of pictures are also consider in construction of a formalism with Grammar System called Splicing Grammar Systems. It is done by applying the defined H-array splicing operation of domino splicing rules on pictures of the grammar model ($2D$-Right linear grammar languages), generating them and following or simulating existing string case language classes Regular, CF or CS, which are studied for the build of automata, to be more elegant for extending from the string case. This Splicing array Grammar System is also compared with small variant grammar for Picture language called 2D-Tabled matrix grammar. Various inclusion results are of the language class similar to the string case are obtained. In conclusion of the class $\mathcal{L}$(HASL) presented above with descriptions, it is an expected simple and elegant contribution in the extension of string case to picture language case. Note that this formalism captures one of the Grammar System branch called parallel Grammar Systems naturally extending to Picture language class and the H-Array Splicing Systems, explicitly self explaining in [44]. Hence we state the definition straight away with out including the the vast study on Grammar Systems and its branches. And all its already existing inspiration to array languages by Freund.et.al [20].

Further more on the underlying study of Picture language comparisons we have another cell biology inspired formalism called Pictural Networks of Evolutionary Processors (PNEP) in consideration. A PNEP has nodes that are very simple processors able to perform just one type of operation, namely insertion or deletion of a row or substitution of a symbol in rectangular arrays. These nodes are endowed with filters defined by some membership or random context conditions. We also include the notion of Contextual Pictural Network of Evolutionary Processors using contextual insertion and deletion rules that we introduced in [18]. We have studied comparisons with Puzzle Grammars and also for the three dimensional notions in [16] extending 2D pictures to higher dimensions.

Tiling rule systems

Also, investigation of Picture Languages has moved towards the definition of formal models capable of characterizing special classes of languages that are not included in the family of recognizable languages generated by tiling systems of Giammaresi and Restivo [22]. An example of such models is tile rewriting grammars (TRG) defined in [10] and further investigated. Indeed, while tiling systems represent an extension to the two dimensional case of regular string grammars, TRG provide an analogue of context-free grammars in the two dimensions, and can generate interesting picture languages that generalize context-free string languages, including for example Dyck languages. Grammar approaches, besides tiling systems and cellular automata (see [22] for a complete survey), reflect the efforts done towards a generalization of classical formal language theory to the two dimensional case. This research direction is now a rich field of investigation (see [1], [2], [27] and [3] as an example).

In this chapter 5, our investigation of picture languages goes in a different direction, since we propose new operations that are not generalizations of classical formal language concepts, but are instead inspired by operations used in modeling DNA self assembly [49]. More precisely, our approach for generating pictures is based on a notion of *tiling rule system*, consisting of an initial finite set of pictures and a set of rules that can be iteratively applied to the initial language to generate a picture language.

A rule consists of a pair of tiles: a *context site* and a *replacement site* tile. Context site is used to specify where the rule can be applied, while the replacement site is used to change part of the context site. This type of rule generalizes to the 2-dimensional case a typical behavior of rules acting on DNA strings, i.e. a context is needed to allow the applications of rules, while replacement specifies how the context will be modified.

In a tiling rule system, at each step a set of rules is simultaneously applied to a picture from the initial language or assembled in a previous computation step. The effect of the simultaneous application of rules is the replacement and insertion of a row or column, respectively, so allowing the growth of a new picture according to the rule system.

Formally a tiling rule system, TRuS system in short, is a triple (I, R, Σ), where I is an initial finite set of pictures, R is a finite set of tiling rules and Σ is the alphabet of the generated pictures.

Observe that our notion of tiling rule system is different from tiling systems, but also from Wang systems [15], which model DNA self assembly by pure growth and which are proved to be equivalent to tiling systems.

In tiling systems by Giammaresi and Restivo [22], picture languages are defined by a projection function applied to a local language defined by a set of tiles.

We show that TRuS systems have greater generative capacity than the tiling systems, even in the case of systems generating one-letter alphabet picture languages. More precisely, the constructive proof of a TRuS system that simulates a tiling system shows that recognizable languages are generated by rules that act always by growing pictures along their borders. On the contrary the class of languages generated by TRuS systems includes languages that seem to strictly require rules acting on specific positions inside the pictures in order to grow the language. This is for example the case of the language of palindromic columns, that has been proved not being a recognizable language in [4]. In this study we have proved $\mathcal{L}(\text{TS}) = \mathcal{L}(\text{sTRuS}) \subset \mathcal{L}(\text{TRuS})$. Where $\mathcal{L}(\text{TS})$ is the recognizable class of picture languages defined with the LOC class. $\mathcal{L}(\text{sTRuS})$ is called the simple Tiling Rule Systems where the picture grows only in columns or rows with tiling rules.

1.3 Structure of the thesis

In Chapter 2 we give the preliminaries for the rest of the Chapters to follow. In section 2.1 we have stated all notations, definitions and examples of Picture Language theory that are relevant to the study. Namely, that of Tiling System Recognizable languages, Wang Systems, Puzzle Grammars, 2D-Right Linear Grammars. Then section 2.2 refers all string case Splicing notions relevant to the thesis.

In Chapter 3 introduces the HAS language class, its restriction language classes, $\mathcal{L}(A_{SCO})$ and $\mathcal{L}(\text{SASL})$, giving definitions in 3.1, examples and counter examples in 3.2 along with the comparison study of $\mathcal{L}(\text{HASL})$, $\mathcal{L}(A_{SCO})$ and $\mathcal{L}(\text{SASL})$. Language class of Simple Tree Splicing Systems $\mathcal{L}(STSS)$ is stated and studied in section 3.3. And the last section of this chapter 3.4 contains the notion of Splicing Array Grammar Systems and its comparison study.

In Chapter 4 the Pictural Network of Evolutionary Processors is studied with contextual insertion, deletion and substitution rules. In section 4.2 three dimensional picture languages are introduced for the notion and some comparison results are followed in the subsections.

In Chapter 5 introduces the new main formalism TRuS and its results. Section 5.1 gives the formalism, its construction and examples. Section

5.2 gives the generative power of the formalism, one of the main results. Then in section 5.3 also comparison of the structure of the formalism with Wang Systems is done.

In Chapter 6 concludes the various results and the structure the study with the two main formalism given, also stating its main results with some proposals for the further study. Each Chapter consists of its References and the full list of Bibliography follows the last chapter.

Chapter References

[1] M. Anselmo, D. Giammarresi, and M. Madonia. New operations and regular expressions for two-dimensional languages over one-letter alphabet. *Theoretical Computer Science*, 340:408 – 431, 2005.

[2] M. Anselmo and M. Madonia. Deterministic two-dimensional languages over one-letter alphabet. *Lecture Notes in Computer Science*, 4728:147–159, 2007.

[3] A. Bertoni, M. Goldwurm, and V. Lonati. On the complexity of unary tiling-recognizable picture lanaguages. *Lecture Notes in Computer Science*, 4393:381–392, 2007.

[4] P. Bonizzoni, C. Ferretti, Anthonath Roslin Sagaya Mary, and G. Mauri. Picture languages generated by assembling tiles. volume Lecture Notes in Computer Science, pages 224–235, 2009.

[5] P. Bonizzoni, C. Ferretti, Anthonath Roslin Sagaya Mary, and G. Mauri. Picture languages generated by assembling tiles. *Fundam. Inform. 110*, 1-4:77 – 93, 2011.

[6] J. Castellanos, Mart. Cín-Vide, V. Mitrana, and C. Sempere. Solving NP-complete problems with networks of evolutionary processors. *LNCS*, 2084, Springer-Verlag:621–628, 2001.

[7] J. Castellanos, Mart. Cín-Vide, V. Mitrana, and J. Sempere. Networks of evolutionary processors. *Acta Informatica*, 39:517–529, 2003.

[8] R. Ceterchi and K. G. Subramanian. Simple circular splicing systems. *Romanian Journal of Inform. Sci. and Tech.*, 6:121–134, 2003.

[9] S. Crespi Reghizzi and M. Pradella. Tile rewriting grammars and picture languages. *Theoretical Computer Science*, 340:257– 272, 2005.

[10] E. Csuhaj-Varjú. Grammar systems: 12 years, 12 problems (short version),. In Freund, R. and Kelemenov, A. (Eds.) Proceedings of the International Workshop on Grammar Systems, Silesian University, Opava.:77–92, 2000.

[11] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, and Gh. Paun. A grammatical approach to distribution and cooperation. *Grammar systems*, Gordon and Breach Science Publishers, 1994.

[12] J. Dassow and V. Mitrana. Self cross-over systems. in : Computing with bio - molecules: Theory and experiments. *(Eds.) Gh. Paun, Discrete Mathematics and Theoretical Computer Science*, Springer-Verlag:pp. 283 – 294, 1998.

[13] L. De Prophetis and V. Varricchio. Recognizability of rectangular pictures by wang systems. *Journal of Automata, Language and Combinatorics*, 2:269–288, 1997.

[14] K. S. Dersanambika and K. Krithivasan. Recognizability of 3D rectangular pictures. *Paper presented at the 12th International Conference of Jangeon Mathematical Society, University of Mysore*, Dec. 2003.

[15] Anthonath Roslin Sagaya Mary Dersanambika. k. S, Subramanian. K. G. 2D and 3D pictural networks of evolutionary processors. *IWINAC, Lecture Notes in Computer Science*, 2:92–101, 2005.

[16] R. Freund. Splicing systems on graphs. *Proc. Intelligence in Neural and Biological Systems*, IEEE Press:189–194, 1995.

[17] R. Freund. Array grammar systems. *Journal of Automata, Languages and Combinatorics*, 5,1:13–29, 2000.

[18] Rozenberg. G Gh. Paun and Salomaa. A. Computing by splicing. *Theoretical Computer Science*, 168:321–336, 1996.

[19] D. Giammarresi and A. Restivo. *Two-dimensional languages, In "Handbook of Formal Languages"*, volume Springer Verlag,3. Springer Verlag, Vol.3, 1997.

[20] T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. *Bull. Math. Biology*, 49:737 –759, 1987.

[21] P. Helen Chandra, K. G. Subramanian, and D. G. Thomas. Parallel splicing on images. *Int. J. of pattern recognition and artificial intelligence*, 18(6):1071–1091, 2004.

[22] P. Helen Chandra, K. G. Subramanian, D. G. Thomas, and D. L. Van. A note on parallel splicing on images. In *IWCIA-2001*, volume Proceedings of the 8th International Workshop on Combinatorial Image Analysis, 2001.

[23] J. Kari and C. Moore. New results on alternating and nondeterministic two-dimensional finite-state automata. *Lecture Notes in Computer Science*, 2010:396–406, 2001.

[24] K. Krithivasan. Splicing systems-the graph model. pages 33–59, 1998.

[25] K. Krithivasan, V.T. Chakaravarthy, and R. Rama. Array splicing systems. *Lecture Notes in Computer Science*, Control Cooperation and Combinatorics", Eds. Gh.Paun and A. Salomaa, 1218, Springer Verlag:346–365, 1997.

[26] V. Martin-Vide, C. Mitrana, M. J. Perez-Jimenez, and F. Sancho-Caparrini. Hybrid networks of evolutionary processors, genetic and evolutionary computation conference (gecco 2003). *Lecture Notes in Computer Science*, 2723, Springer Verlag, Berlin:401–412, 2003.

[27] Anthonath Roslin Sagaya Mary and K. G. Subramanian. Self crossover array languages. *Adv. Int. Soft Computing*, 89:291 – 298, 2011.

[28] M. Mateescu, Gh. Păun, G. Rozenberg, and A. Salomaa. Simple splicing systems. *Discrete Applied Math. 84*, 84:145–163, 1998.

[29] V. Mitrana. contextual insertion and deletion. *The publishing of house of the Romanian Academic*, Mathematical Linguistre and related topic(Gh.Paun ed):271–278, 1994.

[30] V. Mitrana, K. G. Subramanian, and M. Tataram. Pictural networks of evolutionary processors. *RomJIST*, 6(1-2):189–199, 2003.

[31] R. Narasimhan. Labelling schemata and syntactic description of pictures. *Information and Control*, 7:151–179, 1964.

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

[33] G. Siromoney, R. Siromoney, and K. Krithivasan. Abstract families of matrices and picture languages, *computer graphics and image processing* 1 (1972) 234 - 307.

[34] K. G. Subramanian. *Studies in array languages*. PhD thesis, University of Madras, 1979.

[35] K. G. Subramanian and Anthonath Roslin Sagaya Mary. On special forms of splicing on arrays and graphs. *Triangle*, 2011.

[36] K. G. Subramanian, Anthonath Roslin Sagaya Mary, and K. S. Dersanambika. Splicing array grammar systems. *ICTAC*, pages 125–135, 2005.

[37] K. G. Subramanian, R. Siromoney, V. R. Dare, and A. Saoudi. Basic puzzle languages. *Int. J. Pattern Recognition and Artificial Intelligence*, 9:763–775, 1995.

[38] P. S. Wang. Array grammars, patterns and recognizers. *World Scientific Pub. Co.*, 1989.

[39] P. S. Wang and C. Cook. A chomsky hierarchy of isotonic array grammars and languages. *Computer Graphics and Image Processing*, 8:144 – 152, 1978.

[40] E. Winfree. Algorithmic self-assembly of DNA: theoretical motivations and 2D assembly experiments. *Journal of Biomolecular Structures and Dynamics*, 11:263–270, 2000.

Chapter 2

Preliminaries

This chapter contains the pre-requisites of the study to follow in the next chapters. Based on formal language theory all basics of literature can be referred in [6]. We mainly give prerequisites of Picture languages and also string case Splicing languages. The later is given just to refer the generalization of the string case definitions to the new definitions of the study in Picture languages.

That is firstly, we give the notations of picture languages (two-dimensional languages) followed by the definitions of language classes studied with some examples. Then we state the definitions of string case Splicing language classes that we have tried to generalize to picture languages.

In particular we give in the section 2.1 the definitions and examples of the most important classes of picture languages used in our study (that are the $2D$ Matrix Grammars or $2D$ Right Linear Grammar, Basic Puzzle Grammars, Local Languages and the Tiling System Recognizable Languages, Wang Systems, in [9], [3], [8, 10], [2]).

Section 2.2 states the definitions of string case Splicing languages. Namely, H-Splicing Systems, Self Cross-Over Languages and Simple Splicing Languages in [4, 7, 1].

2.1 Pictures, Picture languages

2.1.1 Notations

Let Σ be a finite alphabet. Σ^* is the set of all words over Σ including the empty word λ. An image or a picture p over Σ is a rectangular $n \times m$ array of elements in Σ of the form

$$p = \boxed{\begin{array}{ccc} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{array}}$$

or in short we write a picture $p = [a_{i,j}]_{n\times m}$ where it is without enclosing in square brackets when there is no confusion. The set of all pictures is denoted by Σ^{**}. Also, the notation V is used for denoting the set of alphabet instead of Σ in some formalisms in the following chapters but is mentioned explicitly.

The *size* of the picture p is $n \times m$ or a pair (n, m) where n is the number of rows and m the number of columns of p. The i-th row, the j-th column and the element belonging to both of them in picture p will be denoted by $p^r[i]$, $p^c[j]$ and $p[i, j]$ (or p_{ij}) respectively. Moreover, by $p^r[i..i + k]$ ($p^c[i..i + k]$, respectively) we denote the sub-array of p consisting of the rows (columns, respectively) of p from index i to $i + k$.

The only picture of size $(0, 0)$ is the *empty picture*, denoted by λ. Then Σ^{++} denotes the set of all nonempty pictures over Σ, and define $\Sigma^{**} = \Sigma^{++} \cup \{\lambda\}$.

The *bordered* version of a picture p of size (n, m) is the array $\hat{p}$, i.e. of size $(n + 2, m + 2)$ obtained by surrounding p with special symbols in $\Delta = \{\#, \diamond\}$. Since the alphabet Δ consists of two symbols, we call *canonical* pictures those bordered uniquely with the boundary symbol #.

Given a picture p of size (n, m), a *partial bordered* version of p, or simply a *partial picture*, is the picture $\bar{p}$ of size either $(n', m + 2)$ or $(n + 2, m')$ for $n' = n + 1, m' = m + 1$, obtained from p by adding borders partially along the picture (see example 2.1).

Observe that a partial bordered picture is a bordered one that has a missing border only on one side of the picture.

In the chapter by *pseudo-canonical* picture we mean a picture that is completely bordered using also the symbol $\diamond$ or is partially bordered obtained from a picture p (without border). The notation $\hat{p}$ will be used for *pseudo-canonical* or canonical pictures that are obtained from picture p and partially bordered pictures $\bar{p}$. Moreover, observe that the size of a pseudo-picture is the size of the array over the extended alphabet $\Delta \cup \Sigma$.

Example 2.1. *Pseudo-canonical pictures : partially bordered pictures over*

one letter alphabet and (b) a completely bordered picture with the $\diamond$ symbol:

$$\begin{array}{|ccccc|}\hline \# & \# & \# & \# & \\ \# & a & a & a & \\ \# & a & a & a & \\ \# & a & a & a & \\ \# & \# & \# & \# & \\ \hline\end{array} \quad \begin{array}{|cccc|}\hline & \# & \# & \# & \# \\ & a & a & a & \# \\ & a & a & a & \# \\ & a & a & a & \# \\ & \# & \# & \# & \# \\ \hline\end{array} \quad (b) \begin{array}{|ccccc|}\hline \# & \# & \# & \# & \# \\ \# & a & a & a & \# \\ \# & a & a & a & \diamond \\ \# & a & a & a & \# \\ \# & \# & \# & \# & \# \\ \hline\end{array}$$

$$\begin{array}{|ccccc|}\hline \# & \# & \# & \# & \# \\ \# & a & a & a & \# \\ \# & a & a & a & \# \\ \diamond & a & a & a & \# \\ \hline\end{array} \quad \begin{array}{|ccccc|}\hline \# & a & a & a & \# \\ \# & a & a & a & \# \\ \# & a & a & a & \# \\ \# & \# & \# & \# & \# \\ \hline\end{array}$$

A *sub-picture* $\hat{p}'$ of a (pseudo-canonical) picture $\hat{p}$ is a picture that is a sub-array of $\hat{p}$. Given picture $\hat{p}$ then $B_{h,k}(\hat{p})$ denotes the set of sub-pictures of size (h,k). A *tile* $\begin{array}{|c|c|}\hline a & b \\ \hline c & d \\ \hline\end{array}$ also denoted by $t = \begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ is a $(2,2)$ picture over the alphabet $\Sigma \cup \Delta$, where tiles containing symbols from alphabet Δ are called *border tiles*. A *domino* is a $(1,2)$ or a $(2,1)$ picture denoted by $\begin{array}{|c|}\hline a \\ \hline b \\ \hline\end{array}$ a *column domino* or $\begin{array}{|c|c|}\hline a & b \\ \hline\end{array}$ a *row domino* for some $a, b \in \Sigma^* \cup \{\#\}$, where dominoes over the border symbol # are *border dominoes*.

A picture language or a two dimensional language over Σ is a subset of Σ^{**}.

$$\text{Let } p = \begin{array}{|ccc|}\hline a_{1,1} & \cdots & a_{1,i} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,i} \\ \hline\end{array}, \qquad q = \begin{array}{|ccc|}\hline b_{1,1} & \cdots & b_{1,m} \\ \vdots & \ddots & \vdots \\ b_{j,1} & \cdots & b_{j,m} \\ \hline\end{array}.$$

The ***column concatenation*** $p \oslash q$ of p and q is defined only when $n = j$ and is given by

$$p \oslash q = \begin{array}{|cccccc|}\hline a_{1,1} & \cdots & a_{1,i} & b_{1,1} & \cdots & b_{j,m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,i} & b_{j,1} & \cdots & b_{j,m} \\ \hline\end{array}.$$

Similarly, the ***row concatenation*** $p \ominus q$ of p and q is defined only when $i = m$ and is given by

$$p \ominus q = \begin{array}{|ccc|}\hline a_{1,1} & \cdots & a_{1,i} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,i} \\ b_{1,1} & \cdots & b_{1,m} \\ \vdots & \ddots & \vdots \\ b_{j,1} & \cdots & b_{j,m} \\ \hline \end{array}.$$

The column and row concatenation with *empty picture* λ is defined by $\lambda \oplus p = p \oplus \lambda = p$ and $\lambda \ominus p = p \ominus \lambda = p$.

The ***reflection of*** p ***on the base*** or the ***mirror image*** of p is defined as the picture

$$p_b = \begin{array}{|ccc|}\hline a_{n,1} & \cdots & a_{n,i} \\ \vdots & \ddots & \vdots \\ a_{1,1} & \cdots & a_{1,i} \\ \hline \end{array} = \begin{array}{|c|}\hline p^r[n] \\ p^r[n-1] \\ \vdots \\ p^r[1] \\ \hline \end{array} = mirror(p)$$

The above notion is used to define the palindromic column picture language. The ***reflection of*** p ***on the right leg*** is

$$p_{rl} = \begin{array}{|ccc|}\hline a_{1,i} & \cdots & a_{1,1} \\ \vdots & \ddots & \vdots \\ a_{n,i} & \cdots & a_{n,1} \\ \hline \end{array}$$

If L_1, L_2 are two picture languages over an alphabet Σ, the *column concatenation* $L_1 \oplus L_2$ of L_1 and L_2 is defined by

$$L_1 \oplus L_2 = \{A \oplus B \mid A \in L_1 \text{ and } B \in L_2\}.$$

The *row concatenation* $L_1 \ominus L_2$ of L_1 and L_2 is defined by

$$L_1 \ominus L_2 = \{A \ominus B \mid A \in L_1 \text{ and } B \in L_2\}.$$

2.1.2 Definitions and Examples

Grammars for picture languages

Definition 2.1. *A* $2D$ **Matrix grammar** *is a* $2-$ *tuple grammar* (G_1, G_2) *where* $G_1 = (H_1, I_1, P_1, S)$ *is a grammar,* $G_2 = (G_{21}, G_{22}, \cdots, G_{2k})$

wherein $G_{2i} = (V_{2i}, I_2, P_{2i}, S_i)\ \forall\ 1 \leq i \leq k$ *are regular grammars and are given by,*

H_1 : *a finite set of horizontal nonterminals,*
I_1 : $\{S_1, S_2, \cdots, S_k\}$, *a finite set of intermediates,* $H_1 \cap I_1 = \emptyset$,
P_1 : *a finite set of production rules called horizontal production rules,*
S : *the start symbol,* $S \in H_1$,
V_{2i} : *a finite set of vertical nonterminals,* $V_{2i} \cap V_{2j} = \emptyset,\ i \neq j$,
I_2 : *a finite set of terminals,*
P_{2i} : *a finite set of right linear production rules,*
S_i : *the start symbol.*

The grammar type of G_1 is the grammar type of G , i.e. we say that the given grammar is G regular, context-free, context sensitive, recursively enumerable $2D$ (two-dimensional) Matrix grammars if G_1 is regular, context-free, context sensitive or arbitrary respectively.

Derivations are defined as follows: First a string $S_{i1}S_{i2} \cdots S_{ik} \in I_1^*$ is generated horizontally using the horizontal production rules P_1 in G_1. That is, $S \Rightarrow S_{i1}S_{i2} \cdots S_{im} \in I_1^*$. Vertical derivations proceed as follows: We write

$$\begin{array}{c} A_{i1} \cdots A_{im} \\ \Downarrow \\ a_{i1} \cdots a_{im} \\ B_{i1} \cdots B_{im} \end{array}$$

if $A_{ij} \rightarrow a_{ij}B_{ij}$ are rules in $P_{2j}, 1 \leq j \leq m$. The derivation terminates if $A_j \rightarrow a_{mj}$ are all terminal rules in G_2. The set $L(G)$ of all matrices generated by G is defined to be $n \times m$ arrays $[a_{ij}]$ such that $1 \leq i \leq n,\ 1 \leq j \leq m$ and $S \Rightarrow^*_{G_1} S_{i1}S_{i2} \cdots S_{im} \Rightarrow^*_{G_2} [a_{ij}]$. If the type of G_1 is regular, we call G a Two-Dimensional Right-Linear Grammar and denote it by $2RLG$ [3]. If there exists a $2RLG$, G such that $L = L(G)$. $\mathcal{L}(2RLG)$ denotes the family of all picture languages generated by $2RLG$.

We recall the definition of a Puzzle Grammar introduced and studied in [8, 10].

Definition 2.2. *A* ***Basic Puzzle Grammar*** (BPG) *is a structure* $G = (N, T, R, S)$ *where* N *and* T *are finite sets of symbols;* $N \cap T = \phi$. *Elements of* N *are called non-terminals and elements of* T, *terminals.* $S \in N$ *is the start symbol or the axiom.* R *consists of rules of the following forms:*

$$A \to \textcircled{a}B,\quad A \to a\textcircled{B},\quad A \to B\textcircled{a},\quad A \to \textcircled{B}a$$
$$A \to \begin{matrix}\textcircled{a}\\ B\end{matrix},\quad A \to \begin{matrix}a\\ \textcircled{B}\end{matrix},\quad A \to \begin{matrix}B\\ \textcircled{a}\end{matrix},\quad A \to \begin{matrix}\textcircled{B}\\ a\end{matrix}$$

$A \to \textcircled{a}$,*where* $A,\ B,\ \in N$ *and* $a \in T$

Derivations begin with S written in a unit cell in the two-dimensional plane, with all the other cells containing the blank symbol #, not in $N \cup T$. In a derivation step, denoted $\to$, a non-terminal A in a cell is replaced by the right- hand member of a rule whose left-hand side is A. In this replacement, the circled symbol of the right-hand side of the rule used, occupies the cell to the right or the left or above or below the cell of the replaced symbol depending on the type of rule used. The replacement is possible only if the cell to be filled in by the non-circled symbol contains a blank symbol.

The set of pictures or figures generated by G, denoted by $L(G)$, is the set of connected, digitized finite arrays over T, derivable in one or more steps from the axiom.

Definition 2.3. *A Context-Free Puzzle Grammar* $(CFPG)$ *is a structure* $G = (N, T, R, S)$ *where* N, T, S *are as in definition 2.2 and* R *is the set of rewriting rules of the form* $A \to \alpha$*, where* α *is a finite, connected array of one or more cells, each cell containing either a nonterminal or a terminal symbol, with a symbol in one of the cells of* α *being circled.*

. The set of pictures generated by a context-free puzzle grammar G is defined analogous to definition 2.2

Example 2.2. *The BPG* $G_1 = (N, T, R, S)$ *where* $N = \{S, A, B\}$, $T = \{a\}$.

$$R = \{S \to \textcircled{a}S, S \to \textcircled{a}A, B \to \textcircled{a}S, B \to a, A \to \begin{matrix}A\\ \textcircled{a}\end{matrix}, A \to \begin{matrix}B\\ \textcircled{a}\end{matrix}$$

This BPG generates the picture language of pictures describing 'staircases' is shown below.

$$\begin{array}{ccccc} & & & & a \\ & & & & a \\ & & a & a & a \\ & & a & & \\ & & a & & \\ a & a & a & & \end{array}$$

Example 2.3. *Consider the Context Free Puzzle Grammar,* $G_2 = (N, T, R, S)$ *where*
$N = \{S, C, D, B\}$, $T = \{a\}$
$R = \{S \to \begin{matrix} C@D \\ B \end{matrix}, B \to \begin{matrix} @ \\ B \end{matrix}, C \to C@, D \to @D, C \to a\ ,$
$D \to a\ , B \to a\}$

This CFPG generates the picture language of pictures describing token-T shown below.

$$\begin{matrix} a & a & a & a & a & a \\ & & a & & & \\ & & a & & & \\ & & a & & & \end{matrix}$$

We now recall the notion of recognizability for picture languages where it takes as starting point a well known characterization of recognizable string languages in terms of local languages and projections. Namely, a defined Local picture language by means of a set of square arrays i.e. 2×2 pictures (or tiles) is Tiling system recognizable language when is obtained as a projection of Local picture language.

Tiling system recognizable languages

Definition 2.4. *A* picture language L *consists of a subset of* Σ^{**}. L *is* local *if there exists a finite set* Θ *of tiles over alphabet* $\Sigma \cup \{\#\}$ *such that* $L = \{p \in \Sigma^{**} | B_{2,2}(\hat{p}) \subseteq \Theta$ *where* $\hat{p}$ *are canonical pictures* $\}$. *Then* L *is the local language defined by* Θ.

We consider the set Θ as the set of all possible blocks of size 2×2 of pictures that belong to L. The language L is local if, given such a set Θ, we can exactly retrieve the language L and we write $L = L(\Theta)$. The empty picture λ belongs to L if and only if Θ contains the tile with four $\#$ symbols. The family of local picture languages is denoted by $\mathcal{L}(LOC)$.

Let us now recall the notion of a tiling system and language generated by such system [3].

Definition 2.5 (Tiling systems). *A Tiling system is a 4-tuple* $\tau = (\Sigma, \Gamma, \Theta, \pi)$, *where* Σ *and* Γ *are two finite alphabets,* Θ *is a finite set of tiles over the alphabet* $\Gamma \cup \{\#\}$ *and* $\pi : \Gamma \to \Sigma$ *is a projection.*

Given system τ, *the language defined by the system, denoted* $L(\tau)$, *is the projection by* π *of the local language defined by* Θ.

In this study we denote by $\mathcal{L}(\mathrm{TS})$ the class of languages defined by tiling systems, known as the class of recognizable languages, that has been deeply investigated in [3].

A further type of tiling systems we are interested in is that of Wang systems in [2].

Wang systems

Definition 2.6. *A labelled Wang tile consists of 4 colors, chosen in a finite set of colors Q, and of a label taken from a finite alphabet Γ, with the label at the center, can be represented as*

$$\begin{array}{ccc} & C_a & \\ C_b & (l) & C_c \\ & C_d & \end{array} .$$

A Wang system is a triple $W = (\Gamma, Q, T)$, where T is a set of labelled Wang tiles. In Q there has to be a particular color B.

Informally, given a Wang system W, a picture M over its set of tiles is a *tiling* if the color B is exactly present only along the boundary of the picture, and if for each pair of adjacent tiles in the picture the two colors presented by them on the touching side are identical. (See [2] for a formal definition.)

We say that a Wang system W generates the set of pictures of Γ^{**} defined by the sets of labels of its tilings: each tiling corresponds to a picture having the same dimension, and having as symbol in position (i, j) the label of the tile at the same position in the tiling.

The family of all picture languages generated by Wang systems is denoted by $\mathcal{L}(\mathrm{WS})$.

2.2 Splicing

We first recall the notion of splicing in string case introduced by Tom Head, which has been adapted to Picture Languages in H-array Splicing Systems defined in [5].

Definition 2.7 (Splicing on strings)**.** *Let V be an alphabet. A splicing rule over V is a string of the form $r = u_1; u_2 : u_3; u_4$, where $u_i \in V^*, 1 \leq i \leq 4$. For such a rule r and strings $x, y, z \in V^*$, we write*

$(x, y) \vdash_r z$ *iff* $x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,\ z = x_1u_1u_4y_2$

for some $x_1, x_2, y_1, y_2 \in V^*$. We say that z is obtained by splicing x, y, as indicated by the rule r; u_1u_2 and u_3u_4 are called the *sites* of the splicing.

Definition 2.8. *An H* **scheme** *is a pair $\sigma = (V, R)$, where V is an alphabet and $R \subseteq V^*;V^* : V^*;V^*$ is a set of splicing rules. For a given H scheme $\sigma = (V, R)$ and a language $L \subseteq V^*$, we define*
$\sigma(L) = \{z \in V^* \ / \ (x, y) \vdash_r z \text{ for some } x, y \in L,\ r \in R\ \}$

We can apply σ to L iteratively and obtain

$$\begin{aligned} \sigma^\circ(L) &= L, \\ \sigma^{i+1}(L) &= \sigma^i(L) \cup \sigma(\sigma^i(L)), i \geq 0, \\ \sigma^*(L) &= \bigcup_{i \geq 0} \sigma^i(L). \end{aligned}$$

Definition 2.9 ($2D$ Splicing system)**.** *$S = (\Sigma, I, B, f)$, where $\Sigma = A \cup A', A \cap A' = \emptyset$, A is the alphabet, A' is the set of special symbols, $f : A' \to A$ is a mapping, I is the set of initial images, $B = (B_1, B_2, B_3, B_4)$, where B_i is the set of Type-i patterns. A pattern p is a 9-tuple $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$ where $x_1, x_2, x_3, x_4, x_6, x_7, x_8, x_9 \in \Sigma^{**}$, $x_5 \in \Sigma^{++}$ subject to the condition that p is a proper sequence of cardinality$(3, 3)$. The middle term x_5 is called the crossing of p. The matrix image of p is called the site of the pattern p.*

Four types of splicing operations are defined for images and a splicing operation between two images is uniquely specified by giving the two images, the type of splicing and two matrix splits of cardinality $(5, 5)$, one for each of the two images. The result of the splicing operation consists of two resultants . For the splicing to take place , certain conditions have to be satisfied.

A splicing system S is said to be null-context if all the patterns of S are of the form $(\Lambda, \Lambda, \Lambda, \Lambda, c, \Lambda, \Lambda, \Lambda, \Lambda)$, and $c \in \Sigma^{++}$. In such a system, the crossing itself is a site. The patterns of this form are called null-context patterns.

Definition 2.10 (A self cross-over system)**.** *is a triple (V, A, R) where V is an alphabet, A is a finite subset of V^*, R is a finite commutative relation , $R \subset (V^* \times V^*)^2$; With respect to a self cross-over system, we define for $x \in V^+$*

$x \bowtie y$ if and only if $x = x_1\alpha\beta x_2 = x_3\gamma\delta x_4$; $y = x_1\alpha\delta x_4$; $(\alpha, \beta)R(\gamma, \delta)$.

$\bowtie^*$ is the reflexive and transitive closure of the relation $\bowtie$. The language generated by a self cross-over system is $L(SCO) = \{x \in V^* \ / \ w \bowtie^* x, w \in A\}$.

We first recall the notion of simple splicing on words [7]. For notions of language theory we refer to [6].

Definition 2.11. *Let V be an alphabet. $\$, \#$ are two special symbols, not in V.* A simple splicing rule *over V is a string of the form $r = a\# \ \lambda \ \$ \ a\# \ \lambda$, where $a \in V$. For such a rule r and strings $x, y, z \in V^*$, we write $(x, y) \vdash_r z$ if and only if $x = x_1ax_2$, $y = y_1ay_2$, $z = x_1ay_2$ for some $x_1, x_2, y_1, y_2 \in V^*$. We say that z is obtained by splicing x, y, as indicated by the rule r.*

Definition 2.12. *A* simple splicing scheme *is a pair $\sigma = (V, R)$, where V is an alphabet and R is a set of simple splicing rules. A simple splicing scheme is also referred to as a simple H scheme [7]. For a given simple splicing scheme $\sigma = (V, R)$ and a language $L \subseteq V^*$, we define $\sigma(L) = \{z \in V^* \ / \ (x, y) \vdash_r z$ for some $x, y \in L$, $r \in R\}$*

Chapter References

[1] J. Dassow and V. Mitrana. Self cross-over systems. in : Computing with bio - molecules: Theory and experiments. *(Eds.) Gh. Paun, Discrete Mathematics and Theoretical Computer Science*, Springer-Verlag:pp. 283 – 294, 1998.

[2] L. De Prophetis and V. Varricchio. Recognizability of rectangular pictures by wang systems. *Journal of Automata, Language and Combinatorics*, 2:269–288, 1997.

[3] D. Giammarresi and A. Restivo. *Two-dimensional languages, In "Handbook of Formal Languages"*, volume Springer Verlag,3. Springer Verlag, Vol.3, 1997.

[4] T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. *Bull. Math. Biology*, 49:737 –759, 1987.

[5] P. Helen Chandra, K. G. Subramanian, D. G. Thomas, and D. L. Van. A note on parallel splicing on images. In *IWCIA-2001*, volume Proceedings of the 8th International Workshop on Combinatorial Image Analysis, 2001.

[6] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory,*. Languages and Computation, Reading, MA,, 1979.

[7] M. Mateescu, Gh. Păun, G. Rozenberg, and A. Salomaa. Simple splicing systems. *Discrete Applied Math. 84*, 84:145–163, 1998.

[8] M. Nivat, A. Saoudi, K. G. Subramanian, R. Siromoney, and V. R. Dare. Puzzle grammars and context-free array grammars. *IJPRAI*, 5:663–675, 1992.

[9] G. Siromoney, R. Siromoney, and K. Krithivasan. Abstract families of matrices and picture languages, *computer graphics and image processing* 1 (1972) 234 - 307.

[10] K. G. Subramanian, R. Siromoney, V. R. Dare, and A. Saoudi. Basic puzzle languages. *IJPRAI*, 9:763–775, 1995.

Chapter 3

H-array Splicing System

This Chapter contains the defined formalism H-array Splicing System (HAS) introduced as a generalization of H-Splicing from the string case to Picture languages mainly in [5] with comparisons with other Picture language classes. Then the study follows extending with its restriction languages from same inspiration of the string case. Namely, Self-Cross over Array Languages, Simple H-array Splicing Systems in [9, 14].

The formalism HAS is described by set of domino splicing rules defined over the columns and rows of pictures, to cut a specific column(row) by the given domino splicing rules, of two given pictures in axiom and column(row) concatenate the first picture with the next. Thus making a simple extension of splicing operation to picture languages.

Then such Splicing operations defined in pictures is studied also in problem solving formalisms like Grammar Systems and Network of Evolutionary Processors. i.e. we give new formalisms namely, Splicing Array Grammar Systems (SAGS) in [15] and Pictural Network of Evolutionary Processors in [3]. SAGS is formalized with the Splicing operation in group or set of pictures in parallel, i.e. by deriving the formalism more on generalizing the Parallel Grammar Systems defined in formal language theory.

Section 3.1 states all the definitions of the introduced formalism on H-array Splicing Systems and its restriction languages with descriptions. Then the section 3.2 gives the examples and counter examples followed by the subsections on each language class comparisons.

Section 3.3 introduces a study on trees for the Simple Array Splicing Systems and its results. Section 3.4 gives the definition of Splicing Array Grammar Systems and its comparisons with regular, Context-free, Context-Sensitive languages for bi-dimensions explicitly stated in.

3.1 Definitions

Definition 3.1 (Domino splicing rules)**.** *A Domino column splicing rule is defined as* $r = \alpha_1 ; \alpha_2 : \alpha_3 ; \alpha_4$ *where each* α_k *for* $1 \leq k \leq 4$ *is a column domino over* $\Sigma^* \cup \{\#\}$.

A Domino row splicing rule is defined as $r = \beta_1 ; \beta_2 : \beta_3 ; \beta_4$ *where each* β_k *for* $1 \leq k \leq 4$ *is a row domino over* $\Sigma^* \cup \{\#\}$.

We refer to $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ of a column splicing rule $r = \alpha_1 ; \alpha_2 : \alpha_3 ; \alpha_4$ as the first, second, third and fourth dominoes of r respectively. Similarly, $\beta_1, \beta_2, \beta_3, \beta_4$ for a row splicing rule $r = \beta_1 ; \beta_2 : \beta_3 ; \beta_4$ are the first, second, third and fourth dominoes of r respectively.

Definition 3.2 (H-array splicing rules)**.** *The set of domino column splicing rules* R_c *given by,*

$$r_i = \begin{array}{|c|}\hline a_{i,j} \\ \hline a_{i+1,j} \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline a_{i,j+1} \\ \hline a_{i+1,j+1} \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline b_{i,j'} \\ \hline b_{i+1,j'} \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline b_{i,j'+1} \\ \hline b_{i+1,j'+1} \\ \hline \end{array}$$

and the set of domino row splicing rules R_r *given by,*

$$r_j = \begin{array}{|c|c|}\hline a_{i,j} & a_{i,j+1} \\ \hline \end{array} ; \begin{array}{|c|c|}\hline a_{i+1,j} & a_{i+1,j+1} \\ \hline \end{array} : \begin{array}{|c|c|}\hline b_{i',j} & b_{i',j+1} \\ \hline \end{array} ; \begin{array}{|c|c|}\hline b_{i'+1,j} & b_{i'+1,j+1} \\ \hline \end{array}$$

for all i, i', j, j' *such that* $0 \leq i \leq n,\ 0 \leq i' \leq n',\ 0 \leq j \leq m,\ 0 \leq j' \leq m'$ *are called H-array splicing rules.*

Given two pictures $\hat{p} = [a_{ij}]_{n \times m}$ and $\hat{q} = [b_{ij'}]_{n \times m'}$,

$$p = \begin{matrix} a_{11} & \cdots & a_{1,j} & a_{1,j+1} & \cdots & a_{1m} \\ a_{21} & \cdots & a_{2,j} & a_{2,j+1} & \cdots & a_{2m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{n,j} & a_{n,j+1} & \cdots & a_{nm} \end{matrix} ,$$

$$q = \begin{matrix} b_{11} & \cdots & b_{1,j'} & b_{1,j'+1} & \cdots & b_{1m'} \\ b_{21} & \cdots & b_{2,j'} & b_{2,j'+1} & \cdots & b_{2m'} \\ \vdots & \ddots & \cdots & \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{n,j'} & b_{n,j'+1} & \cdots & b_{nm'} \end{matrix}$$

$a_{ij}, b_{ij'} \in \Sigma$, for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq v' \leq m'$. As explained in chapter 2, $\hat{p}$ and $\hat{q}$ are the bordered pictures or arrays of sizes $(n+2) \times (m+2)$, $(n+2) \times (m'+2)$ obtained by surrounding p and q with # symbols, as shown below.

$$\hat{p} = \begin{array}{cccccccc} \# & \# & \cdots & \# & \# & \cdots & \# & \# \\ \# & a_{11} & \cdots & a_{1,j} & a_{1,j+1} & \cdots & a_{1m} & \# \\ \# & a_{21} & \cdots & a_{2,j} & a_{2,j+1} & \cdots & a_{2m} & \# \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \# & a_{n1} & \cdots & a_{n,j} & a_{n,j+1} & \cdots & a_{nm} & \# \\ \# & \# & \cdots & \# & \# & \cdots & \# & \# \end{array},$$

$$\hat{q} = \begin{array}{cccccccc} \# & \# & \cdots & \# & \# & \cdots & \# & \# \\ \# & b_{11} & \cdots & b_{1,j'} & b_{1,j'+1} & \cdots & b_{1m'} & \# \\ \# & b_{21} & \cdots & b_{2,j'} & b_{2,j'+1} & \cdots & b_{2m'} & \# \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \# & b_{n1} & \cdots & b_{n,j'} & b_{n,j'+1} & \cdots & b_{nm'} & \# \\ \# & \# & \cdots & \# & \# & \cdots & \# & \# \end{array}$$

i.e. by bordered arrays we refer to the top and bottom rows of the arrays with border symbol # as the 0^{th} and $n+1^{th}$ row. Similarly for the leftmost and rightmost columns with symbol # as the 0^{th} and $m+1^{th}$ column.

Firstly, we describe an application of the defined sequence of rules R_c (set of domino column splicing rules) in the given set of H-array splicing rules. Applying R_c to any two arrays $\hat{p}$ and $\hat{q}$ to yield an array $\hat{s}$ is described as below. We write $(p,q) \ |\!\underline{\oplus} \ s$ iff there exists a sequence of rules $R_c = r_0, r_1, r_2, ...r_n$ (not necessarily all different) such that for all $i,\ 0 \leq i \leq n$ and for some $j, j'\ 0 \leq j \leq m+1,\ 0 \leq j' \leq m'+1$:

$$\hat{s} = \begin{array}{cccccccc} \# & \# & \cdots & \# & \# & \cdots & \# & \# \\ \# & a_{11} & \cdots & a_{1,j} & b_{1,j'+1} & \cdots & b_{1m'} & \# \\ \# & a_{21} & \cdots & a_{2,j} & b_{2,j'+1} & \cdots & b_{2m'} & \# \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \# & a_{n1} & \cdots & a_{n,j} & b_{n,j'+1} & \cdots & b_{nm'} & \# \\ \# & \# & \cdots & \# & \# & \cdots & \# & \# \end{array}$$

In other words, we can imagine that a 2×1 window is moved down the j^{th} column of $\hat{p}$. The sequence of dominoes collected are the first dominoes of the rules $r_0, r_1, r_2, ..., r_m$ (not all necessarily different). When a 2×1 window is moved down the $j+1^{th}$ column of $\hat{p}$, the sequence of dominoes collected are the second dominoes of the rules $r_0, r_1, r_2, ..., r_m$. Likewise for the $j'th$ and $j'+1^{th}$ columns of $\hat{q}$. When such rules exist in the system, the column splicing of the arrays p and q amounts to the array $\hat{p}$ being

vertically "cut" between j^{th} and $j+1^{th}$ columns and the array $\hat{q}$ between j'^{th} and $j'+1^{th}$ columns and the resulting left subarray of $\hat{p}$ "pasted" (column catenated) with the right subarray of $\hat{q}$ to yield $\hat{s}$. We now say that s is obtained from p and q by *domino column splicing in parallel*, where s is $\hat{s}$ with surrounding # symbols deleted.

We can similarly define row splicing operation on any two arrays p and q of sizes $n \times m$ and $n' \times m'$, using domino row splicing rules which are H-array splicing rules to yield an array t.

$$p = \begin{matrix} a_{11} & a_{12} & \cdots & a_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i,1} & a_{i,2} & \cdots & a_{i,m} \\ a_{i+1,1} & a_{i+1,2} & \cdots & a_{i+1,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{matrix}, \quad q = \begin{matrix} b_{11} & b_{12} & \cdots & b_{1m'} \\ \vdots & \vdots & \ddots & \vdots \\ b_{i',1} & b_{i',2} & \cdots & b_{i',m'} \\ b_{i'+1,1} & b_{i'+1,2} & \cdots & b_{i'+1,m'} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{n'm'} \end{matrix}$$

$a_{ij}, b_{i'j} \in \Sigma$, for $1 \leq i \leq n$, $1 \leq i' \leq n'$, $1 \leq j \leq m$. Similarly considering $\hat{p}$ and $\hat{q}$ the bordered arrays of sizes $(n+2)\times(m+2)$, $(n'+2)\times(m'+2)$ obtained by surrounding p and q with # symbols.

We write $(p,q) \ |^{\ominus} \ t$ iff there is a sequence of row splicing rules $r_0, r_1, r_2, \cdots r_n$ (not necessarily all different) such that for all j, $0 \leq j \leq n$ and for some i, i' $0 \leq i \leq n+1$, $0 \leq i' \leq n'+1$:

$$\hat{t} = \begin{matrix} \# & \# & \# & \cdots & \# & \# \\ \# & a_{11} & a_{12} & \cdots & a_{1m} & \# \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \# & a_{i,1} & a_{i,2} & \cdots & a_{i,m} & \# \\ \# & b_{i'+1,1} & b_{i'+1,2} & \cdots & b_{i'+1,m} & \# \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \# & b_{n1} & b_{n2} & \cdots & b_{nm} & \# \\ \# & \# & \# & \cdots & \# & \# \end{matrix}$$

As done for the column splicing of arrays, we can imagine 1×2 windows being moved over respective rows. The row splicing of the arrays p and q can be thought of as $\hat{p}$ being horizontally "cut" between the i^{th} and $i+1^{th}$ rows and $\hat{q}$ between i'^{th} and $i'+1^{th}$ rows and the upper subarray of $\hat{p}$ "pasted" (row catenated) to the lower subarray of $\hat{q}$ to yield $\hat{t}$. We now say that t is obtained from p and q by *domino row splicing in parallel*,

where t is $\hat{t}$ with # symbols deleted.

We now introduce an H-array scheme and an H-array splicing system.

Definition 3.3. *An* H-array scheme is a triplet $\Gamma = (\Sigma, R_c, R_r)$ where Σ is an alphabet, R_c = a finite set of domino column splicing rules, and R_r = a finite set of domino row splicing rules.

For a given H-array scheme $\Gamma = (\Sigma, R_c, R_r)$ and a language $L \subseteq \Sigma^{**}$, we define

$\Gamma(L) = \{s, t \in \Sigma^{**} / (\hat{p}, \hat{q}) \mid^{\textcircled{1}} \hat{s}$ or $(\hat{p}, \hat{q}) \mid^{\ominus} \hat{t}$ for some $p, q, s, t \in L\}$.

In other words, $\Gamma(L)$ consists of arrays obtained by column or row splicing any two arrays of L using the array column or row splicing rules. Iteratively we define

$$\begin{aligned} \Gamma^{\circ}(L) &= L \\ \Gamma^{i+1}(L) &= \Gamma^{i}(L) \cup \Gamma(\Gamma^{i}(L)), i \geq 0 \\ \Gamma^{*}(L) &= \bigcup_{i \geq 0} \Gamma^{i}(L) \end{aligned}$$

An *H-array splicing system* HAS is defined by $S = (\Gamma, I)$ where $\Gamma = (\Sigma, R_c, R_r)$ and I is a finite subset of Σ^{**}. The language of S is defined by $L(S) = \Gamma^{*}(I)$ and we call it an *H-array splicing language* HASL and denote the class of such languages by $\mathcal{L}$(HASL) We note that in this chapter we consider I only as a finite subset of Σ^{**} although I can be an infinite subset.

Definition 3.4 (A Self cross-over array system (A_{SCO}))**.** *A Self cross-over array system denoted as A_{SCO} is defined by $S = (\Omega, I)$ where $\Omega = (\Sigma, R_c, R_r)$ is a H-array splicing scheme and I is a finite subset of Σ^{**} and the set of domino splicing rules is applied each time to two identical copies of the same array.*

A Self cross-over array language is defined as in the case of linear strings and we denote the class of Self cross-over array languages by $\mathcal{L}(A_{SCO})$.

We now introduce the notions of simple domino splicing rules and simple array splicing Systems (SAS). These systems are a special kind of H array splicing systems with restriction introduced and studied in [2].

Definition 3.5 (Simple domino splicing rules)**.** *A Simple column or row domino splicing rule is a column or row domino splicing rule of the form* $r = \alpha_1; \lambda : \alpha_3; \lambda$ *or* $r = \beta_1; \lambda : \beta_3; \lambda$ *respectively and are called Simple domino splicing rules.*

In other words in Simple domino splicing rules $\alpha_2 = \alpha_4 = \lambda$ and $\beta_2 = \beta_4 = \lambda$ of a column domino splicing rule or row domino splicing rule respectively. Note that $\lambda = \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$ or $\lambda = \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}$ in domino column splicing rules or domino row splicing rules respectively.

Definition 3.6 (Simple array splicing rules)**.** *The set of Simple column domino splicing rules* R_c *given by,*

$$r_i = \begin{array}{|c|}\hline a_{i,j} \\ \hline a_{i+1,j} \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline b_{i,j'} \\ \hline b_{i+1,j'} \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$

and the set of Simple row domino splicing rules R_r *given by,*

$$r_j = \begin{array}{|c|c|}\hline a_{i,j} & a_{i,j+1} \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline b_{i',j} & b_{i',j+1} \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}$$

for all i, i', j, j'; $0 \leq i \leq n+1$, $0 \leq i' \leq n'+1$, $0 \leq j \leq m+1$, $0 \leq j' \leq m'+1$ *are called the Simple array splicing rules.*

For the given two arrays p and q of sizes $n \times m$ and $n \times m'$ respectively,

$$p = \begin{matrix} a_{1,1} & \cdots & a_{1,j} & \cdots & a_{1,m} \\ a_{2,1} & \cdots & a_{2,j} & \cdots & a_{2,m} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,j} & \cdots & a_{n,m} \end{matrix} ,$$

$$q = \begin{matrix} b_{1,1} & \cdots & b_{1,j'} & \cdots & b_{1,m'} \\ b_{2,1} & \cdots & b_{2,j'} & \cdots & b_{2,m'} \\ \vdots & \ddots & \cdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,j'} & \cdots & b_{n,m'} \end{matrix}$$

$a_{i,j}$, $b_{i,j'} \in \Sigma$, for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq j' \leq m'$. Let $\hat{p}$ and $\hat{q}$ be bordered arrays of sizes $(n+2) \times (m+2)$, $(n'+2) \times (m'+2)$ obtained by surrounding p and q with # symbols, as shown below.

$$\hat{p} = \begin{array}{ccccccc} \# & \# & \cdots & \# & \cdots & \# & \# \\ \# & a_{1,1} & \cdots & a_{1,j} & \cdots & a_{1,m} & \# \\ \# & a_{2,1} & \cdots & a_{2,j} & \cdots & a_{2,m} & \# \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ \# & a_{n,1} & \cdots & a_{n,j} & \cdots & a_{n,m} & \# \\ \# & \# & \cdots & \# & \cdots & \# & \# \end{array},$$

$$\hat{q} = \begin{array}{ccccccc} \# & \# & \cdots & \# & \cdots & \# & \# \\ \# & b_{1,1} & \cdots & b_{1,j'} & \cdots & b_{1,m'} & \# \\ \# & b_{2,1} & \cdots & b_{2,j'} & \cdots & b_{2,m'} & \# \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ \# & b_{n,1} & \cdots & b_{n,j'} & \cdots & b_{n,m'} & \# \\ \# & \# & \cdots & \# & \cdots & \# & \# \end{array}$$

We write $(p, q) \ |\!\underline{\oplus} \ z$ if there is a sequence of simple column splicing rules $r_0, r_1, r_2, ...r_n$ (not necessarily all different) such that and $a_{i,j} = b_{i,j'}$ for all $i,\ 0 \leq i \leq n$ and for some $j, j'\ 0 \leq j \leq m+1,\ 0 \leq k \leq m'+1$ and

$$\hat{z} = \begin{array}{ccccccc} \# & \# & \cdots & \# & \# & \cdots & \# & \# \\ \# & a_{11} & \cdots & a_{1,j} & b_{1,j'} & \cdots & b_{1m'} & \# \\ \# & a_{21} & \cdots & a_{2,j} & b_{2,j'} & \cdots & b_{2m'} & \# \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \# & a_{n1} & \cdots & a_{n,j} & b_{n,j'} & \cdots & b_{nm'} & \# \\ \# & \# & \cdots & \# & \# & \cdots & \# & \# \end{array}$$

In other words, we can imagine that a 2×1 window is moved down the j^{th} column of $\hat{p}$. The sequence of dominoes collected are the first dominoes of the rules $r_0, r_1, r_2, ..., r_n$ (not all necessarily different). Likewise for the j'^{th} column of $\hat{q}$ except that the dominoes collected are the third dominoes of the rules. When such rules exist in the system, the simple column splicing of the arrays p and q amounts to the array $\hat{p}$ being vertically "cut" after j^{th} column and the array $\hat{q}$ after j'^{th} column and the resulting left subarray of $\hat{p}$ "pasted" (column catenated) with the right subarray of $\hat{q}$ to yield $\hat{z}$. We now say that z is obtained from p and q by *simple domino column splicing in parallel*, where z is $\hat{z}$ with surrounding # symbols deleted.

We can similarly define simple row splicing operation of the two arrays p and q of sizes $n \times m$ and $n' \times m'$, using simple domino row splicing rules

to yield an array z'. We write $(p, q) \ |^{\ominus} \ z'$. As done for the column splicing of arrays, we can imagine 1×2 windows being moved over respective rows. The row splicing of the arrays p and q can be thought of as $\hat{p}$ being horizontally "cut" below the j^{th} row and $\hat{q}$ below j'^{th} row and the upper subarray of $\hat{p}$ "pasted" (row catenated) to the lower subarray of $\hat{q}$ to yield $\hat{z'}$. We now say that z' is obtained from p and q by *simple domino row splicing in parallel*, where z' is $\hat{z'}$ with # symbols deleted.

We now introduce the main notion of Simple array splicing systems.

Definition 3.7 (Simple array splicing scheme)**.** *A* Simple array splicing scheme *is a triplet* $\Gamma = (\Sigma, R_c, R_r)$ *where* Σ *is an alphabet,* R_c *= a finite set of simple domino column splicing rules, and* R_r *= a finite set of simple domino row splicing rules. For a given Simple array scheme* $\Gamma = (\Sigma, R_c, R_r)$ *and a language* $L \subseteq \Sigma^{**}$*, we define*
$\Gamma(L) = \{z, z' \in \Sigma^{**} \,/\, (\hat{p}, \hat{q}) \ |^{\oplus} \ \hat{z}$ *or* $(\hat{p}, \hat{q}) \ |^{\ominus} \ \hat{z'}$ *for some* $p, q, z, z' \in L\}$.

In other words, $\Gamma(L)$ consists of arrays obtained by column or row splicing any two arrays of L using the simple domino column or row splicing rules.

A *Simple array splicing system* SAS is defined by $S = (\Gamma, I)$ where $\Gamma = (\Sigma, R_c, R_r)$ and I is a finite subset of Σ^{**}. The language of S is defined by $L(S) = \Gamma^*(I)$ and we call it a *Simple array splicing language* SASL and denote the class of such languages by $\mathcal{L}$(SASL) .

We illustrate the above defined $\mathcal{L}$(HASL) and its restriction languages $\mathcal{L}(A_{SCO})$, $\mathcal{L}$(SASL) with examples in the section below.

3.2 Examples and counter examples

Below example describes HAS rules and its application.

Example 3.1. *Let* $\Sigma = \{a, b\}$, $\quad I = \left\{ \begin{array}{cccc} \# & \# & \# & \# \\ \# & a & b & \# \\ \# & b & a & \# \\ \# & \# & \# & \# \end{array} \right\}$,
$R_c = \{r_1, \cdots, r_6\}$ *and* $R_r = \{r_7, \cdots, r_{12}\}$ *given by,*

$$r_1 = \begin{array}{|c|}\hline a \\ \hline b \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline b \\ \hline a \\ \hline \end{array}$$

$$r_2 = \begin{array}{|c|}\hline b \\ \hline a \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline a \\ \hline b \\ \hline \end{array}$$

$$r_3 = \begin{array}{|c|}\hline \# \\ \hline a \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline b \\ \hline \end{array}$$

$$r_4 = \begin{array}{|c|}\hline \# \\ \hline b \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline a \\ \hline \end{array}$$

$$r_5 = \begin{array}{|c|}\hline a \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline b \\ \hline \# \\ \hline \end{array}$$

$$r_6 = \begin{array}{|c|}\hline b \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline a \\ \hline \# \\ \hline \end{array}$$

$$r_7 = \begin{array}{|c|c|}\hline a & b \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline b & a \\ \hline \end{array}$$

$$r_8 = \begin{array}{|c|c|}\hline b & a \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline a & b \\ \hline \end{array}$$

$$r_9 = \begin{array}{|c|c|}\hline \# & b \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \# & a \\ \hline \end{array}$$

$$r_{10} = \begin{array}{|c|c|}\hline \# & a \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \# & b \\ \hline \end{array}$$

$$r_{11} = \begin{array}{|c|c|}\hline b & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline a & \# \\ \hline \end{array}$$

$$r_{12} = \begin{array}{|c|c|}\hline a & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline b & \# \\ \hline \end{array}$$

On column splicing in parallel, the initial array I with itself using HAS rules r_2, r_4, r_5, we obtain the array which is a column concatenation of I with itself, i.e. given by,

$$\begin{array}{cccc} \# & \# & \# & \# \\ \# & a & b & \# \\ \# & b & a & \# \\ \# & \# & \# & \# \end{array} \;\; |\underline{\oplus} \;\; \begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & \# & \# & \# & \# & \# \end{array} .$$

In fact, the splicing operation can be described in pictures more elaborately as,

$$\left[\begin{array}{ccc|c|ccc} \# & \# & \# & \# & \# & \# & \# \\ \# & a & b & \# & a & b & \# \\ \# & b & a & \# & b & a & \# \\ \# & \# & \# & \# & \# & \# & \# \end{array}\right] \underset{|}{\overset{\oplus}{-}} \left[\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & \# & \# & \# & \# & \# \end{array}\right]$$

Likewise, row splicing in parallel using r_7, r_8, r_9, r_{12} gives

$$\left[\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \hline \# & \# & \# & \# & \# & \# \end{array}\right. , \left.\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & \# & \# & \# & \# & \# \end{array}\right] \underset{|}{\overset{\ominus}{-}} \left[\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & \# & \# & \# & \# & \# \end{array}\right]$$

and using r_7, r_8, r_9, r_{12} gives

$$\left[\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \hline \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & \# & \# & \# & \# & \# \end{array}\right. , \left.\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & \# & \# & \# & \# & \# \end{array}\right] \underset{|}{\overset{\ominus}{-}} \left[\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & a & b & a & b & \# \\ \# & b & a & b & a & \# \\ \# & \# & \# & \# & \# & \# \end{array}\right]$$

A vertical bar '| ' or a horizontal bar '—' is used to indicate the place where splicing is done. L is the language consisting of all "chessboards" with even side-length given in [4]. i.e. pictures of the form in Figure 3.1

The picture or pattern of Figure 3.1 can be represented by an array M, where 'a' stands for white and 'b' for black and

$$M = \begin{array}{cccccccccc} a & b & a & b & a & b & a & b & a & b \\ b & a & b & a & b & a & b & a & b & a \\ a & b & a & b & a & b & a & b & a & b \\ b & a & b & a & b & a & b & a & b & a \\ a & b & a & b & a & b & a & b & a & b \\ b & a & b & a & b & a & b & a & b & a \end{array}.$$

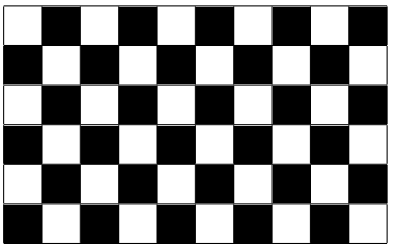


Figure 3.1: Chessboard Pattern

Now, we illustrate the above chessboard pattern example with SAS rules.

Example 3.2. *Let* $\Sigma = \{a, b\}$, $I = \begin{Bmatrix} a & b \\ b & a \end{Bmatrix}$, $R_c = \{r_1, .., r_4\}$ *and* $R_r = \{r_5, ..., r_8\}$ *given by,*

$$r_1 = \begin{bmatrix} b \\ a \end{bmatrix} ; \begin{bmatrix} \lambda \\ \lambda \end{bmatrix} : \begin{bmatrix} b \\ a \end{bmatrix} ; \begin{bmatrix} \lambda \\ \lambda \end{bmatrix}$$

$$r_2 = \begin{bmatrix} \# \\ b \end{bmatrix} ; \begin{bmatrix} \lambda \\ \lambda \end{bmatrix} : \begin{bmatrix} \# \\ b \end{bmatrix} ; \begin{bmatrix} \lambda \\ \lambda \end{bmatrix}$$

$$r_3 = \begin{bmatrix} a \\ \# \end{bmatrix} ; \begin{bmatrix} \lambda \\ \lambda \end{bmatrix} : \begin{bmatrix} a \\ \# \end{bmatrix} ; \begin{bmatrix} \lambda \\ \lambda \end{bmatrix}$$

$$r_4 = \begin{bmatrix} a \\ b \end{bmatrix} ; \begin{bmatrix} \lambda \\ \lambda \end{bmatrix} : \begin{bmatrix} a \\ b \end{bmatrix} ; \begin{bmatrix} \lambda \\ \lambda \end{bmatrix}$$

$$r_5 = \begin{bmatrix} a & b \end{bmatrix} ; \begin{bmatrix} \lambda & \lambda \end{bmatrix} : \begin{bmatrix} a & b \end{bmatrix} ; \begin{bmatrix} \lambda & \lambda \end{bmatrix}$$

$$r_6 = \begin{bmatrix} b & a \end{bmatrix} ; \begin{bmatrix} \lambda & \lambda \end{bmatrix} : \begin{bmatrix} b & a \end{bmatrix} ; \begin{bmatrix} \lambda & \lambda \end{bmatrix}$$

$$r_7 = \begin{bmatrix} \# & b \end{bmatrix} ; \begin{bmatrix} \lambda & \lambda \end{bmatrix} : \begin{bmatrix} \# & b \end{bmatrix} ; \begin{bmatrix} \lambda & \lambda \end{bmatrix}$$

$$r_8 = \begin{bmatrix} a & \# \end{bmatrix} ; \begin{bmatrix} \lambda & \lambda \end{bmatrix} : \begin{bmatrix} a & \# \end{bmatrix} ; \begin{bmatrix} \lambda & \lambda \end{bmatrix}$$

The above given sequence of rules enables the application at any column or row of the two spliced (identical) initial pictures. This is unlike for the HAS rules which were given in example 3.1 also obtaining picture language with pattern in figure 3.1.

The following are more examples of SASL .

Example 3.3. *Let* $\Sigma = \{x, a\}$, $I = \left\{ \begin{array}{cccc} x & x & x & x \\ x & a & a & x \\ x & a & a & x \\ x & x & x & x \end{array} \right\}$, $R_c = \{r_1, ..., r_5\}$ *and* $R_r = \{r_6, ..., r_{10}\}$ *given by,*

$$r_1 = \begin{array}{|c|}\hline x \\ \hline a \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline x \\ \hline a \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$

$$r_2 = \begin{array}{|c|}\hline a \\ \hline a \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline a \\ \hline a \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$

$$r_3 = \begin{array}{|c|}\hline a \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline a \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$

$$r_4 = \begin{array}{|c|}\hline \# \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline \# \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$

$$r_5 = \begin{array}{|c|}\hline x \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline x \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$

$$r_6 = \begin{array}{|c|c|}\hline x & a \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline x & a \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}$$

$$r_7 = \begin{array}{|c|c|}\hline a & a \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline a & a \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}$$

$$r_8 = \begin{array}{|c|c|}\hline a & x \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline a & x \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}$$

$$r_9 = \begin{array}{|c|c|}\hline \# & x \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline \# & x \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}$$

$$r_{10} = \begin{array}{|c|c|}\hline x & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline x & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}$$

L is the language consisting of pictures of the form in Figure 3.2, where white area of the rectangle is interpreted as 'a' and black as 'x'.

Example 3.4. *Let* $\Sigma = \{x, .\}$, $I = \left\{ \begin{array}{ccc} x & . & . \\ x & . & . \\ x & x & x \end{array} \right\}$,
$R_c = \{r_1, .., r_4\}$ *and* $R_r = \{r_5, ..., r_8\}$ *given by,*

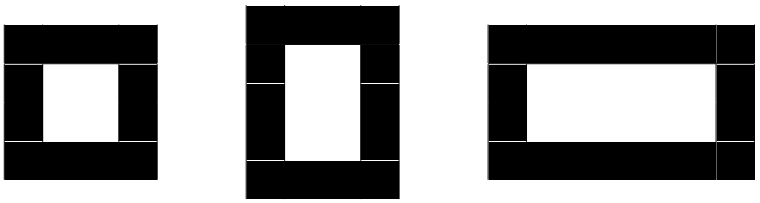


Figure 3.2: Rectangles of 'a' surrounded by 'x'

$$r_1 = \begin{bmatrix}\# \\ .\end{bmatrix} ; \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} : \begin{bmatrix}\# \\ .\end{bmatrix} ; \begin{bmatrix}\lambda \\ \lambda\end{bmatrix}$$

$$r_2 = \begin{bmatrix}. \\ .\end{bmatrix} ; \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} : \begin{bmatrix}. \\ .\end{bmatrix} ; \begin{bmatrix}\lambda \\ \lambda\end{bmatrix}$$

$$r_3 = \begin{bmatrix}. \\ x\end{bmatrix} ; \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} : \begin{bmatrix}. \\ x\end{bmatrix} ; \begin{bmatrix}\lambda \\ \lambda\end{bmatrix}$$

$$r_4 = \begin{bmatrix}x \\ \#\end{bmatrix} ; \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} : \begin{bmatrix}x \\ \#\end{bmatrix} ; \begin{bmatrix}\lambda \\ \lambda\end{bmatrix}$$

$$r_5 = \begin{bmatrix}\# & x\end{bmatrix} ; \begin{bmatrix}\lambda & \lambda\end{bmatrix} : \begin{bmatrix}\# & x\end{bmatrix} ; \begin{bmatrix}\lambda & \lambda\end{bmatrix}$$

$$r_6 = \begin{bmatrix}x & .\end{bmatrix} ; \begin{bmatrix}\lambda & \lambda\end{bmatrix} : \begin{bmatrix}x & .\end{bmatrix} ; \begin{bmatrix}\lambda & \lambda\end{bmatrix}$$

$$r_7 = \begin{bmatrix}. & .\end{bmatrix} ; \begin{bmatrix}\lambda & \lambda\end{bmatrix} : \begin{bmatrix}. & .\end{bmatrix} ; \begin{bmatrix}\lambda & \lambda\end{bmatrix}$$

$$r_8 = \begin{bmatrix}. & \#\end{bmatrix} ; \begin{bmatrix}\lambda & \lambda\end{bmatrix} : \begin{bmatrix}. & \#\end{bmatrix} ; \begin{bmatrix}\lambda & \lambda\end{bmatrix}$$

L is the picture language consisting of all $n \times m$ arrays describing token-L of 'x'.

Remark 3.1. *Note that we consider finite set of rules in the following results, though the rules can be infinite. Also, note that controlling the number of application of R_c and R_r in strict order can also obtain languages describing size and pattern of the rectangles/arrays. i.e. leaving intermediate pictures according to a defined strict application process of R_c and R_r can obtain the pictures in the defined language class with described size and pattern.*

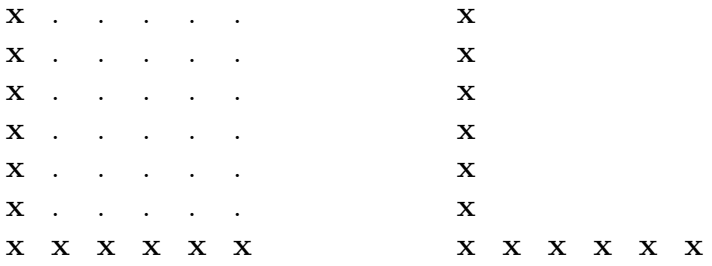


Figure 3.3: Arrays describing token L of 'x'

In the following sections we study some of the properties of the above defined language classes and its comparisons with some of the other given well studied language classes for recognizability in two-dimensional languages/picture languages. Firstly, we state the results of $\mathcal{L}$(HASL) studied in [6].

3.2.1 H-array splicing systems

Now we examine certain closure properties :

Theorem 3.1. *The class $\mathcal{L}$(HASL) is closed under reflections on the base and right leg and rotations by* $90°, 180°$ *and* $270°$.

Proof. We first prove that $\mathcal{L}$(HASL) is closed under reflections. Let $S = (\Gamma, I)$ where $\Gamma = (\Sigma, R_c, R_r)$ and I is a finite subset of Σ^{**} be a splicing system, with rules in R_c of the form

$$r_1 = \begin{array}{|c|}\hline a_1 \\ \hline b_1 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline c_1 \\ \hline d_1 \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline a_2 \\ \hline b_2 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline c_2 \\ \hline d_2 \\ \hline \end{array}$$

and in R_r of the form

$$r_2 = \begin{array}{|c|c|}\hline a_1 & b_1 \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline c_1 & d_1 \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline a_2 & b_2 \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline c_2 & d_2 \\ \hline \end{array}$$

describing a picture language L.

The picture language consisting of images which are reflections of arrays of L on the base can be obtained by an H array splicing system consisting of rules of the form

$$\begin{array}{|c|}\hline b_1 \\ \hline a_1 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline d_1 \\ \hline c_1 \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline b_2 \\ \hline a_2 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline d_2 \\ \hline c_2 \\ \hline \end{array}$$

corresponding to r_1 and rules of the form

$$\boxed{c_2}\boxed{d_2} \;;\; \boxed{a_2}\boxed{b_2} \;:\; \boxed{c_1}\boxed{d_1} \;;\; \boxed{a_1}\boxed{b_1}$$

corresponding to r_2.

Similarly, the reflections of arrays of L on the right leg can be obtained by an H array splicing system with modified rules

$$\begin{array}{|c|}\hline c_2 \\ \hline d_2 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline a_2 \\ \hline b_2 \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline c_1 \\ \hline d_1 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline a_1 \\ \hline b_1 \\ \hline \end{array}$$

and

$$\boxed{b_1}\boxed{a_1} \;;\; \boxed{d_1}\boxed{c_1} \;:\; \boxed{b_2}\boxed{a_2} \;;\; \boxed{d_2}\boxed{c_2}$$

corresponding to r_1 and r_2 respectively.

We next prove that $\mathcal{L}$(HASL) is closed under rotations by $90°, 180°$ and $270°$. We mention only the modified rules of R_c and R_r

$$\begin{array}{|c|}\hline c_2 \\ \hline d_2 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline a_2 \\ \hline b_2 \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline c_1 \\ \hline d_1 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline a_1 \\ \hline b_1 \\ \hline \end{array}$$

and

$$\boxed{b_1}\boxed{a_1} \;;\; \boxed{d_1}\boxed{c_1} \;:\; \boxed{b_2}\boxed{a_2} \;;\; \boxed{d_2}\boxed{c_2}$$

for rotation by $90°$;

$$\begin{array}{|c|}\hline d_2 \\ \hline c_2 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline b_2 \\ \hline a_2 \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline d_1 \\ \hline c_1 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline b_1 \\ \hline a_1 \\ \hline \end{array}$$

and

$$\boxed{d_2}\boxed{c_2} \;;\; \boxed{b_2}\boxed{a_2} \;:\; \boxed{d_1}\boxed{c_1} \;;\; \boxed{b_1}\boxed{a_1}$$

for rotation by $180°$;

$$\begin{array}{|c|}\hline b_1 \\ \hline a_1 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline d_1 \\ \hline c_1 \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline b_2 \\ \hline a_2 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline d_2 \\ \hline c_2 \\ \hline \end{array}$$

and

$$\boxed{c_2}\boxed{d_2} \;;\; \boxed{a_2}\boxed{b_2} \;:\; \boxed{c_1}\boxed{d_1} \;;\; \boxed{a_1}\boxed{b_1}$$

for rotation by $270°$. □

Theorem 3.2. *The class $\mathcal{L}(HASL)$ is not closed under union and row and column concatenation.*

Proof. Let $L_1 = \{(x^{2m})_3, m \geq 1\}$ and $L_2 = \{(x^{3m})_3, m \geq 1\}$, where $(x^{2m})_3$ is an array with 3 rows and $2m$ columns with each entry in the array as the symbol x and $(x^{3m})_3$ has a similar meaning. But on column splicing two initial arrays of the form

$$\begin{matrix} x & x \\ x & x \\ x & x \end{matrix}, \qquad \begin{matrix} x & x & x \\ x & x & x \\ x & x & x \end{matrix}$$

by "splicing" either inside the array or at ends and then "pasting" the resulting arrays according to any domino rules we will obtain arrays which will not be elements of $L_1 \cup L_2$.

Let L_1 be a language consisting of arrays with 3 rows and any number of columns with left border made of symbol 'a', right border of symbol 'b' and inner part of symbol 'x'. A member of L_1 is shown in figure 3.4. Similarly, let L_2 be another language of arrays as in L_1 but left border made of symbol 'c', right border of symbol 'd'. In order to obtain arrays of $L_1 \oplus L_2$ (a member of which is shown in figure 3.4), the column splicing of two arrays should maintain the leftmost column of symbol 'a', the rightmost column of symbol 'd' and two successive innermost column of symbols 'b' and 'c'. But this is not possible due to the "cutting" and "pasting" nature of rules. An analogous argument applies to row concatenation.

$$\begin{matrix} \mathbf{a} & x & x & x & x & x & \mathbf{b} \\ \mathbf{a} & x & x & x & x & x & \mathbf{b} \\ \mathbf{a} & x & x & x & x & x & \mathbf{b} \end{matrix} \qquad \begin{matrix} \mathbf{c} & x & x & x & x & x & \mathbf{d} \\ \mathbf{c} & x & x & x & x & x & \mathbf{d} \\ \mathbf{c} & x & x & x & x & x & \mathbf{d} \end{matrix} \qquad \begin{matrix} \mathbf{a} & x & \mathbf{b} & \mathbf{c} & x & \mathbf{d} \\ \mathbf{a} & x & \mathbf{b} & \mathbf{c} & x & \mathbf{d} \\ \mathbf{a} & x & \mathbf{b} & \mathbf{c} & x & \mathbf{d} \end{matrix}$$

Figure 3.4: a member/picture of L_1, L_2 and $L_1 \oplus L_2$

□

We now compare the generative power of H array splicing systems with other models of picture description.

Theorem 3.3. *The classes $\mathcal{L}(LOC)$ of local array languages and $\mathcal{L}(HASL)$ of H array splicing languages are incomparable but not disjoint.*

Proof. The picture language M consisting of all $n \times m$ arrays ($n \geq 2, m \geq 2$) describing token-L of symbol '1' (interpreting symbol '0' as blank) in figure 3.5 is in LOC. A member of M is shown in figure 3.5.

$$\begin{array}{cccccc} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{array} \qquad \begin{array}{c} \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \end{array}$$

Figure 3.5: pictures describing token-L of symbol '1'

We give an H array splicing system $S = (\Sigma, R_c, R_r, I)$ to describe M.

Let $\Sigma = \{0, 1\}$, $I = \left\{ \begin{array}{cc} 1 & 0 \\ 1 & 1 \end{array} \right\}$, $R_c = \{r_1, ..., r_4\}$ and $R_r = \{r_5, ..., r_8\}$ given by,

$$r_1 = \begin{array}{|c|}\hline 0 \\ \hline 1 \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline 0 \\ \hline 1 \\ \hline \end{array}$$

$$r_2 = \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array}$$

$$r_3 = \begin{array}{|c|}\hline \# \\ \hline 0 \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline \# \\ \hline 1 \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array}$$

$$r_4 = \begin{array}{|c|}\hline 1 \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline 1 \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array}$$

$$r_5 = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline 1 & 1 \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline 1 & 0 \\ \hline \end{array}$$

$$r_6 = \begin{array}{|c|c|}\hline 0 & 0 \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline 1 & 1 \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline 0 & 0 \\ \hline \end{array}$$

$$r_7 = \begin{array}{|c|c|}\hline \# & 1 \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \# & 1 \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \# & 1 \\ \hline \end{array}$$

$$r_8 = \begin{array}{|c|c|}\hline 0 & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline 1 & \# \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline 0 & \# \\ \hline \end{array}$$

The picture language L of all images over $\Sigma = \{a\}$ with 3 columns, some given in figure 3.6, is not in $\mathcal{L}(LOC)$. In fact, it is not possible to fix the number of columns using only one symbol. i.e. the block $\begin{bmatrix} a & a \\ a & a \end{bmatrix}$ can be moved without restriction on the columns. But L is obtained by an H array splicing system where $I = \{a \quad a \quad a\}$, $R_c = \emptyset$ and $R_r = \{r_1, r_2, r_3\}$ below,

$$r_1 = \boxed{a}\boxed{a} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{a}\boxed{a}$$

$$r_2 = \boxed{\#}\boxed{a} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{\#}\boxed{a}$$

$$r_3 = \boxed{a}\boxed{\#} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{a}\boxed{\#}$$

$$\begin{matrix} a & a & a \\ a & a & a \end{matrix} \qquad \begin{matrix} a & a & a \\ a & a & a \\ a & a & a \end{matrix} \qquad \begin{matrix} a & a & a \\ a & a & a \\ a & a & a \\ a & a & a \end{matrix}$$

Figure 3.6: 3 column pictures over $\Sigma = a$

□

It is known [4] that the picture language of square images in which diagonal positions carry symbol '1' but the remaining positions carry symbol '0' is in $\mathcal{L}(LOC)$. But it is not in HAS. Since row and column splicing are independently done, it is clear that arrays with a proportion between rows and columns and in particular pictures with only square size cannot be generated.

Remark 3.2. *The class $\mathcal{L}(HASL)$ intersects $\mathcal{L}(TS)$ since $\mathcal{L}(LOC) \subseteq \mathcal{L}(TS)$.*

Theorem 3.4. *The class $\mathcal{L}(HASL)$ of H array splicing languages and $\mathcal{L}(2RLG)$ of picture languages generated by two dimensional right linear grammar are incomparable but not disjoint.*

Proof. The picture language of "chessboards" with even side-length in example 3.1 is generated by a HAS, and is known in [4] to be generated by a $2RLG$.

The picture language L_1 consisting of arrays describing token-H cannot be generated by any $2RLG$, as the horizontal row of symbol 'x' cannot be maintained by any $2RLG$ given in [13]. Two members of L_1 are shown in figure 3.7

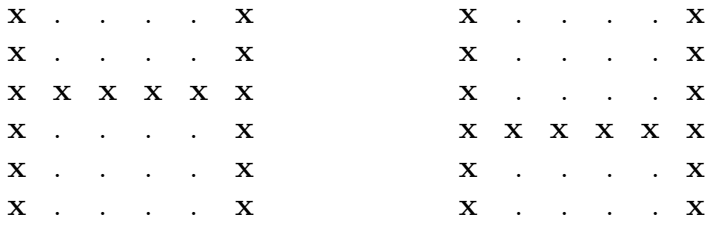


Figure 3.7: Arrays of Token-H

But it is described by the following HAS :

Let $\Sigma = \{x, .\}$, $I = \begin{Bmatrix} x & . & x \\ x & x & x \\ x & . & x \end{Bmatrix}$, $R_c = \{r_1, ..., r_5\}$ and $R_r = \{r_6, ..., r_{15}\}$ given by,

$$r_1 = \begin{array}{|c|}\hline . \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline . \\ \hline x \\ \hline \end{array}$$

$$r_2 = \begin{array}{|c|}\hline . \\ \hline . \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline . \\ \hline . \\ \hline \end{array}$$

$$r_3 = \begin{array}{|c|}\hline x \\ \hline . \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline x \\ \hline . \\ \hline \end{array}$$

$$r_4 = \begin{array}{|c|}\hline \# \\ \hline . \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline x \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline \# \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline . \\ \hline \end{array}$$

$$r_5 = \begin{array}{|c|}\hline . \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline x \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline x \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline . \\ \hline \# \\ \hline \end{array}$$

$$
\begin{array}{rcl}
r_6 &=& \begin{array}{|c|c|}\hline x & . \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline x & x \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline x & . \\ \hline\end{array} \\
r_7 &=& \begin{array}{|c|c|}\hline . & x \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline x & x \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline . & x \\ \hline\end{array} \\
r_8 &=& \begin{array}{|c|c|}\hline . & . \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline x & x \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline . & . \\ \hline\end{array} \\
r_9 &=& \begin{array}{|c|c|}\hline \# & x \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline \# & x \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline \# & x \\ \hline\end{array} \\
r_{10} &=& \begin{array}{|c|c|}\hline x & \# \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline x & \# \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline x & \# \\ \hline\end{array} \\
r_{11} &=& \begin{array}{|c|c|}\hline \# & x \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline \# & x \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline \# & x \\ \hline\end{array} \\
r_{12} &=& \begin{array}{|c|c|}\hline x & . \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline x & x \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline x & . \\ \hline\end{array} \\
r_{13} &=& \begin{array}{|c|c|}\hline . & x \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline x & x \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline . & x \\ \hline\end{array} \\
r_{14} &=& \begin{array}{|c|c|}\hline . & . \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline x & x \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline . & . \\ \hline\end{array} \\
r_{15} &=& \begin{array}{|c|c|}\hline x & \# \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline x & \# \\ \hline\end{array} \;:\; \begin{array}{|c|c|}\hline \# & \# \\ \hline\end{array} \;;\; \begin{array}{|c|c|}\hline x & \# \\ \hline\end{array}
\end{array}
$$

The picture language $L_2 = \{((ab)^p \cup (ba)^q)_n \,/\, p, q, n \geq 1\}$ cannot be described by any HAS. This is due to the fact that the column splicing of any two arrays $((ab)^p)_n$ and $((ba)^q)_n$ will yield an array which is not in L_2. But it is generated by the following $2RLG$:

$$
\begin{array}{rcl}
\Sigma &=& \{a, b\}; \\
\Sigma_I &=& \{A_1, A_2, A_3, A_4\}; \\
V_h &=& \{S, X\}; \\
R_h &=& \left\{ \begin{array}{lll} S \to A_1A_2X, & S \to A_3A_4Y, & X \to A_1A_2X, \\ Y \to A_3A_4Y, & X \to A_1A_2, & Y \to A_3A_4; \end{array} \right. \\
V_v &=& \{A_1, A_2, A_3, A_4\}; \\
R_v &=& \left\{ \begin{array}{lll} A_1 \to aA_1, & A_1 \to a, & A_2 \to bA_2, \\ A_2 \to b, & A_3 \to bA_3, & A_3 \to b, \\ A_4 \to aA_4, & A_4 \to a. & \end{array} \right.
\end{array}
$$

□

Theorem 3.5. *The class $\mathcal{L}(HASL)$ intersects the class of null-context splicing array languages in [8].*

Proof. Let Grid $< X, Y, n, m >$ represent an image G of size $< n, m >$ where n, m are odd positive integers $n, m \geq 3$, and G is given by

$$G[i,j] = \begin{cases} X & \text{if i is odd or j is odd} \\ Y & \text{otherwise} \end{cases}$$

where $1 \leq i \leq n,\ 1 \leq j \leq m$. G is said to be a Grid defined over $< X, Y >$ of size $< n, m >$. GRIDS $< X, Y >$ represent the set of all Grids over $< X, Y >$. A member of GRIDS $< X, . >$ is shown in below Figure 3.8

$$\begin{array}{ccccccc}
\mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\
\mathbf{X} & . & \mathbf{X} & . & \mathbf{X} & . & \mathbf{X} \\
\mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\
\mathbf{X} & . & \mathbf{X} & . & \mathbf{X} & . & \mathbf{X} \\
\mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\
\mathbf{X} & . & \mathbf{X} & . & \mathbf{X} & . & \mathbf{X} \\
\mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X}
\end{array}$$

Figure 3.8: Grid $< X, ., 7, 7 >$

It is known that GRIDS $< X, ., n, m >$ is a null-context splicing array language in [8]. We give an H array splicing system $S = (\Sigma, R_c, R_r, I)$, generating it.

Let $\Sigma = \{X, .\}$, $I = Grid < X, ., 3, 3 > = \left\{ \begin{array}{ccc} X & X & X \\ X & . & X \\ X & X & X \end{array} \right\}$, $R_c = \{r_1, .., r_4\}$ and $R_r = \{r_5, .., r_8\}$ given by,

$$r_1 = \begin{array}{|c|}\hline X \\ \hline X \\ \hline\end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline\end{array} \; : \; \begin{array}{|c|}\hline X \\ \hline X \\ \hline\end{array} \; ; \; \begin{array}{|c|}\hline X \\ \hline . \\ \hline\end{array}$$

$$r_2 = \begin{array}{|c|}\hline X \\ \hline X \\ \hline\end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline\end{array} \; : \; \begin{array}{|c|}\hline X \\ \hline X \\ \hline\end{array} \; ; \; \begin{array}{|c|}\hline . \\ \hline X \\ \hline\end{array}$$

$$r_3 = \begin{array}{|c|}\hline \# \\ \hline X \\ \hline\end{array} \; ; \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline\end{array} \; : \; \begin{array}{|c|}\hline \# \\ \hline X \\ \hline\end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline X \\ \hline\end{array}$$

$$r_4 = \begin{array}{|c|}\hline X \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline X \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline X \\ \hline \# \\ \hline \end{array}$$

$$r_5 = \begin{array}{|c|c|}\hline X & X \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline X & X \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline X & . \\ \hline \end{array}$$

$$r_6 = \begin{array}{|c|c|}\hline X & X \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline X & X \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline . & X \\ \hline \end{array}$$

$$r_7 = \begin{array}{|c|c|}\hline \# & X \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline \# & X \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \# & X \\ \hline \end{array}$$

$$r_8 = \begin{array}{|c|c|}\hline X & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array} \;:\; \begin{array}{|c|c|}\hline X & \# \\ \hline \end{array} \;;\; \begin{array}{|c|c|}\hline X & \# \\ \hline \end{array}$$

□

Definition 3.8. *[4] Let Σ be a finite alphabet and let S_1 and $S_2 \subseteq \Sigma^*$ be two string languages over V. The row-column combination of S_1 and S_2 is a picture language $L = S_1 \oplus S_2 \subseteq \Sigma^{**}$ such that a picture $p \in \Sigma^{**}$ belongs to L if and only if the strings corresponding to the rows and columns of p belong to S_1 and S_2 respectively.*

Theorem 3.6. *The class $\mathcal{L}(HASL)$ intersects the class of picture language of the form $L = S_1 \oplus S_2 \subseteq \Sigma^{**}$ which is the row column combination of S_1 and S_2.*

Proof. We describe a picture language L in HAS which is a row-column combination picture language given by,

$$\Sigma = \{0,1\}, \quad I = \left\{ \begin{array}{ccccccccccccc} 1 & 0 & 1 & & 1 & 0 & 1 & & 1 & 1 & 1 & & 1 & 1 & 1 \\ 1 & 0 & 1 & , & 1 & 1 & 1 & , & 1 & 0 & 1 & , & 1 & 1 & 1 \end{array} \right\},$$

$R_c = \{r_1, r_2, r_3\}$ where $x_i \in \{0,1\}$, for $1 \le i \le 8$ and
$R_r = \{r_4, r_5, r_6\}$ where $y_i \in \{0,1\}$, for $1 \le i \le 4$ are given by,

$$r_1 = \begin{array}{|c|}\hline x_1 \\ \hline x_2 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline x_3 \\ \hline x_4 \\ \hline \end{array}$$

$$r_2 = \begin{array}{|c|}\hline \# \\ \hline x_5 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline 1 \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline \# \\ \hline 1 \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \# \\ \hline x_6 \\ \hline \end{array}$$

$$r_3 = \begin{array}{|c|}\hline x_7 \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline 1 \\ \hline \# \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline 1 \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline x_8 \\ \hline \# \\ \hline \end{array}$$

$$r_1 = \boxed{y_1}\boxed{y_2} \;;\; \boxed{\lambda}\boxed{\lambda} \;:\; \boxed{\lambda}\boxed{\lambda} \;;\; \boxed{y_3}\boxed{y_4}$$
$$r_2 = \boxed{\#}\boxed{1} \;;\; \boxed{\lambda}\boxed{\lambda} \;:\; \boxed{\lambda}\boxed{\lambda} \;;\; \boxed{\#}\boxed{1}$$
$$r_3 = \boxed{1}\boxed{\#} \;;\; \boxed{\lambda}\boxed{\lambda} \;:\; \boxed{\lambda}\boxed{\lambda} \;;\; \boxed{1}\boxed{\#}$$

Here L consists of all pictures over $\Sigma = \{0, 1\}$ whose first and last columns consist only of symbol '1'. In fact $L = (\{1\}S_1\{1\}) \oplus \Sigma^*$ where $S_1 \subseteq \Sigma^*$. □

Remark 3.3. *The concept of controlling the application of rules in a derivation of a word in a grammar is standard in literature [7]. Here, by controlling the application of domino column/row splicing rules, we can obtain square arrays which cannot be described by domino column/row splicing rules.*

Let $\Sigma = \{a\}$, $I = \left\{ \begin{matrix} a & a \\ a & a \end{matrix} \right\}$, $R_c = \{r_1, r_2, r_3\}$ and $R_r = \{r_4, r_5, r_6\}$ given by,

$$r_1 = \begin{matrix}\boxed{a}\\\boxed{a}\end{matrix} \;;\; \begin{matrix}\boxed{\#}\\\boxed{\#}\end{matrix} \;:\; \begin{matrix}\boxed{\#}\\\boxed{\#}\end{matrix} \;;\; \begin{matrix}\boxed{a}\\\boxed{a}\end{matrix}$$
$$r_2 = \begin{matrix}\boxed{\#}\\\boxed{a}\end{matrix} \;;\; \begin{matrix}\boxed{\#}\\\boxed{\#}\end{matrix} \;:\; \begin{matrix}\boxed{\#}\\\boxed{\#}\end{matrix} \;;\; \begin{matrix}\boxed{\#}\\\boxed{a}\end{matrix}$$
$$r_3 = \begin{matrix}\boxed{a}\\\boxed{\#}\end{matrix} \;;\; \begin{matrix}\boxed{\#}\\\boxed{\#}\end{matrix} \;:\; \begin{matrix}\boxed{\#}\\\boxed{\#}\end{matrix} \;;\; \begin{matrix}\boxed{a}\\\boxed{\#}\end{matrix}$$

$$r_4 = \boxed{a}\boxed{a} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{a}\boxed{a}$$
$$r_5 = \boxed{\#}\boxed{a} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{\#}\boxed{a}$$
$$r_6 = \boxed{a}\boxed{\#} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{a}\boxed{\#}$$

Let the control language be $\{R_cR_r\}^n$. Then L the language of squares of order $2^n \times 2^n, n > 0$ over one letter alphabet $\{a\}$ is obtained.

Now we recall the notion of Self cross-over array languages in [6] which was based on the study of Mitrana et. al for string case and we also state its closure properties.

Based on the study of [1] on strings, we now introduce self cross-over array systems based on H-array splicing systems and state them with the results in [6],[9].

In this section we give the examples, counter-examples and thus some comparison results with various other Picture Language classes for Self cross-over array languages $\mathcal{L}(A_{SCO})$ in [6],[9].

3.2.2 Self Cross-Over Array Languages

By the following results it can be seen that various properties of Self cross-over array languages are analogous to the properties of H-array splicing systems. We now prove a comparison of string case H-splicing languages and Self cross-over languages extended to family of H-array splicing languages $\mathcal{L}$(HASL) and the family of self cross-over array languages $\mathcal{L}(A_{SCO})$.

Theorem 3.7. *The family $\mathcal{L}$(HASL) is incomparable with the language family $\mathcal{L}(A_{SCO})$ and is not disjoint.*

Proof. It is known in [5] that the language L cannot be generated by any A_{SCO} for $L = \{(a^p)_n(b^q)_n(a^r)_n(b^s)_n / n \geq 2, p, q, r, s \geq 0\}$ where $(x^m)_n$ is an array consisting of n rows with each row having x^m elements for a given m.

Now, to prove that L can be generated by a H-array splicing system. We give the HAS generating the language : Let $\Sigma = \{a, b\}$,

$$I = \left\{ \begin{array}{cccccc} a & a & b & a & b \\ a & a & b & a & b \end{array} \right., \left. \begin{array}{cccc} a & b & a & b \\ a & b & a & b \end{array} \right., \left. \begin{array}{cccccc} a & b & a & a & b & b \\ a & b & a & a & b & b \end{array} \right\},$$

$R_c = \{r_1, r_2, r_3\}$ and $R_r = \{r_4, r_5, r_6\}$ given by,

$$r_1 = \begin{array}{|c|}\hline \# \\ \hline b \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline a \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline \# \\ \hline b \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline a \\ \hline \end{array}$$

$$r_2 = \begin{array}{|c|}\hline b \\ \hline b \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline a \\ \hline a \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline b \\ \hline b \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline a \\ \hline a \\ \hline \end{array}$$

$$r_3 = \begin{array}{|c|}\hline b \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline a \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline b \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline a \\ \hline \# \\ \hline \end{array}$$

$$r_4 = \boxed{a}\boxed{a} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{a}\boxed{a}$$

$$r_5 = \boxed{a}\boxed{b} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{a}\boxed{b}$$

$$r_6 = \boxed{b}\boxed{a} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{b}\boxed{a}$$

we illustrate the column and row splicing rules given above:

$$\left[\begin{array}{ccc|cc} a & a & b & a & b \\ a & a & b & a & b \end{array}\right], \left[\begin{array}{cc|ccc} a & b & a & a & b \\ a & b & a & a & b \end{array}\right] \;\vdash_{\underline{R_c}}\; \left[\begin{array}{ccccccc} a & a & b & a & a & b & b \\ a & a & b & a & a & b & b \end{array}\right]$$

$$\left[\begin{array}{ccccccc} a & a & b & a & a & b & b \\ \underline{a} & \underline{a} & \underline{b} & \underline{a} & \underline{a} & \underline{b} & \underline{b} \end{array}\right], \left[\begin{array}{ccccccc} \overline{a} & \overline{a} & \overline{b} & \overline{a} & \overline{a} & \overline{b} & \overline{b} \\ a & a & b & a & a & b & b \end{array}\right] \;\vdash_{\underline{R_r}}\; \left[\begin{array}{ccccccc} a & a & b & a & a & b & b \\ a & a & b & a & a & b & b \\ a & a & b & a & a & b & b \\ a & a & b & a & a & b & b \end{array}\right]$$

Thus, we know that iterating the above rules proves $L \in \mathcal{L}(\text{HASL})$.

Now consider language $L' = \{(b)_n(a^{2m})_n(b)_n \;/\; n \geq 2,\ m \geq 0\}$ where n is the number of rows and the power counts the column by denoting the number of that element in that row. We write the rules of A_{SCO} language L'. It can be seen that this language cannot be generated by a H-array splicing system. Now we construct the rules ;

Let $\Sigma = \{a, b\}$, $I = \left\{ \begin{array}{ccc} b & a & b \\ b & a & b \end{array} \right\}$, $R_c = \{r_1\}$ and $R_r = \{r_2, r_3\}$ given by,

$$r_1 = \begin{array}{|c|}\hline a \\ \hline a \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline b \\ \hline b \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline b \\ \hline b \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline a \\ \hline a \\ \hline \end{array}$$

$$r_2 = \boxed{b}\boxed{a} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{b}\boxed{a}$$

$$r_3 = \boxed{a}\boxed{b} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{a}\boxed{b}$$

we can illustrate the column and row splicing rules similar to the above illustration. Thus we prove that the family of H-array splicing system is incomparable with the family of Self cross-over array system. And by various common examples of the classes given and illustrated in section 3.2 we know that the classes are not disjoint. □

We now compare $\mathcal{L}(A_{SCO})$ with two-dimensional classes of languages $\mathcal{L}(LOC), \mathcal{L}(2RLG)$.

Theorem 3.8. *The family $\mathcal{L}(A_{SCO})$ is incomparable with the class of local languages $\mathcal{L}(LOC)$ but is not disjoint with it.*

Proof. The picture language L' consisting of all $n \times m$ arrays for $(n \geq 2, m \geq 2)$ describing token-L of symbol 'x' is in $\mathcal{L}(LOC)$. Now we give an A_{SCO} , $S = (\Sigma, R_c, R_r, I)$ to describe L'.

Let $\Sigma = \{0, x\}$, $I = \left\{ \begin{array}{cc} x & 0 \\ x & x \end{array} \right\}$, $R_c = \{r_1, ..., R_4\}$ and $R_r = \{r_5, ..., r_8\}$ given by,

$$r_1 = \begin{array}{|c|}\hline 0 \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline 0 \\ \hline x \\ \hline \end{array}$$

$$r_2 = \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array}$$

$$r_3 = \begin{array}{|c|}\hline \# \\ \hline 0 \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline \# \\ \hline x \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array}$$

$$r_4 = \begin{array}{|c|}\hline x \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline \# \\ \hline \# \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline x \\ \hline \# \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array}$$

$$r_5 = \begin{array}{|c|c|}\hline x & 0 \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline x & x \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline x & 0 \\ \hline \end{array}$$

$$r_6 = \begin{array}{|c|c|}\hline 0 & 0 \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline x & x \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline 0 & 0 \\ \hline \end{array}$$

$$r_7 = \begin{array}{|c|c|}\hline \# & x \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \# & x \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline \# & x \\ \hline \end{array}$$

$$r_8 = \begin{array}{|c|c|}\hline 0 & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline x & \# \\ \hline \end{array} \; : \; \begin{array}{|c|c|}\hline \# & \# \\ \hline \end{array} \; ; \; \begin{array}{|c|c|}\hline 0 & \# \\ \hline \end{array}$$

The picture language L'' of all images over $\Sigma = \{a\}$ with three columns is not in LOC. In fact, it is not possible to fix the number of columns using only one symbol. i.e. we can be move without restriction on the number of columns the block $\begin{bmatrix} a & a \\ a & a \end{bmatrix}$. But it is obtained by an A_{SCO} where $I = \{a \quad a \quad a\}$, $R_c = \{\emptyset\}$ and $R_r = \{r_1, r_2, r_3\}$ given by,

$$r_1 = \boxed{a}\boxed{a} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{a}\boxed{a}$$
$$r_2 = \boxed{\#}\boxed{a} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{\#}\boxed{a}$$
$$r_3 = \boxed{a}\boxed{\#} \;;\; \boxed{\#}\boxed{\#} \;:\; \boxed{\#}\boxed{\#} \;;\; \boxed{a}\boxed{\#}$$

It is known in [4] that the picture language of square images in which diagonal positions carry symbol '1' but the remaining positions carry symbol '0' is in LOC. But it is not in $\mathcal{L}(A_{SCO})$. Since row and column splicing are independently done, it is clear that arrays with a proposition between rows and columns and in particular pictures with only square size cannot be generated. □

Theorem 3.9. *The family $\mathcal{L}(A_{SCO})$ is incomparable with the language family of two-dimensional right linear grammars $\mathcal{L}(2RLG)$ but is not disjoint with it.*

Proof. The picture language of chessboards with even side-length is generated by a $\mathcal{L}(A_{SCO})$ and is known to be generated by a $2RLG$ given in [4].

The picture language $L_2 = \{((ab)^p \cup (ba)^q)_n \,/\, p, q, n \geq 1\}$ cannot be described by any A_{SCO} . This is due to the fact that the column splicing of any two arrays $((ab)^p)_n$ and $((ba)^q)_n$ will yield an array which is not in L_2. But it is generated by $2RLG$, i.e. we refer to the $2RLG$ example given in [4] to prove the comparison with HAS .

The picture language L_1 consisting of arrays describing token-H cannot be generated by any $2RLG$, as the horizontal row of symbol 'x' cannot be maintained by any $2RLG$. But it is described by the A_{SCO} where $\Sigma = \{x, 0\}$, $I = \left\{ \begin{matrix} x & 0 & x \\ x & x & x \\ x & 0 & x \end{matrix} \right\}$, and R_c is such that the domino rules are written to cut the last column and the first column of the two identical copies of I to paste and make the pattern of more number of columns.

Similarly R_r is such that the cutting and pasting is at the bordered row to enlarge the pattern of I. Thus the incomparability result. □

Remark 3.4. *As a corollary of the theorem above the family of $\mathcal{L}(A_{SCO})$ is incomparable with the families of regular and context-free two-dimensional matrix languages given in [12].*

We recall some closure properties in [6] of the family of $\mathcal{L}(A_{SCO})$ under certain operations given in [12].

Theorem 3.10. *The family $\mathcal{L}(A_{SCO})$ is*

(i) not closed under union and column (row) concatenation.

(ii) closed under reflections on the base and right leg and rotations by $90°, 180°$ and $270°$.

The non-closure result can be seen by constructing picture languages analogous to the linear string case given in [6]. The closure under the geometric operations in (ii) can be shown as in the case of H-array splicing languages given in section 3.2.1.

We now compare some of the picture language classes studied with its examples above for generative power of $\mathcal{L}$(HASL) and $\mathcal{L}(A_{SCO})$ also with $\mathcal{L}$(SASL) .

3.2.3 Simple array splicing langauges

Theorem 3.11. *The class of $\mathcal{L}(LOC)$ of local array languages and $\mathcal{L}$(SASL) of Simple array splicing languages are incomparable but not disjoint in [6].*

Proof. The picture language M consisting of all $n \times m$ arrays ($n \geq 2, m \geq 2$) describing token-L of symbol '1' is in LOC given in [6]. It is also described by a Simple array splicing system $S = (\Sigma, R_c, R_r, I)$ as in example 3.4.

It is known in theorem 3.3 that the picture language L of all rectangular arrays over $\Sigma = \{a\}$ with three columns is not in LOC. But it is generated by a Simple array splicing system where

$$I = \left\{ \begin{array}{ccc} a & a & a \\ a & a & a \\ a & a & a \end{array} \right\}, R_c = \emptyset \text{ and } R_r = \{r_1, r_2, r_3\}$$

$$r_1 = \boxed{a}\boxed{a} \;;\; \boxed{\lambda}\boxed{\lambda} \;:\; \boxed{a}\boxed{a} \;;\; \boxed{\lambda}\boxed{\lambda}$$
$$r_2 = \boxed{\#}\boxed{a} \;;\; \boxed{\lambda}\boxed{\lambda} \;:\; \boxed{\#}\boxed{a} \;;\; \boxed{\lambda}\boxed{\lambda}$$
$$r_3 = \boxed{a}\boxed{\#} \;;\; \boxed{\lambda}\boxed{\lambda} \;:\; \boxed{a}\boxed{\#} \;;\; \boxed{\lambda}\boxed{\lambda}$$

It is known [4] that the picture language of square images in which diagonal positions carry symbol '1' but the remaining positions carry symbol '0' is in $\mathcal{L}(LOC)$. But it is not in $\mathcal{L}$(SASL) . Since row and column splicing are independently done, it is clear that arrays with a proportion between rows and columns and in particular pictures with only square size cannot be generated by any $\mathcal{L}$(SASL) . □

Theorem 3.12. *The class $\mathcal{L}$(SASL) of Simple array splicing languages and $\mathcal{L}(2RLG)$ in [9] of picture languages generated by two dimensional right linear grammars are incomparable but not disjoint.*

Proof. The picture language of "chessboards" with even side-length given in Figure 3.1 is also generated by a SAS given in example and is known to be generated by a $2RLG$.

The picture language L_1 consisting of arrays describing token-H given in figure 3.9 cannot be generated by any $2RLG$, as the horizontal row of symbol 'x' cannot be maintained by any $2RLG$. But the language consisting of picture arrays describing token-H with three rows and any number of columns can be generated by the following SAS :

$$\text{Let } \Sigma = \{x, .\},\ I = \left\{ \begin{array}{cccc} x & . & . & x \\ x & x & x & x \\ x & . & . & x \end{array} \right\},\ R_c = \{r_1, ..., r_4\} \text{ and } R_r = \emptyset$$

$$r_1 = \begin{array}{|c|}\hline . \\ \hline x \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline . \\ \hline x \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$
$$r_2 = \begin{array}{|c|}\hline x \\ \hline . \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline x \\ \hline . \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$
$$r_3 = \begin{array}{|c|}\hline \# \\ \hline . \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline \# \\ \hline . \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$
$$r_4 = \begin{array}{|c|}\hline . \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \;:\; \begin{array}{|c|}\hline . \\ \hline \# \\ \hline \end{array} \;;\; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}$$

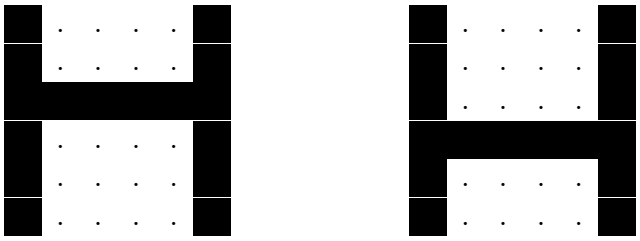
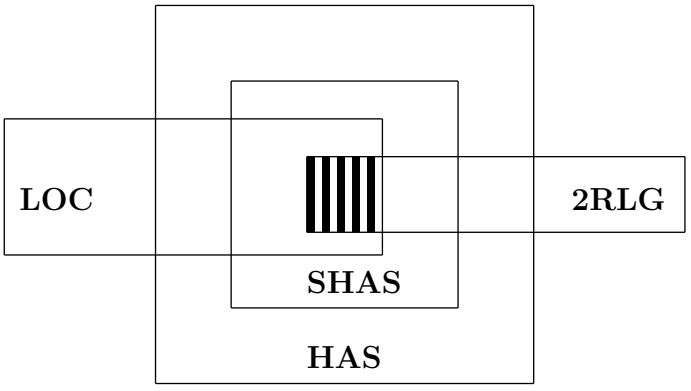


Figure 3.9: Token-H

Note that the above defined language by SAS is also given by HAS in theorem comparing 2DRLG. Also, we already know that the picture language $L_2 = \{((ab)^p \cup (ba)^q)_m \,/\, p, q, m \geq 1\}$ cannot be described by any HAS . This is due to the fact that the column splicing of any two arrays $((ab)^p)_m$ and $((ba)^q)_m$ will yield an array which is not in L_2. Similarly, for the same property of SAS to that of HAS there can be no SAS generating L_2. But as stated before it is generated by the $2RLG$. □

Remark 3.5. *We have the following relationship among* $\mathcal{L}$*(SASL) ,* $\mathcal{L}(LOC), \mathcal{L}(2RLG)$ *as seen from the above theorems*



Note that the picture language of "chessboards" with even side-length is generated by a SAS in example 3.1 and is known to be generated by a $2RLG$ in the example in 3.4. It is also a local language.Thus it is a picture language in all the three classes.

In fact the set Θ of 2×2 windows describing this language is

$$\Theta = \left\{ \begin{array}{cccccccccccc} \# & \# & \# & \# & \# & b & a & \# & \# & \# & \# & \# \\ \# & a, & b & \#, & \# & \#, & \# & \#, & a & b, & b & a \\ \\ \# & b & a & \# & a & b & b & a & a & b & b & a \\ \# & a, & b & \#, & \# & \#, & \# & \#, & b & a, & a & b \\ \\ 1 & 0 & 0 & 0 & \# & a & b & \# & & & & \\ 1 & 1, & 1 & 1, & \# & b, & a & \# & & & & \end{array} \right\}$$

3.3 Simple Tree Splicing Systems

We now introduce the notion of Simple splicing on trees as an application of the simple splicing rules introduced in [14]. The splicing of trees considered here is a special form of a general notion of splicing of trees considered in [11].

We consider labelled rooted trees T which are connected cycle-free graphs with a designated node r called the root of the tree and a label $l(v)$ for every node v. Since there is a unique simple path from the root r to any other node v in T, this determines a direction to the edges of the tree and thus tree is viewed as a directed graph with a precedence relationship such that every node has zero or more descendant nodes. A node with a zero descendant is called a leaf and any other node is called an interior node. A subtree of a given tree T is also a labelled rooted tree with its root as an interior vertex of tree.

It is usual to denote a tree T with root label a and subtrees $T_1, ..., T_n$ with root labels $b_1, ..., b_n$ in linear form as given below $T = a(T_1, ..., T_n)$. We call a labelled rooted tree T with labels in a set V, simply as a tree T over V. The yield of a tree T is a string obtained by reading the labels of the leaves from left to right.

Example 3.5. *Let $V = (a, b, c)$. let $T = c(a, c(a, c(a, c, b), b), b)$ where T is a tree. The yield of tree T is a^3cb^3.*

We now introduce the notion of simple splicing of two trees, by considering simple splicing rules used in the splicing of words [10].

Definition 3.9. *Let V be an alphabet. Let r be a simple splicing rule $r = (c\ ;\ \lambda\ :\ c\ ;\ \lambda)$ where $c \in V$. Let T_1 and T_2 be two trees such that c is a root label of a subtree $T_1^{'}$ of T_1 and a subtree $T_2^{'}$ of T_2. We say that a tree T_3 is obtained by simple tree splicing of T_1 with T_2, if T_3 is the tree obtained from T_1 removing the subtree $T_1^{'}$ with root v having label c and attaching $T_2^{'}$ at the node v.*

Definition 3.10. *A simple tree splicing system $S = (V, A, R)$ where V is an alphabet, A is a finite set of trees over V and R is a finite set of simple tree splicing rules. The tree language $T(S)$ consists of all trees obtained from the trees in A by repeatedly applying the tree splicing rules of R. The string language associated with the system S is the set $L(S)$ of words which are the yields of the trees in $T(S)$.*

Example 3.6. *Let $V = (a, b, c)$. let T be a tree $T = c(a, c, b)$. Let $S = (V, T, r)$ where $r = (c; \lambda : c; \lambda)$. On splicing T with itself we obtain a tree $T^{'} = c(a, c(a, c, b), b)$. Repeatedly using the splicing operation, we obtain a set of trees $c(a, c, b), c(a, c(a, c, b), b), c(a, c(a, c(a, c, b)b)b), ...$ which constitutes the tree language $T(S)$. The string language of S is $L(S) = a^n c b^n : n \geq 1$.*

We exhibit the relation between the set of derivation trees of a context free grammar and the tree language of a simple tree splicing system.

Theorem 3.13. *Given a context free language $G = (V_N, V_T, P, S)$, there exists a simple tree splicing system S such that the tree language $T(S)$ is exactly the set of derivation trees of G. As a consequence the context free language $L(G)$ generated by G is simply the string language of $L(S)$*

Proof. Assume that the given CFG is in Chomsky Normal Form with rules of the form $A \to BC$ or $A \to a$ where A, B, C are non-terminals and a is a terminal. For each rule $A \to BC$ we associate a tree $A(B, C)$. For each rule $A \to a$ we associate a tree $A(a)$. A corresponding simple tree splicing system S is constructed as follows : $S = (V_N \cup V_T, A^{'}, R)$ where $A^{'}$ consists of trees associated with the rules of P. R consists of simple splicing rules of the form $X; \lambda : X; \lambda$ whenever X is the left hand side of a rule in G and is a symbol in the right hand side of a rule in G. It is straight forward to see that $T(S)$ consists of exactly the derivation trees of G. □

Remark 3.6. *It is known that the string language of a simple H system [10] is regular. It is of interest to note from the theorem 3.1 that the string language of a simple tree splicing system is context free and thus the generative power of the simple splicing rules is increased when they are applied on the tree structures.*

Now in addition to the above study on the Splicing operation and its variants or restriction over Picture languages, we define a new formalism of this kind of Splicing defined with Grammar Systems. Grammar System being a well studied notion of formal languages for problem solving, we have tried to study set of pictures as components of such systems with operation of splicing (HAS) defined.

3.4 Splicing Array Grammar Systems

We now introduce the notion of Splicing grammar systems in which the component grammars are 2DMatrix Grammars, named Image Splicing Grammar Systems.

Definition 3.11. *An Image Splicing Grammar System is a construct* $\Gamma = (V_h, \Sigma_I, V_v, T, (S_1, R_1^h, R_1^v), ..., (S_n, R_n^h, R_n^v), M)$ *where,*

V_h *is a finite set of variables called horizontal variables;*
V_v *is a finite set of variables called vertical variables;*
$\Sigma_I \subseteq V_v$ *is a finite set of intermediates;*
T *is a finite set of terminals;*
$S_i, 1 \leq i \leq n$ *is the start symbol of the corresponding horizontal component;*
$R_i^h, 1 \leq i \leq n$ *is a finite set of rules called horizontal productions and the rules can be regular or context free or context sensitive;*
$R_i^v, 1 \leq i \leq n$ *is a finite set of right linear rules called vertical productions;*
$M = \{R_c, R_r\}$ *is a finite set of domino column or row splicing rules of the form*

$$m = \alpha_1 \ ; \ \alpha_2 \ : \ \alpha_3 \ ; \ \alpha_4 \quad or \quad \beta_1 \ ; \ \beta_2 \ : \ \beta_3 \ ; \ \beta_4$$

where $\alpha_i = \begin{array}{|c|}\hline a \\ \hline b \\ \hline \end{array}$ *and* $\beta_i = \begin{array}{|c|c|}\hline c & d \\ \hline \end{array}$ *for some* $a, b, c, d \in V_v \cup \{T\} \cup \{\lambda\}$.

The derivations take place in two phases as follows :

Each component grammar generates a word called intermediate word, over intermediates starting from its own start symbol and using its horizontal production rules ; the derivations in this phase are done with the component grammars working in parallel.

In the second phase any of the following steps can take place :

(i) each component grammar can rewrite as in a two dimensional matrix grammar using the vertical rules, starting from its own intermediate word generated in the first phase. (The component grammars rewrite in parallel).Note that the component grammars together terminate or together continue rewriting in the vertical direction.

(ii) At any instant the array X generated in the i^{th} component for some $1 \leq i \leq n$ and the array Y generated in the j^{th} component for some $1 \leq j \leq n$ can be spliced using column / row domino splicing rules as in definition 3.1, thus yielding array Z in i^{th} component and W in the j^{th} component; In fact Z will have a prefix of X column concatenated with a suffix of Y and W will have a prefix of Y, column concatenated with a suffix of X, the prefixes and suffixes being given by the splicing rules. In any other components (other than i^{th}, j^{th} components), the arrays generated at this instant will remain unchanged during this splicing process.

There is no priority between steps (i) and (ii).

The language $L_i(\Gamma)$ generated by the ith component of Γ consists of all arrays, generated over T, by the derivations described above.

This language will be called the *individual language* of the system and we may choose this to be the language of the first component and $L_t(\Gamma) = \bigcup_{i=1}^{n} L_i(\Gamma)$ as the *total language*. The family of individual languages generated by ISGS with n components of type X for $X \in \{REG, CF\}$ is denoted by $I_{isgs}L_n(X)$, and the corresponding family of total languages by $T_{isgs}L_n(X)$ respectively and $Y_{isgs}L_n(X)$ when $Y \in \{I, T\}$. We basically deal with individual languages although the results obtained apply to total languages as well.

Example 3.7. *Let* $\Gamma = (\{S, X\}, \{A, B, C\}, \{A, B, C, D\}\{., x\},$
$(S, R^h, R^v), (S, R^h, R^v), (S, R^h, R^v), M)$ *where*

$$
\begin{aligned}
R^h &= \{S \to AX,\ X \to BX,\ X \to C\ \} \\
R^v &= \{A \to xA,\ A \to x,\ B \to xD, D \to .D, \\
&\quad\ D \to x,\ C \to xC,\ C \to x\}, \\
M &= \{R_c, R_r\}\ \textit{given below,}
\end{aligned}
$$

$$
\begin{array}{|c|}\hline x \\ \hline . \\ \hline \end{array}\ ;\ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}\ :\ \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}\ ;\ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}
$$

$$
\begin{array}{|c|}\hline . \\ \hline . \\ \hline \end{array}\ ;\ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}\ :\ \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}\ ;\ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}
$$

$$
\begin{array}{|c|}\hline . \\ \hline D \\ \hline \end{array}\ ;\ \begin{array}{|c|}\hline x \\ \hline C \\ \hline \end{array}\ :\ \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array}\ ;\ \begin{array}{|c|}\hline x \\ \hline A \\ \hline \end{array}
$$

$$
\begin{array}{|c|c|}\hline x & . \\ \hline \end{array}\ ;\ \begin{array}{|c|c|}\hline A & D \\ \hline \end{array}\ :\ \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}\ ;\ \begin{array}{|c|c|}\hline x & x \\ \hline \end{array}
$$

$$
\begin{array}{|c|c|}\hline . & . \\ \hline \end{array}\ ;\ \begin{array}{|c|c|}\hline D & D \\ \hline \end{array}\ :\ \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}\ ;\ \begin{array}{|c|c|}\hline x & x \\ \hline \end{array}
$$

$$
\begin{array}{|c|c|}\hline . & x \\ \hline \end{array}\ ;\ \begin{array}{|c|c|}\hline D & D \\ \hline \end{array}\ :\ \begin{array}{|c|c|}\hline \lambda & \lambda \\ \hline \end{array}\ ;\ \begin{array}{|c|c|}\hline x & x \\ \hline \end{array}
$$

The horizontal rules in a component generate intermediate words of the form AB^nC with the same value of $n \geq 1$ at a time. The vertical rules of the components generate from an intermediate word rectangle pictures of (.)'s surrounded or bordered by x's except the bottom border which will be of the form AD^nC. At this stage with domino splicing rules, column or row splicing of the array in a component with the array in another component can take place before rewriting is terminated in the components with terminating vertical rules. In fact any picture generated in the individual language of this Image splicing grammar system will be either (i) rectangular pictures in which any row, except the first and the last, will be of the form $(x(.)^n)^k x$ for some $k\epsilon\{1, 2, 3\}$ or (ii)rectangular pictures in which any column, except the first and the last, will have a similar feature or (iii) simply a column of $x's$. Two such pictures obtained are shown in Figures 3.10 and 3.11.

Example 3.8. *Let*

$$
\begin{aligned}
\Gamma &= (\{S, X\}, \{A, B, E\}, \{A, B, C, D, E\} \\
&\qquad \{., x\}, (S, R^h, R^v), (S, R^h, R^v), (S, R^h, R^v), M)
\end{aligned}
$$

```
x x x x x x x x x x x x x
x . . . x . . . x . . . x
x . . . x . . . x . . . x
x . . . x . . . x . . . x
x x x x x x x x x x x x x
```

Figure 3.10: Horizontal boxes with 'x' frame

```
x x x x x
x . . . x
x . . . x
x x x x x
x . . . x
x . . . x
x x x x x
x . . . x
x . . . x
x x x x x
```

Figure 3.11: Vertical boxes with 'x' frame

where

$$
\begin{array}{lcl}
R^h & = & \{S \to EXE, \quad X \to AXB, \quad X \to AB\} \\
R^v & = & \{A \to aC, \quad C \to .C, \quad C \to a, \quad B \to bD, \\
 & & \quad D \to .D, \quad D \to b, \quad E \to xE, \quad E \to x \quad \} \text{ and} \\
M & = & \{R_c\} \text{ given by,}
\end{array}
$$

$$
\begin{array}{ccccccc}
\begin{array}{|c|}\hline b \\ \hline . \\ \hline \end{array} & ; & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & : & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & ; & \begin{array}{|c|}\hline a \\ \hline . \\ \hline \end{array} \\
\begin{array}{|c|}\hline . \\ \hline . \\ \hline \end{array} & ; & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & : & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & ; & \begin{array}{|c|}\hline . \\ \hline . \\ \hline \end{array} \\
\begin{array}{|c|}\hline . \\ \hline D \\ \hline \end{array} & ; & \begin{array}{|c|}\hline x \\ \hline E \\ \hline \end{array} & : & \begin{array}{|c|}\hline x \\ \hline E \\ \hline \end{array} & ; & \begin{array}{|c|}\hline . \\ \hline C \\ \hline \end{array}
\end{array}
$$

The horizontal rules generate in a component intermediate words of the form EA^nB^nE with the same value of $n \geq 1$ at a time. The vertical rules of the components generate from an intermediate word rectangle

pictures of (.)'s bordered on the top by words of the form $xa^m b^m x$, on the bottom by words of the form $EC^m D^m E$, the leftmost column being a column of x's ending with E and the rightmost column being a column of x's ending with E. At this stage with domino splicing rules, column splicing of the array in a component with the array in another component can take place before rewriting is terminated in the components with terminating vertical rules. One such picture obtained is shown in Figure 3.12.

x	.	.	.	.	.	.	.	.	.	.	.	.	x
x	.	.	.	.	.	.	.	.	.	.	.	.	x
x	.	.	.	.	.	.	.	.	.	.	.	.	x
x	a	a	b	b	a	a	b	b	a	a	b	b	x
x	.	.	.	.	.	.	.	.	.	.	.	.	x
x	.	.	.	.	.	.	.	.	.	.	.	.	x
x	.	.	.	.	.	.	.	.	.	.	.	.	x

Figure 3.12: $Xa^2b^2a^2b^2a^2b^2X$

3.4.1 Comparisons

Theorem 3.14. *For* $Y \in \{I, T\}$,

(i) $2DRML = Y_{isgs}L_1(REG)$

(ii) $2DRML \subset Y_{isgs}L_2(REG)$

(iii) $2DCFML = Y_{isgs}L_1(CF)$

(iv) $2DCFML \subset Y_{isgs}L_2(CF)$

Statements (i) and (iii) are obvious. The proper inclusion in statement (ii) is a consequence of Example 3.7. In fact the pictures in Figures 3.10 and 3.11 cannot be generated by any $2DRMG$ as both the rules in both the horizontal and vertical phases are only regular rules. Likewise the proper inclusion in statement (iv) is a consequence of Example 3.8 since the rules in the horizontal phase of a $2DCFMG$ are only CF rules and so the pictures as in Figure 3.12 require CS rules in the first phase.

Example 3.9. *Let*

$$\begin{aligned}\Gamma &= (\{S_1, \cdots S_n, X\}, \{A_1, \cdots, A_n, B_1, \cdots, B_n, C_1, \ldots, C_n, \}, \\ &\quad \{A_1, \cdots, A_n, B_1, \cdots, B_n, C_1, \ldots, C_n, D_1, \ldots, D_n\}, \{., x, a, b\}, \\ &\quad (S, R^{h_1}, R^{v_1}), (S, R^{h_2}, R^{v_2}), \cdots, (S, R^{h_n}, R^{v_n}), M)\end{aligned}$$

where

$$\begin{aligned}R^{h_1} &= \{S_1 \to A_1X, X \to B_1X, X \to C_1 \\ R^{v_1} &= \{A_1 \to xA_1, A_1 \to x, B_1 \to aD_1, D_1 \to .D_1, D_1 \to a, \\ &\quad C_1 \to xC_1, C_1 \to x, D_i \to a \ if \ i \geq 2 \ and \ i \ odd, \\ &\quad D_i \to b \ if \ i \geq 2 \ and \ i \ even, C_i \to x\}\end{aligned}$$

For $i > 1$ *and* i *even*

$$\begin{aligned}R^{h_i} &= \{S_i \to A_iX, X \to B_iX, X \to C_i \\ R^{v_i} &= \{A_i \to xA_i, A_i \to x, B_i \to bD_i, D_i \to .D_i, C_i \to xC_i, \}.\end{aligned}$$

For $i > 1$ *and* i *odd*

$$\begin{aligned}R^{h_i} &= \{S_i \to A_iX, X \to B_iX, X \to C_i \\ R^{v_i} &= \{A_i \to xA_i, A_i \to x, B_i \to aD_i, D_i \to .D_i, C_i \to xC_i, \}.\end{aligned}$$

$M = \{R_c\}$ *given below,*

$$\begin{array}{|c|}\hline a \\ \hline . \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \ : \ \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline b \\ \hline . \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \ : \ \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline . \\ \hline . \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \ : \ \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline . \\ \hline D_i \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline C_i \\ \hline \end{array} \ : \ \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline A_i \\ \hline \end{array}$$

We note that the top and bottom rows of the rectangular arrays generated in the individual language will be of the form $xa^mxb^mxa^mxb^m...x$ as there are n component grammars.

Theorem 3.15. *For* $Y \in \{I, T\}$,

(i) $2DRML = Y_{isgs}L_1(REG) \subset Y_{isgs}L_2(REG) \subset \cdots \subset Y_{isgs}L_n(REG) \subset \cdots$

(ii) $2DCFML = Y_{isgs}L_1(CF) \subset Y_{isgs}L_2(CF) \subset \cdots \subset Y_{isgs}L_n(CF) \subset \cdots$

The first statement follows in view of the Example 3.15. The second can be shown on similar lines.

We now give the various results studied in [2] which were studied in the same line as section 3.4 where we incorporate the HAS on 2D Matrix Grammars along with grammar system, in parallel.

Now introduce the notion of Splicing array grammar system in which the component grammars consist of rules of 2d tabled matrix grammars. And we give some comparisons explaining the construction of the formalism. Even though its a simple variant of the above defined Image Splicing Grammar System, we again state the definition below.

Definition 3.12. *A Splicing Array Grammar system (SAGS) is a construct*
$\Gamma = (V_h, \Sigma_I, V_v, T, (S_1, R_1^h, R_1^v), ..., (S_n, R_n^h, R_n^v), M)$ *where,*
V_h *is a finite set of variables called horizontal variables;*
V_v *is a finite set of variables called vertical variables;*
$\Sigma_I \subseteq V_v$ *is a finite set of intermediates;*
T *is a finite set of terminals;*
$S_i, 1 \leq i \leq n$ *is the start symbol of the corresponding horizontal component;*
$R_i^h, 1 \leq i \leq n$ *is a finite set of rules called horizontal productions and the rules can be regular or context free or context sensitive;*
$R_i^v, 1 \leq i \leq n$ *is a finite set tables of right linear rules called vertical productions;The productions in a table are all either of the form* $A \to aB$ *or of the form* $A \to a$*;*
$M = \{R_c, R_r\}$ *is a finite set of domino column or row splicing rules of the form*

$m = \alpha_1 \ ; \ \alpha_2 \ : \ \alpha_3 \ ; \ \alpha_4 \quad or \quad \beta_1 \ ; \ \beta_2 \ : \ \beta_3 \ ; \ \beta_4$

where $\alpha_i = \boxed{\begin{array}{|c|}\hline a \\ \hline b \\ \hline \end{array}}$ *and* $\beta_i = \boxed{\begin{array}{|c|c|}\hline c & d \\ \hline \end{array}}$ *for some* $a, b, c, d \in V_v \cup \{T\} \cup \{\lambda\}$.

The derivations take place in two phases as follows :

Each component grammar generates a word called intermediate word, over intermediates starting from its own start symbol and using its horizontal production rules ; the derivations in this phase are done with the component grammars working in parallel.

In the second phase any of the following steps can take place :

(i) each component grammar can rewrite as in a two dimensional matrix grammar using the tables of vertical rules, starting from its own intermediate word generated in the first phase. (The component grammars rewrite in parallel and the rules of a table are applied together). Note that the component grammars together terminate or together continue rewriting in the vertical direction.

(ii) at any instant the array X generated in the i^{th} component for some $1 \leq i \leq n$ and the array Y generated in the j^{th} component for some $1 \leq j \leq n$ can be spliced using column / row domino splicing rules as in definition 3.1, thus yielding array Z in i^{th} component and W in the j^{th} component; In fact Z will have a prefix of X column concatenated with a suffix of Y and W will have a prefix of Y, column concatenated with a suffix of X, the prefixes and suffixes being given by the splicing rules. In any other components (other than i^{th}, j^{th} components), the arrays generated at this instant will remain unchanged during this splicing process.

There is no priority between steps (i) and (ii).

The language $L_i(\Gamma)$ generated by the *ith* component of Γ consists of all arrays, generated over T, by the derivations described above.

This language will be called the *individual language* of the system and we may choose this to be the language of the first component and $L_t(\Gamma) = \bigcup_{i=1}^{n} L_i(\Gamma)$ as the *total language*. The family of individual languages generated by SAGS with n components of type X for $X \in \{REG, CF\}$ is

denoted by $I_{sags}L_n(X)$, and the corresponding family of total languages by $T_{sags}L_n(X)$ respectively and $Y_{sags}L_n(X)$ when $Y \in \{I, T\}$. We basically deal with individual languages although the results obtained apply to total languages as well.

Remark 3.7. *The image splicing grammar system (ISGS) introduced in [2] in which the component grammars are 2D Matrix grammars [13] is a special case of SAGS.*

Example 3.10. *Let*

$$\begin{aligned}\Gamma &= (\{S, X, Y, Z\}, \{A, B, E, C\}, \{A, B, C, D, E, F, T, U\}\{., x\}, \\ &\quad (S, R^h, R^v), (S, R^h, R^v), (S, R^h, R^v), M)\end{aligned}$$

where,

$$\begin{aligned}R^h &= \{S \to AX, X \to BX, X \to BY, Y \to EZ, Z \to C\} \\ R^v &= \{t_1, t_2, ..., t_6\} \\ t_1 &= \{A \to xA, B \to .B, E \to .E, C \to xC\}. \\ t_2 &= \{A \to xA, B \to .D, E \to .F, C \to xC\}. \\ t_3 &= \{A \to xA, D \to .D, F \to .F, C \to xC\}. \\ t_4 &= \{A \to xA, D \to xT, F \to yU, C \to xC\} \\ t_5 &= \{A \to xA, T \to .T, U \to .U, C \to xC\} \\ t_6 &= \{A \to x, T \to ., U \to ., C \to x\}\end{aligned}$$

$M = \{R_c\}$ *given by,*

$$\begin{array}{ccccccc}
\begin{array}{|c|}\hline . \\ \hline . \\ \hline \end{array} & ; & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & : & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & ; & \begin{array}{|c|}\hline . \\ \hline . \\ \hline \end{array} \\
\begin{array}{|c|}\hline . \\ \hline y \\ \hline \end{array} & ; & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & : & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & ; & \begin{array}{|c|}\hline . \\ \hline x \\ \hline \end{array} \\
\begin{array}{|c|}\hline y \\ \hline . \\ \hline \end{array} & ; & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & : & \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} & ; & \begin{array}{|c|}\hline x \\ \hline . \\ \hline \end{array} \\
\begin{array}{|c|}\hline . \\ \hline U \\ \hline \end{array} & ; & \begin{array}{|c|}\hline x \\ \hline C \\ \hline \end{array} & : & \begin{array}{|c|}\hline x \\ \hline A \\ \hline \end{array} & ; & \begin{array}{|c|}\hline . \\ \hline T \\ \hline \end{array}
\end{array}$$

The horizontal rules in a component generate intermediate words of the form AB^nEC with the same value of $n \geq 1$ at a time. The vertical rules of the components generate from an intermediate word rectangular pictures of digitized token H surrounded in the left and right by x's and the bottom border of the form AT^nUC. At this stage with domino splicing rules, column splicing of the array in a component with the array in another component can take place before rewriting is terminated in the components with terminating vertical rules. In fact any picture generated in the individual language of this splicing array grammar system will be rectangular pictures in which any row, except a middle row, will be of the form $x(.)^{kn}x$ and a middle row will be of the form $x((x)^ny)^kx$ for some $k\epsilon\{1,2,3\}$. One such picture obtained is shown in Figure 3.13.

x	.	.	.	.	.	.	.	.	.	.	.	x
x	.	.	.	.	.	.	.	.	.	.	.	x
x	x	x	x	y	x	x	x	y	x	x	y	x
x	.	.	.	.	.	.	.	.	.	.	.	x
x	.	.	.	.	.	.	.	.	.	.	.	x
x	.	.	.	.	.	.	.	.	.	.	.	x

Figure 3.13: $Xx_n^3yx_n^3yx_n^3yx_n^3X$

Example 3.11. *Let*

$$\begin{aligned}\Gamma &= (\{S,X\},\{A,B,E\},\{A,B,C,D,E\} \\ &\quad \{.,x\},(S,R^h,R^v),(S,R^h,R^v),(S,R^h,R^v),M)\end{aligned}$$

where,

$$\begin{aligned}R^h &= \{S \to EXE, X \to AXB, X \to AB\} \\ R^v &= \{t_1,t_2,t_3,t_4,t_5\} \\ t_1 &= \{A \to .A, B \to .B, E \to xE\} \\ t_2 &= \{A \to .C, B \to .D, E \to xE,\} \\ t_3 &= \{C \to aY, D \to bZ, E \to xE,\} \\ t_4 &= \{Y \to .Y, Z \to .Z, E \to xE,\} \\ t_5 &= \{Y \to ., Z \to ., E \to x\} \text{ and} \\ M &= \{R_c\}, \text{ given by,}\end{aligned}$$

$$\begin{array}{c|c} b \\ \hline . \end{array} \; ; \; \begin{array}{c|c} x \\ \hline x \end{array} \; : \; \begin{array}{c|c} x \\ \hline x \end{array} \; ; \; \begin{array}{c|c} a \\ \hline . \end{array}$$

$$\begin{array}{c|c} . \\ \hline b \end{array} \; ; \; \begin{array}{c|c} x \\ \hline x \end{array} \; : \; \begin{array}{c|c} x \\ \hline x \end{array} \; ; \; \begin{array}{c|c} . \\ \hline a \end{array}$$

$$\begin{array}{c|c} . \\ \hline . \end{array} \; ; \; \begin{array}{c|c} x \\ \hline x \end{array} \; : \; \begin{array}{c|c} x \\ \hline x \end{array} \; ; \; \begin{array}{c|c} . \\ \hline . \end{array}$$

$$\begin{array}{c|c} . \\ \hline Z \end{array} \; ; \; \begin{array}{c|c} x \\ \hline E \end{array} \; : \; \begin{array}{c|c} x \\ \hline E \end{array} \; ; \; \begin{array}{c|c} . \\ \hline Y \end{array}$$

The horizontal rules generate in a component intermediate words of the form EA^nB^nE with the same value of $n \geq 1$ at a time. The vertical rules of the components generate from an intermediate word rectangle pictures of (.)'s with a middle row of the form xa^mb^mx, and the bottom row of the form EC^mD^mE , the leftmost column being a column of x's ending with E and the rightmost column being a column of x's ending with E. At this stage with domino splicing rules, column splicing of the array in a component with the array in another component can take place before rewriting is terminated in the components with terminating vertical rules. One such picture obtained is already shown for ISG, it can be referred in Figure 3.12.

Theorem 3.16. *For* $Y \in \{I, T\}$,

(i) $2dRML = Y_{isgs}L_1(REG) \subset 2dTRML = Y_{sags}L_1(REG)$

(ii) $2dRML \subset Y_{isgs}L_2(REG)$

(iii) $2dCFML = Y_{isgs}L_1(CF) \subset 2dTCFML = Y_{sags}L_1(CF)$

(iv) $2dTRML \subset Y_{sags}L_2(REG)$

(v) $2dTCFML \subset Y_{sags}L_2(CF)$

(vi) $Y_{sags}L_3(REG) - CS \neq \phi$.

Proof. The equalities in statements (i) and (iii) are clear from definitions and the proper inclusions are known in [13]. Statement (ii) is proved in [2]. Inclusions in statement (iv) and (v) are clear. The proper inclusion in statement (iv) is a consequence of Example 3.10. In fact the picture language of the Example 3.10 even with $k = 2$, cannot be generated by any $2dTRMG$ as the rules in both the horizontal and vertical phases are

only regular rules. Likewise the proper inclusion in statement (v) is a consequence of Example 3.11 even with two components since the rules in the horizontal phase of a $2dCFMG$ are only CF rules but the pictures generated will require CS rules.The last statement follows from example 3.10 as the the pictures in Figure 3.13 require CS rules in the first phase to generate these.

□

Example 3.12. *Let*

$$\begin{aligned}\Gamma &= (\{S_1, \cdots S_n, X\}, \{A_1, \cdots, A_n, B_1, \cdots, B_n, C_1, \ldots, C_n, \}, \\ &\quad \{A_1, \cdots, A_n, B_1, \cdots, B_n, C_1, \ldots, C_n, D_1, \ldots, D_n\}, \\ &\quad \{., x, a, b\}, (S, R^{h_1}, R^{v_1}), (S, R^{h_2}, R^{v_2}), \cdots, (S, R^{h_n}, R^{v_n}), M)\end{aligned}$$

where,

$$\begin{aligned}R^{h_1} &= \{S_1 \to A_1X, X \to B_1X, X \to C_1\} \\ R^{v_1} &= \{A_1 \to xA_1, A_1 \to x, B_1 \to aD_1, D_1 \to .D_1, D_1 \to a, \\ &\quad C_1 \to xC_1, C_1 \to x, D_i \to a \ if \ i \geq 2 \ and \ i \ odd, \\ &\quad D_i \to b \ if \ i \geq 2 \ and \ i \ even, C_i \to x\}.\end{aligned}$$

For $i > 1$ and i even

$$\begin{aligned}R^{h_i} &= \{S_i \to A_iX, X \to B_iX, X \to C_i\} \\ R^{v_i} &= \{A_i \to xA_i, A_i \to x, B_i \to bD_i, D_i \to .D_i, C_i \to xC_i, \}.\end{aligned}$$

For $i > 1$ and i odd

$$\begin{aligned}R^{h_i} &= \{S_i \to A_iX, X \to B_iX, X \to C_i\} \\ R^{v_i} &= \{A_i \to xA_i, A_i \to x, B_i \to aD_i, D_i \to .D_i, C_i \to xC_i, \}.\end{aligned}$$

$M = \{ R_c\}$ given by,

$$\begin{array}{|c|}\hline a \\ \hline . \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \ : \ \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline b \\ \hline . \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \ : \ \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \ ; \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline . \\ \hline . \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline x \\ \hline x \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline . \\ \hline D_i \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline x \\ \hline C_i \\ \hline \end{array} \; : \; \begin{array}{|c|}\hline \lambda \\ \hline \lambda \\ \hline \end{array} \; ; \; \begin{array}{|c|}\hline x \\ \hline A_i \\ \hline \end{array}$$

We note that the top and bottom rows of the rectangular arrays generated in the individual language will be of the form $xa^m xb^m xa^m xb^m ...x$ as there are n component grammars and the example is given in [2].

Theorem 3.17. *For* $Y \in \{I, T\}$,

(i) $2dRML = Y_{isgs}L_1(REG) \subset Y_{isgs}L_2(REG) \subset \cdots \subset Y_{isgs}L_n(REG) \subset \cdots$

(ii) $2dCFML = Y_{isgs}L_1(CF) \subset Y_{isgs}L_2(CF) \subset \cdots \subset Y_{isgs}L_n(CF) \subset \cdots$

(iii) $2dTRML = Y_{sags}L_1(REG) \subset Y_{sags}L_2(REG) \subset \cdots \subset Y_{sags}L_n(REG) \subset \cdots$

(iv) $2dTCFML = Y_{sags}L_1(CF) \subset Y_{sags}L_2(CF) \subset \cdots \subset Y_{sags}L_n(CF) \subset \cdots$

The first statement has been proved in [2] using the Example 3.12. The remaining statements can be seen similarly.

Chapter References

[1] J. Dassow and V. Mitrana. Self cross-over systems. in : Computing with bio - molecules: Theory and experiments. *(Eds.) Gh. Paun, Discrete Mathematics and Theoretical Computer Science*, Springer-Verlag:pp. 283 – 294, 1998.

[2] K. S. Dersanambika, K. G. Subramanian, and Anthonath Roslin Sagaya Mary. Image splicing grammar systems. *Proceedings on Grammar Systems Week*, 2004.

[3] Anthonath Roslin Sagaya Mary Dersanambika. k. S, Subramanian. K. G. 2D and 3D pictural networks of evolutionary processors. *IWINAC, Lecture Notes in Computer Science*, 2:92–101, 2005.

[4] D. Giammarresi and A. Restivo. *Two-dimensional languages, In "Handbook of Formal Languages"*, volume Springer Verlag,3. Springer Verlag, Vol.3, 1997.

[5] P. Helen Chandra, K. G. Subramanian, and D. G. Thomas. Parallel splicing on images. *Int. J. of pattern recognition and artificial intelligence*, 18(6):1071–1091, 2004.

[6] P. Helen Chandra, K. G. Subramanian, D. G. Thomas, and D. L. Van. A note on parallel splicing on images. In *IWCIA-2001*, volume Proceedings of the 8th International Workshop on Combinatorial Image Analysis, 2001.

[7] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory,*. Languages and Computation, Reading, MA,, 1979.

[8] K. Krithivasan, V.T. Chakaravarthy, and R. Rama. Array splicing systems. *Lecture Notes in Computer Science*, Control Cooperation and Combinatorics", Eds. Gh.Paun and A. Salomaa, 1218, Springer Verlag:346–365, 1997.

[9] Anthonath Roslin Sagaya Mary and K. G. Subramanian. Self crossover array languages. *Adv. Int. Soft Computing*, 89:291 – 298, 2011.

[10] M. Mateescu, Gh. Păun, G. Rozenberg, and A. Salomaa. Simple splicing systems. *Discrete Applied Math. 84*, 84:145–163, 1998.

[11] Y. Sakakibara and C. Ferretti. Splicing on tree-like structures. *Theoretical Computer Science*, 210(2):227–243, 1999.

[12] G. Siromoney, R. Siromoney, and K. Krithivasan. Abstract families of matrices and picture languages, *computer graphics and image processing* 1 (1972) 234 - 307.

[13] R. Siromoney, K. G. Subramanian, and K. Rangarajan. Parallel sequential rectangular arrays with tables. *Inter. J. Computer Math.*, 6:143–158, 1977.

[14] K. G. Subramanian and Anthonath Roslin Sagaya Mary. On special forms of splicing on arrays and graphs. *Triangle*, 2011.

[15] K. G. Subramanian, Anthonath Roslin Sagaya Mary, and K. S. Dersanambika. Splicing array grammar systems. *ICTAC*, pages 125–135, 2005.

Chapter 4

Pictural Networks of Evolutionary Processors

We now introduce the notion of contextual pictural networking of evolutionary processors using contextual insertion and deletion rules. These rules are a special case of contextual insertion/deletion studied by Mitrana [3].

4.1 Contextual Pictural Networks of Evolutionary Processors

A *contextual pictural network of evolutionary processors* (CPNEP) of size n is a construct

$$\Gamma = (V, N_1, N_2, \ldots, N_n, G, N_{i_0}),$$

where:

- V is an alphabet,
- for each $1 \leq i \leq n$, $N_i = (A_i, M_i, PI_i, FI_i, PO_i, FO_i)$ is the i-th evolutionary node processor of the network. The parameters of every processor are:
  - A_i is a multiset of finite support of 2D pictures over V. This multiset represents the 2D pictures existing in the i-th node at the beginning of any computation. Actually, in what follows, we consider that each 2D picture appearing in any node at any step has an arbitrarily large number of copies in that node, so

that we identify multisets by their supports. Therefore, the set A_i is the set of initial pictures in the i-th node.

– M_i is a finite set of contextual evolution rules of one of the following forms:

$A \to B$(substitution rules),
$(a, \varepsilon) \to (a, A)(r)$(right cell insertion rules),
$(\varepsilon, a) \to (A, a)(l)$(left cell insertion rules),
$\begin{pmatrix} \varepsilon \\ a \end{pmatrix} \to \begin{pmatrix} A \\ a \end{pmatrix} (u)$(up cell insertion rules)
$\begin{pmatrix} a \\ \varepsilon \end{pmatrix} \to \begin{pmatrix} a \\ A \end{pmatrix} (d)$(down cell insertion rules)

$(a, A) \to (a, \varepsilon)(r)$(right cell deletion rules),
$(A, a) \to (\varepsilon, a)(l)$(left cell deletion rules),
$\begin{pmatrix} A \\ a \end{pmatrix} \to \begin{pmatrix} \varepsilon \\ a \end{pmatrix} (u)$(up cell deletion rules)
$\begin{pmatrix} a \\ A \end{pmatrix} \to \begin{pmatrix} a \\ \varepsilon \end{pmatrix} (d)$(down cell deletion rules)

More clearly, the set of evolution rules of any processor contains either substitution, or deletion or insertion rules.

– PI_i and FI_i are subsets of V representing the input filter. This filter, as well as the output filter, is defined by random context conditions; PI_i forms the enforcing context condition and FI_i forms the forbidding context condition. A 2D picture $w \in V^{*2}$ can pass the input filter of the node processor i, if w contains each element of PI_i irrespective of the direction in which it appears, but w can contain no element of FI_i. Note that any of the random context conditions may be empty, in which case the corresponding context check is omitted.

With respect to the input filter we define the predicate

$$\rho_i(w) : w \textit{ can pass the input filter of the node processor } i.$$

– PO_i and FO_i are subsets of V representing the output filter. Analogously, a 2D picture can pass the output filter of a node processor if it satisfies the random context conditions associated with that node. Similarly, we define the predicate

$$\tau_i(w) : w \textit{ can pass the output filter of the node processor } i.$$

- $G = (\{N_1, N_2, \ldots, N_n\}, E)$ is an undirected graph called the *underlying graph* of the network. The nodes of G correspond to the evolutionary processors of the CPNEP. The edges of G, that is, the elements of E, are given in the form of sets of two nodes.

- N_{i_0} is the *output node* of the network.

By a configuration (state) of an CPNEP as above we mean an n-tuple $C = (L_1, L_2, \ldots, L_n)$, with $L_i \subseteq V^{*2}$ for all $1 \leq i \leq n$. A configuration represents the sets of 2D pictures (remember that each 2D picture appears in an arbitrarily large number of copies) which are present in any node at a given moment; clearly the initial configuration of the network is $C_0 = (A_1, A_2, \ldots, A_n)$. A configuration can change either by an *evolutionary* step or by a *communicating* step. When changing by an evolutionary step, each component L_i of the configuration is changed in accordance with the evolutionary rules associated with the node i.

Formally, we say that the configuration $C_1 = (L_1, L_2, \ldots, L_n)$ directly changes for the configuration $C_2 = (L'_1, L'_2, \ldots, L'_n)$ by an evolutionary step, written as $C_1 \rightarrow C_2$ if L'_i is the set of 2D pictures obtained by applying the rules of R_i to the 2D pictures in L_i as follows (we present one of the cases of contextual insertion/deletion, the other cases being similar):

- A node having substitution rules performs a substitution as follows: one occurrence of the lefthand side of a substitution rule is replaced by the righthand side of that rule. If a letter can be replaced by more than one new letter (there are more than one substitution rules with the same lefthand side), then this replacement will be done in different copies of the original 2D picture, thus resulting in a multiset of new pictures, in which each 2D picture appears in an arbitrary number of copies.

  If the procedure above is applicable to more than one occurrence of the same letter, then each such occurrence will be replaced accordingly, thus resulting again in an even larger multiset of new 2D pictures, in which each 2D picture appears in an arbitrary number of copies.

- A node having a left cell insertion rule

  $(\varepsilon, a) \rightarrow (A, a)(l)$ performs a cell insertion as follows: A is inserted on the left of the cell containing a. Similarly for the other insertion

rules. Each newly inserted cell is formed by a combination of the symbols appearing in the righthand side of the cell insertion rules of the nodes, thus resulting in a multiset of new 2D pictures, in which each 2D picture appears in an arbitrary number of copies.

- A node having a left cell deletion rule

  $(A, a) \to (\varepsilon, a)(l)$ performs a cell deletion as follows: A is deleted on the left of the cell containing a. Similarly for the other deletion rules. A cell can be deleted if it contains symbols in the lefthand side of the cell deletion rule, only.

More precisely, since an arbitrarily large number of copies of each 2D picture is available in every node, after an evolutionary step, in each node one gets again an arbitrarily large number of copies of any 2D picture which can be obtained by using any rule associated with that node as defined above. By definition, if L_i is empty for some $1 \leq i \leq n$, then L'_i is empty as well.

We say that the configuration $C_1 = (L_1, L_2, \ldots, L_n)$ directly changes for the configuration $C_2 = (L'_1, L'_2, \ldots, L'_n)$ by a communication step, written as $C_1 \vdash C_2$ if for every $1 \leq i \leq n$,

$$L'_i = L_i \backslash \{w \in L_i \mid \tau_i(w) = \underline{true}\} \cup \bigcup_{\{N_i, N_j\} \in E} \{x \in L_j \mid (\tau_j(x) \wedge \rho_i(x)) = \underline{true}\}.$$

Note that the 2D pictures which can pass the output filter of a node are sent out irrespective of they being received by any other node.

Let $\Gamma = (V, N_1, N_2, \ldots, N_n, G, N_{i_0})$ be an CPNEP. By a computation in Γ we mean a sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is the initial configuration, $C_{2i} \to C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$ for all $i \geq 0$.

If the sequence is finite, we have a finite computation. The result of any finite or infinite computation is a 2D picture language which is collected in a designated node called the output (master) node of the network. If $C_0, C_1, \ldots$ is a computation, then all 2D pictures existing in the node N_{i_0} at some step t – the i_0-th component of C_t – belong to the 2D picture language generated by the network. Let us denote this language by $L(\Gamma)$.

Example 4.1. *Consider the CPNEP generating pictures of staircases*

$$\Gamma = (\{a, A, B\}, N_1, N_2, N_3, N_4, N_5, K_5, N_5)$$

$$N_1 = (\{a\}, (a,\varepsilon) \to (a,A)(r), (A,\varepsilon) \to (A,A)(r), \\ (A,\varepsilon) \to (A,B)(r), (C,\varepsilon) \to (C,A)(r), \{C\}, \{A,B\}, \{B\}, \phi$$

$$N_2 = (\phi, A \to a, \{A,B\}, \phi, \{a,B\}, \{A\})$$

$$N_3 = (\phi, \begin{pmatrix} \varepsilon \\ B \end{pmatrix} \to \begin{pmatrix} B \\ B \end{pmatrix}(u), \begin{pmatrix} \varepsilon \\ B \end{pmatrix} \to \begin{pmatrix} C \\ B \end{pmatrix}(u), \{B\}, \{A\}, \{C\}, \phi)$$

$$N_4 = (\phi, B \to a, \{B,C\}, \{A\}, \{a,C\}, \{B\})$$

$$N_5 = (\phi, C \to a, \{a,C\}, \{A,B\}, \phi, \{a\})$$

Example 4.2. *Consider the CPNEP generating pictures of token-T*

$$\Gamma = (\{a,A,B\}, N_1, N_2, K_2, N_2)$$

$$N_1 = (\{aAa\}, (a,\varepsilon) \to (a,a)(r), (\varepsilon,a) \to (a,a)(l), \\ \begin{pmatrix} A \\ \varepsilon \end{pmatrix} \to \begin{pmatrix} A \\ A \end{pmatrix}(d), \begin{pmatrix} A \\ \varepsilon \end{pmatrix} \to \begin{pmatrix} A \\ B \end{pmatrix}(d), \phi, \phi, \{B\}, \phi)$$

$$N_2 = (A \to a, B \to a, \{B\}, \phi, \phi, \{a\})$$

Theorem 4.1. *(i) The families $\mathcal{L}$(2D-CPNEP) and $\mathcal{L}$(BPG) intersect.*

(ii) The family $\mathcal{L}$(2D-CPNEP) also intersects $\mathcal{L}$(CFPG).

The above statements of the theorem are clear from Examples 4.1 and 4.2.

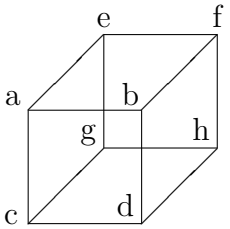
4.2 3D-Pictural Network of Evolutionary Processors

4.2.1 Three Dimensional Picture Languages

For a given alphabet V, a 3D picture p of size $l \times m \times n$ over V is a 3D array of the form $p = (a_{ijk})_{i\in\overline{1,l},\ j\in\overline{1,m},\ k\in\overline{1,n}}$ with $a_{ijk} \in V$ for $i \in \overline{1,l},\ j \in \overline{1,m},\ k \in \overline{1,n}$. We denote V^{***} the set of all 3D pictures over V (including the empty picture denoted by λ). A 3D picture language over V is a subset of V^{***} . A 3D subpicture of a 3D picture p is a 3D sub array of p. A $(2 \times 2 \times 2)$ subpicture of p is called a cube of p. The set of all cubes of p is denoted by $B_{2,2,2}(p)$. In the sequel, we will identify the boundaries of a picture by surrounding it with the marker #. A picture p bounded by markers # is denoted by $\hat{p}$.

4.2.2 Recognizability of 3D-rectangles by cubic system

Here we consider local and recognizable 3D-rectangular languages [1].
A 3D-rectangle of the form given below is called a cube over the alphabet $\{a, b, c, d, e, f, g, h\}$.



Hereafter we follow this notation for representing a cube. Given a 3D-rectangle p, $B_{i,j,k}(p)$ denotes the set of all sub 3D-rectangles p of size (i, j, k), cube is a 3D-rectangle of size $(2, 2, 2)$.We denote by $\Sigma^{l\times m\times n}$ the set of 3D-rectangles of size (l, m, n) over the alphabet Σ. $B_{2,2,2}$ is in fact a set of cubes.

Definition 4.1. *Let Σ be a finite alphabet. The 3D-rectangular language $L \subset \Sigma^{\star\star\star}$ is local if there exists a set of cubes $\bigtriangleup \subseteq (\Sigma \cup \{\#\})^{2\times2\times2}$ such that $L = \{p \in \Sigma^{\star\star\star} | B_{2,2,2}(\hat{p}) \subseteq \bigtriangleup\}$.*

The family of local picture languages, denoted by 3D-LOC, is a generalization of the two dimensional case of local languages defined in [2]. Given a set of cubes Δ, the 3D-local picture language L generated by Δ is denoted by $L(\Delta)$.

Definition 4.2. *Let Σ be a finite alphabet. A 3D-rectangular language $L \subseteq \Sigma^{\star\star\star}$is called recognizable if there exists a local 3D-rectangular language L' (given by a set $\bigtriangleup$ of cubes) over an alphabet of Γ and a mapping $\Pi : \Gamma \to \Sigma$ such that $L = \Pi(L')$.*

Example 4.3. *The language L of 3D-rectangular pictures over single alphabet of any size (l, m, n) surrounded by # symbol on all 6 faces is a local 3D rectangular language [1]. Note that the 3D-rectangular languages over one letter alphabet with all sides of equal length is not local but it is a recognizable languages.*

4.2.3 The Family 3D-PNEP

We now extend the notion of PNEP to 3D rectangular pictures

A *3D- pictural network of evolutionary processors* (3D-PNEP for short) of size n is a construct

$$\Gamma = (V, N_1, N_2, \ldots, N_n, G, N_{i_0}),$$

where the components are as in [4] except the objects are 3D rectangular pictures and evolution rules are as follows.

$a \to b$, $a, b \in V$ (substitution rules),

$a \to \varepsilon(x)(b)$, $a \in V$ (back face deletion rules),

$a \to \varepsilon(x)(f)$, $a \in V$ (front face deletion rules),

$a \to \varepsilon(y)(r)$, $a \in V$ (right face deletion rules),

$a \to \varepsilon(y)(l)$, $a \in V$ (left face deletion rules),

$a \to \varepsilon(z)(t)$, $a \in V$ (top face deletion rules),

$a \to \varepsilon(z)(bo)$, $a \in V$ (bottom face deletion rules),

$\varepsilon \to a(x)(b)$, $a \in V$ (back face insertion rules),

$\varepsilon \to a(x)(f)$, $a \in V$ (front face insertion rules),

$\varepsilon \to a(y)(r)$, $a \in V$ (right face insertion rules),

$\varepsilon \to a(y)(l)$, $a \in V$ (left face insertion rules),

$\varepsilon \to a(z)(t)$, $a \in V$ (top face insertion rules),

$\varepsilon \to a(z)(bo)$,$a \in V$ (bottom face insertion rules),

$a \to \varepsilon(x)(/)$, $a \in V$ (front or back face deletion rules),

$a \to \varepsilon(y)(-)$, $a \in V$ (left or right face deletion rules),

$a \to \varepsilon(z)(|)$, $a \in V$ (top or bottom face deletion rules),

$\varepsilon \to a(x)(/)$, $a \in V$ (front or back face insertion rules),

$\varepsilon \to a(z)(|)$, $a \in V$ (top and bottom face insertion rules),

$\varepsilon \to a(y)(-)$, $a \in V$ (left or right face insertion rules),

More clearly, the set of evolution rules of any processor contains either substitution, or deletion or insertion rules.

Let $\Gamma = (V, N_1, N_2, \ldots, N_n, G, N_{i_0})$ be an 3D-PNEP. By a computation in Γ we mean a sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is the initial configuration, $C_{2i} \to C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$ for all $i \geq 0$ where $\to$ denotes evolution and $\vdash$ denotes communication.

If the sequence is finite, we have a finite computation. The result of any finite or infinite computation is a 3D rectangular picture language which is collected in a designated node called the output (master) node of the network. If $C_0, C_1, \ldots$ is a computation, then all 3D rectangular pictures existing in the node N_{i_0} at some step t – the i_0-th component of C_t – belong to the 3D rectangular picture language generated by the network. Let us denote this language by $L(\Gamma)$.

4.2.4 Comparison

We start with some examples which constitute our basis for comparing the class of 3D rectangular picture languages generated by 3D-PNEP with other 3D rectangular picture generating devices.

Example 4.4. *Let L_1 denote the set of all 3D rectangular pictures p over the alphabet $\{a\}$. The 3D rectangular language L_1 described as $L_1 = \{p \in \{a\}^{***} | x(p), y(p), z(p) \geq 1\}$. The language L_1 can be generated in the fourth node N_4 by the following 3D-PNEP of size 4.*

$\Gamma = (\{a, W\}, N_1, N_2, N_3, N_4, K_4, N_4)$, *where*
$N_1 = (\{\varepsilon\}, \{\varepsilon \to a(x)(/), \varepsilon \to W(x)(/)\}, \phi, \{A, T\}, \{W\}, \phi)$
$N_2 = (\phi, \{\varepsilon \to a(y)(-), \varepsilon \to T(y)(-)\}, \{W\}, \phi, \{T\}, \phi)$
$N_3 = (\phi, \{T \to a, W \to a\}, \{W, T\}, \phi, \{, a\}, \{W, T\})$
$N_4 = (\phi, \{\varepsilon \to a(z)(|)\}, \{a\}, \{T\}, \phi, \{a\})$
Here $L(\Gamma) = L_1$.

Example 4.5. *Let X be the 3D-rectangular picture of size $(2, 2, 2)$ over the alphabet $\{a\}$. Let L_2 be the set of all 3D-rectangular pictures p over*

the alphabet $\{a\}$ *with all sides equal. This 3D-rectangular language* L_2 *can be formally described as*

$$L_2 = \{p \in \{a\}^{***} \mid x(p) = y(p) = z(p) \geq 1\}.$$

L_2 *can be generated in the output node* N_5 *by the following complete 3D-PNEP of size five:*

$$\Gamma = (\{a, A, B, C\}, N_1, N_2, N_3, N_4, N_5, K_5, N_5),$$

where $N_1 = (\{X\}, \{\varepsilon \to A(x)(b)\}, \{a\}, \{A, B, C\}, \{a, A\}, \emptyset)$
$N_2 = (\emptyset, \{\varepsilon \to B(z)(bo)\}, \{a, A\}, \{B, C\}, \{a, A, B\}, \emptyset)$
$N_3 = (\emptyset, \{\varepsilon \to C(y)(l)\}, \{a, A, B\}, \{C\}, \{a, A, B, C\}, \emptyset)$
$N_4 = (\emptyset, \{A \to a, B \to a, C \to a\}, \{a, A, B, C\}, \emptyset, \{a\}, \{A, B, C\})$
$N_5 = (\emptyset, \{a \to a\}, \{a\}, \{B, C\}, \emptyset, \{a\})$.
Here $L(\Gamma) = L_2$

Now we set:

3D-LOC is the class of local 3D picture languages [1].

3D-REC is the class of recognizable picture languages

$\mathcal{L}$(3D-PNEP) is the class of 3D-rectangular picture languages generated by 3D-PNEP's.

Now we are ready to give the result:

Theorem 4.2. *The family* $\mathcal{L}$*(3D-PNEP)*
(i) is incomparable with family 3D-LOC but not disjoint.
(ii) intersects the family 3D-REC.

Proof. (i) The language of 3D-rectangular pictures over $\{a\}$ from Example 4.4 is in 3D-LOC ([1]) and $\mathcal{L}$(3D-PNEP). On the other hand, the language from Example 4.5 is not in 3D-LOC ([1]) but is in $\mathcal{L}$(3D-PNEP). The language of 3D-arrays of equal size in which all the diagonal elements are 1 and the remaining elements are 0 and it is known to be in 3D-LOC [1] but it cannot be generated by any 3D-PNEP as, informally speaking, 3D-PNEP's have no ability to fix the position of symbols 1 in the diagonal when using face insertion rule. Hence 3D-LOC and $\mathcal{L}$(3D-PNEP) are incomparable.
(ii) The language generated by the 3D-PNEP in Example 4.5 is in 3D-REC[1] □

Chapter References

[1] K. S. Dersanambika and K. Krithivasan. Recognizability of 3D rectangular pictures. *Paper presented at the 12th International Conference of Jangeon Mathematical Society, University of Mysore*, Dec. 2003.

[2] D. Giammarresi and A. Restivo. *Two-dimensional languages, In "Handbook of Formal Languages"*, volume Springer Verlag,3. Springer Verlag, Vol.3, 1997.

[3] V. Mitrana. contextual insertion and deletion. *The publishing of house of the Romanian Academic*, Mathematical Linguistre and related topic(Gh.Paun ed):271–278, 1994.

[4] V. Mitrana, K. G. Subramanian, and M. Tataram. Pictural networks of evolutionary processors. *RomJIST*, 6(1-2):189–199, 2003.

Chapter 5

Tiling Rule Systems

In this Chapter we give the new formalism called Tiling Rule Systems in [1, 2] which is based on tiles instead of domino rules, where the picture is generated by assembling tiles in columns(rows) with given rules over tiles. This formalism is also a bio-inspired one generating pictures again with application on its columns(rows), it seems natural to extend the inspiration from self-assembling nature of the DNA sequences. Here we just try to take the inspiration and apply tiling rules that we define to assembling tiles generating picture language classes. We then study the generative power of the classes.

Section 5.1 introduces the formalism. Sections 5.2 and the next section 5.3 are with the results on TRuS which has one of the first results and the main results of the given study.

5.1 Tiling Rule Systems (TRuS)

In this section, we define the notion of tiling rule and tiling rule system.

A *general tiling rule* r over a set Θ of tiles is defined by a pair t_1, t_2 of tiles in Θ, where t_1 is the *context site* of rule r, while t_2 is the *replacement site* of rule r. Then r is denoted as $r = t_1 \rightarrow t_2$. We distinguish two types of rules: *row* and *column* rules. The context site of a row rule is denoted by a tile $t = \frac{a \quad b}{c \quad d}$, while the context site of column rules is denoted by a tile $t = \begin{matrix} a \\ c \end{matrix} \Big| \begin{matrix} b \\ d \end{matrix}$.

When a row rule r is applied, then t_2 replaces the bottom row domino of tile t_1. Similarly, when a column rule is applied then t_2 replaces the rightmost column domino of tile t_1. A rule acts together with other rules

to enlarge a picture: this fact is formalized by the notion of *rule sequence* defined below.

Definition 5.1 (row rule sequence)**.** *A sequence $S = r_1 \cdot r_2 \cdot ... \cdot r_m$ of rules is a* row rule sequence*, in short* r-sequence*, iff for each j, $1 \leq j \leq m$, it holds that $r_j = \frac{a_j\ a_{j+1}}{b_j\ b_{j+1}} \rightarrow \begin{smallmatrix} a'_j & a'_{j+1} \\ b'_j & b'_{j+1} \end{smallmatrix}$ (tiles in r_j horizontally overlaps with tiles in r_{j+1}).*

Given a row rule sequence S in the above Definition 5.1, the application of rules in S defines the pseudo-pictures $p_{(S,r)}$ and $q_{(S,r)}$ consisting of 2 rows and $m+1$ columns called the *context site* and *replacement site* of S respectively, such that $p_{(S,r)}[1,j] = a_j$, $p_{(S,r)}[2,j] = b_j$, while $q_{(S,r)}[1,j] = a'_j$ and $q_{(S,r)}[2,j] = b'_j$ for each column j.

Definition 5.2 (column rule sequence)**.** *A sequence $S = r_1 \cdot r_2 \cdot ... \cdot r_n$ of rules is a* column rule sequence*, in short* c-sequence*, iff for each i, $1 \leq i \leq n$, it holds that $r_i = \begin{smallmatrix} a_i & b_i \\ a_{i+1} & b_{i+1} \end{smallmatrix} \rightarrow \begin{smallmatrix} a'_i & b'_i \\ a'_{i+1} & b'_{i+1} \end{smallmatrix}$ (tiles in r_i vertically overlaps with tiles in r_{i+1}).*

Given a column rule sequence S in the above Definition 5.2, the application of S produces the pseudo-pictures $p_{(S,c)}$ and $q_{(S,c)}$ consisting of $n+1$ rows and 2 columns called the *context site of S* and *replacement site of S* respectively, such that $p_{(S,c)}[i,1] = a_i$, $p_{(S,c)}[i,2] = b_i$, while $q_{(S,c)}[i,1] = a'_i$, $q_{(S,c)}[i,2] = b'_i$ for each row i.

Example 5.1. *Let the row rules be given by, $r_1 = \frac{a\ b}{e\ f} \rightarrow \begin{smallmatrix} a' & b' \\ e' & f' \end{smallmatrix}$, $r_2 = \frac{b\ c}{f\ g} \rightarrow \begin{smallmatrix} b' & c' \\ f' & g' \end{smallmatrix}$ and $r_3 = \frac{c\ d}{g\ h} \rightarrow \begin{smallmatrix} c' & d' \\ g' & h' \end{smallmatrix}$. Then $S = r_1 \cdot r_2 \cdot r_3$ is a r-sequence, picture $p_{(S,r)}$ is the context site of S, while picture $q_{(S,r)}$ is the replacement site of S in figure 5.1.*

Example 5.2. *Let the column rules be given by, $r_1 = \begin{smallmatrix} a & e \\ b & f \end{smallmatrix} \rightarrow \begin{smallmatrix} a' & e' \\ b' & f' \end{smallmatrix}$, $r_2 = \begin{smallmatrix} b & f \\ c & g \end{smallmatrix} \rightarrow \begin{smallmatrix} b' & f' \\ c' & g' \end{smallmatrix}$ and $r_3 = \begin{smallmatrix} c & g \\ d & h \end{smallmatrix} \rightarrow \begin{smallmatrix} c' & g' \\ d' & h' \end{smallmatrix}$. Then $S = r_1 \cdot r_2 \cdot r_3$ is a c-sequence, such that the context site of S is $p_{(S,c)}$ and the replacement site of S is $q_{(S,c)}$ in figure 5.1.*

Now a rule sequence S can be applied to pictures to generate new ones, whenever the context site of S is inside the picture.

$$p_{(S,r)} = \boxed{\begin{array}{cccc} a & b & c & d \\ e & f & g & h \end{array}}, \quad q_{(S,r)} = \boxed{\begin{array}{cccc} a' & b' & c' & d' \\ e' & f' & g' & h' \end{array}},$$

$$p_{(S,c)} = \boxed{\begin{array}{cc} a & e \\ b & f \\ c & g \\ d & h \end{array}}, \quad q_{(S,c)} = \boxed{\begin{array}{cc} a' & e' \\ b' & f' \\ c' & g' \\ d' & h' \end{array}}$$

Figure 5.1: Rule sequence

Definition 5.3 (application of row rule sequences)**.** *Let $S = r_1 \cdot r_2 \cdot \ldots \cdot r_{m-1}$ be a r-sequence having context site $p_{(S,r)}$. Then S can be applied to the picture $\hat{p}$ of size (n, m) at the i-th row of $\hat{p}$, where $1 \leq i < n$ iff $\hat{p}^r[i..i{+}1] = p_{(S,r)}$.*

Definition 5.4 (application of column rule sequences)**.**
Let $S = r_1 \cdot r_2 \cdot \ldots \cdot r_{n-1}$ be a c-sequence having context site $p_{(S,c)}$. Then S can be applied to the picture $\hat{p}$ of size (n, m) at the i-th column of $\hat{p}$, where $1 \leq i < m$ iff $\hat{p}^c[i..i+1] = p_{(S,c)}$.

Observe that by the above definitions, a rule sequence S can be applied at the i-th row (or column) of a picture $\hat{p}$ if the context site of S is given by the i-th and $(i+1)$-th rows (or columns) of $\hat{p}$.

$$\hat{p} \begin{array}{|ccccc|} \hline \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline a_{i,1} & a_{i,2} & \cdots & a_{i,m} & a_{i,m} \\ \hline a_{i+1,1} & a_{i+1,2} & \cdots & a_{i+1,m} & a_{i+1,m} \\ \hline a_{i+2,1} & a_{i+2,2} & \cdots & a_{i+2,m} & a_{i+2,m} \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline \end{array} \rightarrow \begin{array}{|ccccc|} \hline \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline a_{i,1} & a_{i,2} & \cdots & a_{i,m} & a_{i,m} \\ \hline a'_1 & a'_2 & \cdots & a'_m & a'_m \\ \hline b'_1 & b'_2 & \cdots & b'_m & b'_m \\ \hline a_{i+2,1} & a_{i+2,2} & \cdots & a_{i+2,m} & a_{i+2,m} \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline \end{array}$$

Figure 5.2: Application of S to the i-th row of $\hat{p}$

Then a new picture can be generated by the application of S to picture $\hat{p}$ by means of the replacement site $q_{(S,r)}$ of S. Indeed, the first row (or column) of $q_{(S,r)}$ (or $q_{(S,c)}$) will replace the i+1-th row (or column) of $\hat{p}$, while the second row (or column) of $q_{(S,r)}$ (or $q_{(S,c)}$) will be inserted in between the replaced row (or column) and the $(i+2)$-th row (or column) of $\hat{p}$ (if present); see Figure 5.2 and 5.3.

$$\hat{p}\begin{array}{|c|c|c|c|c|}\hline \cdots & a_{1,i} & a_{1,i+1} & a_{1,i+2} & \cdots \\ \cdots & a_{2,i} & a_{2,i+1} & a_{2,i+2} & \cdots \\ \ddots & \vdots & \vdots & \vdots & \ddots \\ \cdots & a_{n,i} & a_{n,i+1} & a_{n,i+2} & \cdots \\ \cdots & a_{n,i} & a_{n,i+1} & a_{n,i+2} & \cdots \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|c|c|}\hline \cdots & a_{1,i} & a'_1 & b'_1 & a_{1,i+2} & \cdots \\ \cdots & a_{2,i} & a'_2 & b'_2 & a_{2,i+2} & \cdots \\ \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \\ \cdots & a_{n,i} & a'_n & b'_n & a_{n,i+2} & \cdots \\ \cdots & a_{n,i} & a'_n & b'_n & a_{n,i+2} & \cdots \\ \hline \end{array}$$

Figure 5.3: Application of S to the i-th column of $\hat{p}$

Observe that the application of a rule sequence to a picture produces an array over alphabet $\Sigma \cup \Delta$ which could not be a canonical or pseudo-canonical one. Thus in following Definitions 5.5 and 5.6 we will only refer to the application to pictures $\hat{p}$ of rule sequences generating new pictures $\hat{q}$.

Definition 5.5 (row derived picture)**.** *Let S be a r-sequence such that S can be applied at the i-th row of a picture $\hat{p}$ of size (n, m). Then the picture $\hat{q}$ of size $(n+1, m)$ is* derived from $\hat{p}$ by S *iff the following holds:*

$\hat{q}^r[1..i] = \hat{p}^r[1..i]$, $\hat{q}^r[i+1..i+2]$ is equal to the replacement site $q_{(S,r)}$ of S, and $\hat{q}^r[i+3..n+1] = \hat{p}^r[i+2..n]$ (if present).

Definition 5.6 (column derived picture)**.** *Let S be a c-sequence such that S can be applied to a picture $\hat{p}$ of size (n, m). Then the picture $\hat{q}$ of size $(n, m+1)$ is* derived from $\hat{p}$ by S *iff the following holds:*

$\hat{q}^c[1..i] = \hat{p}^c[1..i]$, $\hat{q}^c[i+1..i+2]$ is equal to the replacement site $q_{(S,c)}$ of S, and $\hat{q}^c[i+3..m+1] = \hat{p}^c[i+2..m]$ (if present).

The rule sequence S of the above two definitions is called *admissible* for picture $\hat{p}$ since it allows the generation of a new array which is still a picture. The application of S to $\hat{p}$ to derive picture $\hat{q}$ is denoted by $\hat{p} \rightarrow_S \hat{q}$. The i-iterated application of S over a picture $\hat{p}$ to generate picture $\hat{q}$ is denoted by $\hat{p} \rightarrow_S^i \hat{q}$

A picture $\hat{p}'$ is *derived* from a picture $\hat{p}$, denoted by $\hat{p} \Rightarrow_R \hat{p}'$, iff there exist rule sequences $S_1, \cdots, S_k$ such that $\hat{p} \rightarrow_{S_1} \hat{p}_1 \rightarrow_{S_2} \hat{p}_2 ... \rightarrow_{S_k} \hat{p}'$. Then $\hat{p} \rightarrow_{S_1} \hat{p}_1 \rightarrow_{S_2} \hat{p}_2 ... \rightarrow_{S_k} \hat{p}'$ is called *derivation of $\hat{p}'$ from $\hat{p}$*, while $d = S_1, \cdots, S_k$ is the *derivation sequence* applied to derive $\hat{p}'$ from $\hat{p}$.

A derivation sequence $d = S_1, \cdots, S_k$ is called *strict* iff for each i, with $1 \leq i < k$, the context site of S_{i+1} is the replacement site of S_i.

A strict derivation sequence is *unambiguous* if given sequence S_i it can be only followed by sequence S_{i+1}, given a set of available rules.

Given an initial finite set of pictures and a finite set of rules, then rules can be combined to produce c-sequences or r-sequences that can be

applied iteratively to the initial pictures to generate an infinite language of pictures. This process of generating pictures is described by the notion of a tiling rule system and language generated by such type of systems.

Definition 5.7 (Tiling Rule System (TRuS)). *A tiling rule system, in short TRuS system, is a triple* $T = (I, R, \Sigma)$ *where* I *is a finite set of canonical pictures, called* initial set *and* R *is a set of rules over alphabet* $\Sigma \cup \Delta$*, where* Δ *is a special two symbols alphabet disjoint from* Σ*.*

Thus let us define the language generated by a TRuS system.

Definition 5.8 (language). *Given a TRuS system* T*, then the language generated by* T*, denoted by* $L(T)$ *is the set* $\{p : \hat{p} \in L\}$*, where* $L = I \cup \{\hat{p_1} : \hat{p} \Rightarrow_R \hat{p_1}, \hat{p} \in I, \hat{p_1}$ *is canonical* $\}$*. Language* L *is the* canonical *language generated by the system.*

Then $\mathcal{L}$(TRuS) denotes the class of languages generated by TRuS systems.

Remark 5.1. *Assume that* $p' \in L(T)$ *and* $\hat{p} \rightarrow_{S_1} \hat{p_1} \rightarrow_{S_2} \hat{p_2} \rightarrow ... \rightarrow_{S_{k-1}} \hat{p_{k-1}} \rightarrow_{S_k} \hat{p'}$*, where* $\hat{p} \in I$*. Observe that by Definition 5.8, the intermediate pictures* $\hat{p_i}$*, with* $1 \leq i < k$ *are not necessarily canonical pictures, but are pseudo-canonical ones.*

5.2 First Results on TRuS

In this section we investigate the computational power of TRuS systems. The class of languages generated by TRuS systems properly includes the one of recognizable languages. Indeed, we first show by Proposition 5.1 that recognizable languages are generated by a subclass of TRuS systems.

The strict inclusion will follow by the fact that the language of palindromic columns can be generated by TRuS systems, while it has been proved in [3] that it is not in the class $\mathcal{L}$(TS) .

Let us first define the notion of *simple TRuS systems.*

Definition 5.9 (Simple Tiling Rule System (sTRuS)). *A tiling rule system* $T = (I, R, \Sigma)$ *is* simple*, in short sTRuS system if* R *is over alphabet* $\Sigma \cup \{\#\}$*.*

The language generated by a sTRuS system is defined by considering only *standard derivations.*

A derivation is called *standard* if and only if rules generate pictures in the language first generating a pseudo-canonical $(2, m)$ picture and then rows are added to generate a (n, m) canonical picture.

Definition 5.10 (language of sTRuS)**.** *Given a sTRuS system T, then the language generated by T, denoted by $L(T)$ is the set $\{p : \hat{p} \in L\}$, where $L = \{\hat{p_1} : p_\epsilon \Rightarrow_R \hat{p_1}, \hat{p_1}$ is canonical $\}$ and $p_\epsilon \Rightarrow_R \hat{p_1}$ is a* standard derivation.

Then $\mathcal{L}$(sTRuS) denotes the class of languages generated by sTRuS systems.

Proposition 5.1. *$\mathcal{L}(TS) \subseteq \mathcal{L}(sTRuS)$.*

Proof. Let L be a recognizable language and let $\tau = (\Sigma, \Gamma, \Theta, \pi)$ be a tiling system for L where Σ, Γ are finite alphabets, Θ is a set of tiles over $(\Gamma \cup \#)$ and $\pi : \Gamma \longrightarrow \Sigma$ is a projection. Let us define the following binary relations $=_r$ and $=_c$ over pairs of tiles in Θ: $t_1 =_c t_2$ iff $t_1 = \begin{matrix} a_1 & a_2 \\ a_3 & a_4 \end{matrix}$, $t_2 = \begin{matrix} b_1 & b_2 \\ b_3 & b_4 \end{matrix}$ where $a_2 = b_1, a_4 = b_3$, $t_1 =_r t_2$ iff $t_1 = \begin{matrix} b_1 & b_2 \\ b_3 & b_4 \end{matrix}$, $t_2 = \begin{matrix} c_1 & c_2 \\ c_3 & c_4 \end{matrix}$ where $b_3 = c_1, b_4 = c_2$.

In the following we define a TRuS -system $T = (I, R, A)$ for generating L, where I consists of the empty picture $\begin{matrix} \# & \# \\ \# & \# \end{matrix}$ and alphabet A consists of $\Sigma \cup \Gamma \cup A'$, being $A' = \{[a\ b] : a, b \in \Gamma\}$.

Now, the set R of rules are listed below and are grouped according to the pair of tiles in Θ related by the relations $=_c$ and $=_r$, respectively. Indeed, rules should reproduce the tiling of a local picture and at the same time the projection of the local language. In order to do so, given a pair of tiles t_1, t_2 such that $t_1 =_r t_2$, we build a row rule r having the replacement site given by tile t such that the upper row of t (i.e. $t^r[1]$) will project the upper row domino of tile t_2, while the bottom row of t (i.e. $t^r[2]$) "memorizes" by using symbols in A' the tile t_2 that will have the upper row projected. Similarly, for the context-site of rule r. More precisely, given a pair of tiles where $\begin{matrix} a\ b \\ c\ d \end{matrix} =_r \begin{matrix} c\ d \\ e\ f \end{matrix}$ then we should build a row rule of the form $\begin{matrix} \pi(a)\ \pi(b) \\ [a\ c]\ [b\ d] \end{matrix} \rightarrow \begin{matrix} \pi(a)\ \pi(b) \\ [a\ c]\ [b\ d] \end{matrix}$.

Similarly we build rules for pair of tiles related by the $=_c$ relation. Clearly, when tiles have border symbols then the construction of rules will

be specific, as described below. In the following column rules are grouped into the set $R_{c,U} = R_{c,U,1} \cup R_{c,U,2} \cup R_{c,U,3} \cup R_{c,U,4}$ of column rules that are derived by every pair of border tiles in Θ related by the binary relation $=_c$.

Note that rules in $R_{c,U}$ only form c-sequences of length one that can be combined to form derivation sequences that apply to the initial empty picture to reproduce the tiling of the uppermost rows of a picture in L. Then row rules are grouped into three sets. The set $R_{r,L} = R_{r,L,1} \cup R_{r,L,2} \cup R_{r,L,3} \cup R_{r,L,4}$ groups rules, called *l-type rules*, that are based on pair of left most column (bordered) tiles, while set $R_{r,R} = R_{r,R,1} \cup R_{r,R,2} \cup R_{r,R,3} \cup R_{r,R,4}$ groups rules, called *r-type rules*, that are based on rightmost pairs of (bordered) tiles in Θ.

Finally, set $R_{r,M} = R_{r,M,1} \cup R_{r,M,2} \cup R_{r,M,3} \cup R_{r,M,4}$ groups6 rules, called *intermediate rules or m-type rules*, that are based on tiles that are non bordered on the left and right column dominoes and that form r-sequences starting with a l-type rule and ending with a r-type rule.

$$R_{c,U,1} = \begin{array}{c|c} \# & \# \\ \# & \# \end{array} \rightarrow \begin{array}{cc} \# & \# \\ a & \# \end{array} : \begin{array}{cc} \# & \# \\ \# & a \end{array} =_c \begin{array}{cc} \# & \# \\ a & \# \end{array},$$

$$R_{c,U,2} = \begin{array}{c|c} \# & \# \\ \# & \# \end{array} \rightarrow \begin{array}{cc} \# & \# \\ a & [a\ b] \end{array} : \begin{array}{cc} \# & \# \\ \# & a \end{array} =_c \begin{array}{cc} \# & \# \\ a & b \end{array},$$

$$R_{c,U,3} = \begin{array}{c|c} \# & a \\ \# & [a\ b] \end{array} \rightarrow \begin{array}{cc} \# & \# \\ b & [b\ c] \end{array} : \begin{array}{cc} \# & \# \\ a & b \end{array} =_c \begin{array}{cc} \# & \# \\ b & c \end{array},$$

$$R_{c,U,4} = \begin{array}{c|c} \# & a \\ \# & [a\ b] \end{array} \rightarrow \begin{array}{cc} \# & \# \\ b & \# \end{array} : \begin{array}{cc} \# & \# \\ a & b \end{array} =_c \begin{array}{cc} \# & \# \\ b & \# \end{array}.$$

$$R_{r,L,1} = \begin{array}{cc} \# & \# \\ \# & a \end{array} \rightarrow \begin{array}{cc} \# & \pi(a) \\ \# & \# \end{array} : \begin{array}{cc} \# & \# \\ \# & a \end{array} =_r \begin{array}{cc} \# & a \\ \# & \# \end{array},$$

$$R_{r,L,2} = \begin{array}{cc} \# & \# \\ \# & a \end{array} \rightarrow \begin{array}{cc} \# & \pi(a) \\ \# & [a\ b] \end{array} : \begin{array}{cc} \# & \# \\ \# & a \end{array} =_r \begin{array}{cc} \# & a \\ \# & b \end{array},$$

$$R_{r,L,3} = \begin{array}{cc} \# & \pi(a) \\ \# & [a\ b] \end{array} \rightarrow \begin{array}{cc} \# & \pi(b) \\ \# & [b\ c] \end{array} : \begin{array}{cc} \# & a \\ \# & b \end{array} =_r \begin{array}{cc} \# & b \\ \# & c \end{array},$$

$$R_{r,L,4} = \begin{array}{cc} \# & \pi(a) \\ \# & [a\ b] \end{array} \rightarrow \begin{array}{cc} \# & \pi(b) \\ \# & \# \end{array} : \begin{array}{cc} \# & a \\ \# & b \end{array} =_r \begin{array}{cc} \# & b \\ \# & \# \end{array}.$$

$$R_{r,R,1} = \begin{array}{cc} \# & \# \\ \hline a & \# \end{array} \rightarrow \begin{array}{cc} \pi(a) & \# \\ \# & \# \end{array} : \begin{array}{cc} \# & \# \\ a & \# \end{array} =_r \begin{array}{cc} a & \# \\ \# & \# \end{array},$$

$$R_{r,R,2} = \begin{array}{cc} \# & \# \\ \hline a & \# \end{array} \rightarrow \begin{array}{cc} \pi(a) & \# \\ [a\ b] & \# \end{array} : \begin{array}{cc} \# & \# \\ a & \# \end{array} =_r \begin{array}{cc} a & \# \\ b & \# \end{array},$$

$$R_{r,R,3} = \begin{array}{cc} \pi(a) & \# \\ \hline [a\ b] & \# \end{array} \rightarrow \begin{array}{cc} \pi(b) & \# \\ [b\ c] & \# \end{array} : \begin{array}{cc} a & \# \\ b & \# \end{array} =_r \begin{array}{cc} b & \# \\ c & \# \end{array},$$

$$R_{r,R,4} = \begin{array}{cc} \pi(a) & \# \\ \hline [a\ b] & \# \end{array} \rightarrow \begin{array}{cc} \pi(b) & \# \\ \# & \# \end{array} : \begin{array}{cc} a & \# \\ b & \# \end{array} =_r \begin{array}{cc} b & \# \\ \# & \# \end{array}.$$

$$R_{r,M,1} = \begin{array}{cc} \# & \# \\ \hline a & b \end{array} \rightarrow \begin{array}{cc} \pi(a) & \pi(b) \\ \# & \# \end{array} : \begin{array}{cc} \# & \# \\ a & b \end{array} =_r \begin{array}{cc} a & b \\ \# & \# \end{array},$$

$$R_{r,M,2} = \begin{array}{cc} \# & \# \\ \hline a & b \end{array} \rightarrow \begin{array}{cc} \pi(a) & \pi(b) \\ [a\ c] & [b\ d] \end{array} : \begin{array}{cc} \# & \# \\ a & b \end{array} =_r \begin{array}{cc} a & b \\ c & d \end{array},$$

$$R_{r,M,3} = \begin{array}{cc} \pi(a) & \pi(b) \\ \hline [a\ c] & [b\ d] \end{array} \rightarrow \begin{array}{cc} \pi(c) & \pi(d) \\ [c\ e] & [d\ f] \end{array} : \begin{array}{cc} a & b \\ c & d \end{array} =_r \begin{array}{cc} c & d \\ e & f \end{array},$$

$$R_{r,M,4} = \begin{array}{cc} \pi(a) & \pi(b) \\ \hline [a\ c] & [b\ d] \end{array} \rightarrow \begin{array}{cc} \pi(c) & \pi(d) \\ \# & \# \end{array} : \begin{array}{cc} a & b \\ c & d \end{array} =_r \begin{array}{cc} c & d \\ \# & \# \end{array}.$$

Now, assume that $p \in L$ is a picture of size (n, m) that is the projection of a local picture q. Then we show how the picture is generated by the iterations of the above defined rules.

Assume first that $n = m = 1$. Assume that $p = \pi(a)$, for $a \in \Gamma$. Then, there exists a rule r_0, with $r_0 \in R_{c,U,1}$ that applies to the empty picture to generate the pseudo-canonical picture $\hat{q}^r[1..2]$ with the single element a. Then, there exists a r-sequence $r_1 \cdot r_2$ with $r_1 \in R_{r,L,1}$ and $r_2 \in R_{r,R,1}$ such that is applied at the context site given by the pseudo-canonical picture $\hat{q}^r[1..2]$ producing the canonical picture $\hat{p}$.

Assume now that $n = 1$ and $m > 1$. We first prove the generation of the pseudo-canonical picture $\hat{q}^r[1..2]$ (the picture made by the first 2 rows of $\hat{q}$). In the following define tile $t_{i,j}$ as the sub-picture of $\hat{q}$ of size $(2, 2)$ with the first element consisting of $\hat{q}_{i,j}$. Clearly, it holds that $t_{1,j} =_c t_{1,j+1}$ for each j, $1 \leq j < m + 2$ and thus there exists a column rule $r_j \in R_{c,U}$ corresponding to this pair of tiles, where $r_j \in R_{c,U,2}$ if $j = 1$ and $r_j \in R_{c,U,4}$ if $j = m + 1$, otherwise $r_j \in R_{c,U,3}$. It follows that the derivation sequence d consisting of the c-sequences $S_1, S_2, ..., S_{m+1}$, where $S_j = r_j$ will produce the tiling of $\hat{q}^r[1..2]$ as required.

Once $\hat{q}^r[1..2]$ is generated, then it is easy to verify the existence of rules $r_j^0 \in R_{r,M,1}$ for $1 < j < m + 1$, based on the pair of tiles $t_{2,j} =_r t_{3,j}$ that will produce the projection of symbols in tile $t_{2,j}$. Similarly, there is a rule $r_1^0 \in R_{r,L,1}$ based on the pair of tiles $t_{2,1} =_r t_{3,1}$ and rule $r_{m+1}^0 \in R_{r,R,1}$ corresponding to the pair of tiles $t_{2,m+1} =_r t_{3,m+1}$ that will produce the

projection of the first and last symbol of the picture $\hat{q}^r[1..2]$.

Then the r-sequence $r_1^0 \cdot r_2^0 \cdot r_j^0 \cdot r_{m+1}^0$ is applied to picture $\hat{q}^r[1..2]$ having the required context site, thus producing the canonical picture $\hat{p}^r[1..3]$.

Assume now that $n > 1$ and $m \geq 1$. We first list the rule sequences that will be used to generate the rows from 1 to $n-1$ of picture $\hat{p}$. Let us recall that for each indexes i, j, with $1 \leq i \leq n+1$ and $1 \leq j \leq m+1$ there are tiles $t_{ij} =_r t_{i+1,j}$. Then, we distinguish two cases.

Case 1: assume $i = 1$. Then by construction there exists rule r_j^1 in $R_{r,M,2}$ corresponding to such pair of tiles for $j \neq 1$ and $j \neq m+1$ and rules r_1^1, r_{m+1}^1 respectively in $R_{r,L,2}$ and $R_{r,R,2}$, such that the r-sequence $S_{2,m} = r_1^1 \cdot r_2^1 \cdots r_j^1 \cdots r_{m+1}^1$ will produce the context site picture consisting of $\hat{q}^r[1..2]$ and the replacement site consisting of the two rows $[\#, \pi(q_{11}), \cdots, \pi(q_{1m}), \#]$ and $[\#, [q_{11}\ q_{21}], \cdots, [q_{1m}\ q_{2m}], \#]$.

Case 2: assume that $1 < i < n$. Recalling that for each index j, $1 \leq j \leq m+1$ there are tiles $t_{ij} =_r t_{i+1,j}$, by construction there exists a m-type rule r_j^i in $R_{r,M,3}$ corresponding to such pair of tiles for $j \neq 1$ and $j \neq m+1$. Moreover, there exists a l-type rule $r_1^i \in R_{r,L,3}$ and a r-type rule $r_{m+1}^i \in R_{r,R,3}$ corresponding to the above pair of tiles for $j = 1$ and $j = m+1$, respectively.

It is immediate to verify that the r-sequence $S_{3,i,m} = r_1^i \cdot r_2^i \cdots r_j^i \cdots r_{m+1}^i$ produces the context site picture with rows $[\#, \pi(q_{i1}), \cdots, \pi(q_{im}), \#]$, $[\#, [q_{i1}\ q_{i+1,1}], \cdots, [q_{im}\ q_{i+1,m}], \#]$ and the replacement site picture with rows $[\#, \pi(q_{i+1,1}), \cdots, \pi(q_{i+1,m}), \#]$, $[\#, [q_{i+1,1}\ q_{i+2,1}], \cdots, [q_{i+1,m}\ q_{i+2,m}], \#]$.

Now, let us show the construction of $\hat{p}$ using the above specified rule sequences $S_{2,m}$ and $S_{3,i,m}$.

The first step will consist of the generation of the pseudo-picture $\hat{p}_0 = \hat{q}^r[1..2]$ which has been detailed above for the case $n = 1, m > 1$. Then, as a second step, the r-sequence $S_{2,m}$ is applied to picture $\hat{p}_0$ to generate picture $\hat{p}_1$. The derivation $\hat{p}_1 \rightarrow_{S_{3,1,m}} \rightarrow_{S_{3,2,m}} \cdots \rightarrow_{S_{3,i,m}} \cdots \rightarrow_{S_{3,n-1,m}} \hat{p}_n$ will produce the picture $\hat{p}_n$ such that $\hat{p}_n[1..n] = \hat{p}[1..n]$, that is $\hat{p}_n$ and $\hat{p}$ have the same n rows. By the property stated above picture $\hat{p}_n$ will have last two rows consisting of $[\#, \pi(q_{n-1,1}), \cdots, \pi(q_{n-1,m}), \#]$, $[\#, [q_{n-1,1}\ q_{n,1}], \cdots, [q_{n-1,m}\ q_{nm}], \#]$.

Finally, the last step consists of applying the r-sequence $S_{4,m} = r_1^n \cdot r_2^n \cdots r_j^n \cdots r_{m+1}^n$, where rules $r_1^n \in R_{r,L,4}$, $r_{m+1}^n \in R_{r,R,4}$ and $r_j^n \in R_{r,M,4}$ are based on pairs of tiles $t_{nj} =_r t_{n+1,j}$. Applying $S_{4,m}$ to $\hat{p}_n$ will produce the picture $\hat{p}$ as required.

Let us now show that $L(T) \subseteq L$. We prove the equivalent statement that for any canonical picture $\hat{q} \in L(T)$ there exists a picture $\hat{p}$ such that

$B_{2,2}(\hat{p}) \subseteq \Theta \bigwedge \hat{q} = \pi(\hat{p})$ ("full property"). Here, for the sake of simplicity, we consider π extended by the mapping $\# \to \#$.

The proof will focus on the case when $n, m > 1$. It will be by induction on the number of rows, showing that $\hat{q}$ is built by growing a series of pseudo-canonical pictures, where each pseudo-canonical picture q_i with i rows generates $i-1$ rows of the projection of $\hat{p}$ i.e. there exists p_i such that $B_{2,2}(p_i) \subseteq \Theta \wedge q_i^r[1..i-1] = \pi(p_i)$("weak property"). We finally state that the target full property is obtained when eventually producing $\hat{q}$ by applying the r-sequence adding the bottom border.

The base case of induction, and the only possible starting steps in T, is the building of q_i having $i = 3$ rows, obtained by starting from the empty picture in I and by applying rules in two phases:

– a derivation sequence of c-sequences over the empty picture ; $S_1 \to S_{2,1} \to S_{2,2} \cdots \to S_{2,m} \to S_3$ where rules of S_1 are in $R_{c,U,2}$, rules of $S_{2,k}$ in $R_{c,U,3}$ and rules of S_3 are in $R_{c,U,4}$, producing the pseudo-canonical picture q_0.

– then a r-sequence over picture q_o

where combining rules of the r-sequence are from $R_{r,L,2}, R_{r,M,2}, R_{r,R,2}$, in that order.

The definition of rule sets involved with the second phase, and the requirement of combining of rules in row sequences, imply that $q_i^r[1..2]$ is the projection of a two rows partial picture covered by tiles from Θ, thus satisfying the (weak) property being induced.

Over pictures with such 3 rows, or more rows, only rules from $R_{r,\{L,M,R\},3}$ may be applied (before adding bottom border with rules from $R_{r,\{L,M,R\},4}$). This is our induction step: applying a row sequence of rules from $R_{r,\{L,M,R\},3}$ to a picture q_i with i rows, we obtain a picture q_j with $j = i+1$ rows, where $q_i^r[1..i-1] = q_j^r[1..i-1]$ satisfies the property, while the bottom $q_i^r[i]$ is replaced in q_j by two new rows. $q_j^r[1..j-1]$ satisfies the property because the added set of tiles $B_{2,2}(q_j^r[i-1..j-1])$ on it are the projection of two rows covered by tiles from Θ, as a consequence of the definition of the applied rules.

A similar reasoning goes when instead we may apply a rule sequence from $R_{r,\{L,M,R\},4}$, obtaining a canonical picture $\hat{q}_j$ with $j = i+1$ rows from a picture q_i with i rows. This time, the definition of the rules being applied shows that the added set of tiles $B_{2,2}(\hat{q}_j^r[i-1..j])$, which includes the bottom border, are projection of a picture covered by tiles from Θ, while by induction the same holds for $\hat{q}_j^r[1..i-1]$, therefore completing our proof.

□

We now consider how to build palindromic languages by using TRuS systems.

Proposition 5.2. *Let L be the palindromic column language defined as $L = \{p \mid p = q \ominus q' : q' = mirror(q), q$ is a picture over alphabet $\Sigma\}$. Then $L \in \mathcal{L}(TRuS)$.*

Proof. By $x_1, x_1', x'', x_2, x_2', x_3, x_3'$ we will denote the generic symbol variables over alphabet Σ, while $\Sigma' = \Sigma \cup \{c, d\}$, and $\Delta = \{\#, \diamond\}$ as usual. We construct a TRuS system $T = (I, R, \Sigma')$ that generates language L. For every $x_1 \in \Sigma$, I contains the canonical picture $\hat{p_1}$ over Σ where the picture p_1 is $\boxed{\begin{matrix} x_1 \\ x_1 \end{matrix}}$. Clearly, $p_1 \in L$. We define R consisting of the set of rules defined below where $R = R_{c,COL} \cup R_{r,INTER1} \cup R_{r,INTER2} \cup R_{r,INTER3} \cup R_{r,END}$.

$$R_{c,COL} = \left\{ \begin{array}{rcl} r_1 & = & \left(\begin{array}{c|c} \# & \# \\ \# & x_1 \end{array}\right) \rightarrow \begin{matrix} \# & \# \\ x_1 & x_1' \end{matrix}, \\ r_2 & = & \left(\begin{array}{c|c} \# & x_1 \\ \# & x_1 \end{array}\right) \rightarrow \begin{matrix} x_1 & x_1' \\ x_1 & x_1' \end{matrix}, \\ r_3 & = & \left(\begin{array}{c|c} \# & x_1 \\ \# & \# \end{array}\right) \rightarrow \begin{matrix} x_1 & x_1' \\ \# & \# \end{matrix}, \\ r_4 & = & \left(\begin{array}{c|c} \# & \# \\ x_1 & x_1' \end{array}\right) \rightarrow \begin{matrix} \# & \# \\ x_1' & x_1'' \end{matrix}, \\ r_5 & = & \left(\begin{array}{c|c} x_1 & x_1 \\ x_1' & x_1' \end{array}\right) \rightarrow \begin{matrix} x_1' & x_1'' \\ x_1' & x_1'' \end{matrix}, \\ r_6 & = & \left(\begin{array}{c|c} x_1 & x_1' \\ \# & \# \end{array}\right) \rightarrow \begin{matrix} x_1' & x_1'' \\ \# & \# \end{matrix}, \end{array} \right.$$

$$R_{r,INTER1} = \left\{ \begin{array}{rcl} r_7 & = & \begin{matrix} \# & x_1 \\ \# & x_1 \end{matrix} \rightarrow \begin{matrix} \diamond & c \\ \# & x_1 \end{matrix}, \\ r_8 & = & \begin{matrix} x_1 & \# \\ x_1 & \# \end{matrix} \rightarrow \begin{matrix} c & \# \\ x_1 & \# \end{matrix}, \\ r_9 & = & \begin{matrix} x_1 & x_1' \\ x_1 & x_1' \end{matrix} \rightarrow \begin{matrix} c & c \\ x_1 & x_1' \end{matrix}, \\ r_{10} & = & \begin{matrix} x_1' & \# \\ x_1' & \# \end{matrix} \rightarrow \begin{matrix} c & \# \\ x_1' & \# \end{matrix}, \end{array} \right.$$

$$R_{r,INTER2} = \begin{cases} r_9 = \frac{\# \quad x_1}{\diamond \quad c} \rightarrow \begin{matrix} \# & x_3 \\ \diamond & d \end{matrix}, \\ r_{10} = \frac{x_1 \quad \#}{c \quad \#} \rightarrow \begin{matrix} x_3 & \# \\ d & \# \end{matrix}, \\ r_{11} = \frac{x_1 \quad x_1'}{c \quad c} \rightarrow \begin{matrix} x_3 & x_3' \\ d & d \end{matrix}, \\ r_{12} = \frac{x_1' \quad \#}{c \quad \#} \rightarrow \begin{matrix} x_3' & \# \\ d & \# \end{matrix}, \end{cases}$$

$$R_{r,INTER3} = \begin{cases} r_{13} = \frac{\# \quad x_3}{\diamond \quad d} \rightarrow \begin{matrix} \diamond & c \\ \# & x_3 \end{matrix}, \\ r_{14} = \frac{x_3 \quad \#}{d \quad \#} \rightarrow \begin{matrix} c & \# \\ x_3 & \# \end{matrix}, \\ r_{15} = \frac{x_3 \quad x_3'}{d \quad d} \rightarrow \begin{matrix} c & c \\ x_3 & x_3' \end{matrix}, \\ r_{16} = \frac{x_3' \quad \#}{d \quad \#} \rightarrow \begin{matrix} c & \# \\ x_3' & \# \end{matrix}, \end{cases}$$

$$R_{r,END} = \begin{cases} r_{17} = \frac{\# \quad x_1}{\diamond \quad c} \rightarrow \begin{matrix} \# & x_2 \\ \# & x_2 \end{matrix}, \\ r_{18} = \frac{x_1 \quad \#}{c \quad \#} \rightarrow \begin{matrix} x_2 & \# \\ x_2 & \# \end{matrix}, \\ r_{19} = \frac{x_1 \quad x_1'}{c \quad c} \rightarrow \begin{matrix} x_2 & x_2' \\ x_2 & x_2' \end{matrix}, \\ r_{20} = \frac{x_1' \quad \#}{c \quad \#} \rightarrow \begin{matrix} x_2' & \# \\ x_2' & \# \end{matrix}, \end{cases}$$

We denote the two possible c-sequences produced by the rules in $R_{c,COL}$ as $S_{c_1} = r_1.r_2.r_3$ and $S_{c_2} = r_4.r_5.r_6$. We denote the two r-sequences produced by set of rules in $R_{r,INTER1}$ by $S_{r_1} = r_7.r_8$ and $S_{r_1,i} = r_7.(r_9)^i.r_{10}$ where for any integer i, r_9 can be iterated i times. Similarly, we denote each pair of r-sequences which can only be produced by the set of rules in $R_{r,INTER2}, R_{r,INTER3}, R_{END}$ by $S_{r_2} = r_{11}.r_{12}$ and $S_{r_2,j} = r_{11}.(r_{13})^j.r_{14}$, $S_{r_3} = r_{15}.r_{16}$ and $S_{r_3,k} = r_{15}.(r_{17})^k.r_{18}$, $S_{r_4} = r_{19}.r_{20}$ and $S_{r_4,l} = r_{19}.(r_{21})^l.r_{22}$ respectively.

By definition 5.1, 5.2 we can see that only rules in the same set of rules in $\{R_{c,COL}, R_{r,INTER1}, R_{r,INTER2}, R_{r,INTER3}, R_{END}\}$ can be combined to form specific α-sequences, with $\alpha \in \{c, r\}$, while no two rules in distinct sets can be combined to form rule sequences that can be applied to pictures.

In the following we verify that the above defined set of rules form derivation sequences that can be uniquely applied over initial picture to generate only column palindromic pictures. It can be seen that the α-sequences that can be applied to $\hat{p} \in I$ are the c-sequences S_{c_1} followed by S_{c_2} where each of the sequence can be iterated $m, m'($ for some arbitrary integers $m, m')$ times generating $\hat{p}_m, \hat{p}_{m'}$ respectively.

We illustrate in figure 5.4 the derivation sequence d_1 over the initial picture. We can see that after the first application of S_{c_1} at the left most column as context site the iteration of the c-sequences S_{c_1}, S_{c_2} can continue arbitrarily many times, where S_{c_2} can be applied at any column as context site generating canonical palindromic pictures for each iterated number of columns. Thus d_1 generates all canonical $2 \times m$ column palindromic pictures, where :

$$d_1 = \hat{p} \rightarrow^{m}_{S_{c_1}} \hat{p}_m \rightarrow^{m'}_{S_{c_2}} \hat{p}_{m'}.$$

$$\hat{p} \;\; \rightarrow_{(S_{c_1})^m} \;\; \hat{p}_m \qquad \begin{array}{|cccccc|}\hline \# & \# & \# & \cdots & \# & \# \\ \# & \boldsymbol{x_1} & x'_1 & \cdots & x'_1 & \# \\ \# & \boldsymbol{x_1} & x'_1 & \cdots & x'_1 & \# \\ \# & \# & \# & \cdots & \# & \# \\ \hline \end{array}$$

$$\rightarrow_{(S_{c_2})^{m'}} \;\; \hat{p}_{m'} \qquad \begin{array}{|cccccc|}\hline \# & \# & \# & \cdots & \# & \# \\ \# & \boldsymbol{x_1} & \boldsymbol{x'_1} & \cdots & x''_1 & \# \\ \# & \boldsymbol{x_1} & \boldsymbol{x'_1} & \cdots & x''_1 & \# \\ \# & \# & \# & \cdots & \# & \# \\ \hline \end{array}$$

Figure 5.4: Application of derivation sequence d_1 (iterative Context sites, in boldface).

Also, we can see that the only possible other derivation sequence over picture $\hat{p}$ is :

$$d_2 = \hat{p} \rightarrow_{S_{r_1}} \hat{p'} \;\; (\rightarrow_{S_{r_2}} \hat{p'_1} \rightarrow_{S_{r_3}} [\hat{p'}])^x \rightarrow_{S_{r_4}} \hat{p''}$$

where by $[\hat{p'}]$ we denote the pseudo-canonical picture with strictly same context-site (r-sequences in R) as that of $\hat{p'}$, $0 \leq x \leq n$ denotes the number of iterations of the parenthesized derivation sequences.

We now verify that d_2 generates all column palindromic pictures of sizes $2n \times 1$. It can be seen that when only r-sequences are applied over $\hat{p}$ of size 2×1 it can only increase the number of rows and the number of

column remains 1. S_{r_1} of d_2 is the only possible other r-sequence over $\hat{p}$ apart from the c-sequence S_{c_1} of d_1, where S_{r_1} generates pseudo-canonical picture $\hat{p'}$ such that $\hat{p'}^r[2] = [\diamond, c, \#]$, i.e. the "middle row" of the picture "marked" with the special symbols $\diamond$, c.

Then the only possible r-sequences that can be applied over $\hat{p'}$ are S_{r_2} or S_{r_4}. We can see that S_{r_4} applied over it generates canonical 4×1 column palindromic picture $\hat{p''}$ by replacing the marked "middle row" of $\diamond, c, \#$ with the replacement sites $[\#, x_2, \#]$,$[\#, x_2, \#]$ which are palindromic. Whereas S_{r_2} when applied over $\hat{p'}$ at context site ("middle row") generates $\hat{p'_1}$ a pseudo-canonical picture with the replacement site $\hat{p'_1}^r[3] = [\#, x_3, \#]$, $\hat{p'_1}^r[4] = [\diamond, d, \#]$ which is uniquely followed by the r-sequence S_{r_3} generating $[\hat{p'}]$ with the replacement site $[\hat{p'_1}]^r[4] = [\diamond, c, \#]$, $\hat{p'_1}^r[5] = [\#, x_3, \#]$. Thus derivation sequence d_2 can iterate for n number of times generating a palindromic row sequence i.e. of $2n$ number of rows at each iteration.

The only possible derivation that can be applied to any picture $\hat{p}_u \in \{\hat{p}_m \cup \hat{p}_{m'}\}$ consists of d_3 :

$$d_3 = \hat{p}_u \rightarrow_{S_{r_1,i}} \hat{p}_{u_1}(\rightarrow_{S_{r_2,j}} \hat{p}_{u_2} \rightarrow_{S_{r_3,k}} [\hat{p}_{u_1}])^y \rightarrow_{S_{r_4,l}} \hat{p}_{u_4}$$

where $[\hat{p}_{u_1}]$ denotes the pseudo-canonical picture with strictly same context sites as that of $\hat{p}_{u_1}$, $0 \leq y \leq n$ denotes the number of iteration of the parenthesized derivation sequences. Figure 5.5 describes the generation of derivation sequence d_3 applied over $\hat{p''}$, i.e. all $2 \times m$ canonical palindromic pictures for $m < 1$ generated by d_1.

We can see by the illustration given in figure 5.5 that the derivation sequence d_3 generates canonical picture only with palindromic rows on each sequence application (iteratively) and incrementing the row by $2n$ for every fixed number of columns in the chosen picture to apply as described in the the generation of d_2.

Now, let us show by induction on size that column palindromic pictures are generated by the system. i.e. $L \subseteq L(T)$. It is obvious that the pictures in L are of size $2n \times m$. Since the pictures in L of size 2×1 are in I (initial language) it follows that they are included in $L(T)$ with bordered versions in I. Given a column palindromic picture of size $(2n \times m)$ with arbitrary m in $L(T)$, in the above definition of the rules in R we can generate the column palindromic pictures of size $2(n + 1) \times m$.

To complete the proof we have to show that $L(T) \subseteq L$. We can verify that the above defined rules form sequences which are distinctly applied over the picture and the derivation sequences are ambiguously framed to generate all size of column palindromic pictures. Thus we can say that

$$\hat{p}_u \rightarrow_{S_{r_1,i}} \hat{p}_{u_1} \begin{vmatrix} \# & \boldsymbol{x_1} & \cdots & \boldsymbol{x'_1} & \# \\ \diamond & \boldsymbol{c} & \cdots & \boldsymbol{c} & \# \\ \# & x_1 & \cdots & x'_1 & \# \end{vmatrix} (d'_3)^y \rightarrow_{S_{r_4,l}} \begin{vmatrix} \# & x_1 & \cdots & x'_1 & \# \\ \# & x_2 & \cdots & x'_2 & \# \\ \# & x_2 & \cdots & x'_2 & \# \\ \# & x_1 & \cdots & x'_1 & \# \end{vmatrix}$$

$$d'_3 = \hat{p}_{u_1} \rightarrow_{S_{r_2,j}} \begin{vmatrix} \# & x_1 & \cdots & x'_1 & \# \\ \# & \boldsymbol{x_3} & \cdots & \boldsymbol{x'_3} & \# \\ \diamond & \boldsymbol{d} & \cdots & \boldsymbol{d} & \# \\ \# & x_1 & \cdots & x'_1 & \# \end{vmatrix} \rightarrow_{S_{r_3,k}} \hat{p}_{u_1} \begin{vmatrix} \# & x_1 & \cdots & x'_1 & \# \\ \# & \boldsymbol{x_3} & \cdots & \boldsymbol{x'_3} & \# \\ \diamond & \boldsymbol{c} & \cdots & \boldsymbol{c} & \# \\ \# & x_3 & \cdots & x'_3 & \# \\ \# & x_1 & \cdots & x'_1 & \# \end{vmatrix}$$

Figure 5.5: Application of derivation sequence d_3 (iterative Context sites, in boldface).

no other pictures are possible starting from a picture $p \in L$, thus showing that the closure of L under rules in R. Thus we have $L = L(T)$.

□

5.3 Comparing TRuS Systems with Wang Systems

In this section we show that the computational power of the subclass sTRuS of TRuS systems is the same as that of recognizable pictures languages.

We will show this result by using a known result stating the equivalence of the class of picture languages defined by Wang systems with the family of picture languages recognized by tiling systems [4].

We can now state the main comparison proved in this section, and then we will be able to show the main theorem, summarizing the results.

Proposition 5.3. *$\mathcal{L}$(sTRuS) $\subseteq$ $\mathcal{L}$(WS) .*

Proof. Let $T = (I, R, \Sigma)$ be a simple tiling system. Let $L = L(T)$.

Then we build a Wang system $W = (\Gamma, Q, T_w)$ and a projection $\pi : \Gamma \rightarrow \Sigma$ such that $L = \pi(L(W))$. Recalling that Wang languages are closed under projection [4], our result follows. The following notions will be used to define the set T_w of Wang tiles of the system.

Given a column rule: $r = \begin{smallmatrix} a & c \\ b & d \end{smallmatrix} \rightarrow \begin{smallmatrix} e & f \\ g & h \end{smallmatrix}$, the *composition of column rule* r, denoted by $comp(r)$ is the tile $\begin{smallmatrix} a & e \\ b & g \end{smallmatrix}$. Moreover $dom[r,2] = (b,d),(g,h)$ denotes the pair of bottom row dominoes of the two tiles of the rule, while $dom[r,1] = (a,c),(e,f)$ denotes the pair of upper row dominoes of the two tiles of the rule.

Similarly given a row rule: $r = \frac{a\ b}{c\ d} \rightarrow \begin{smallmatrix} e & f \\ g & h \end{smallmatrix}$, then $comp(r)$ is the tile $\begin{smallmatrix} a & b \\ e & f \end{smallmatrix}$. Moreover $dom[r,1] = (a,c),(e,g)$ denotes the pair of left column dominoes of the two tiles of the rules, while $dom[r,2] = (b,d),(f,h)$ denotes the pair of right column dominoes of the two tiles of the rule.

Let us recall that given two column rules r_1, r_2, then the c-sequence $r_1 \cdot r_2$ is defined if and only if $dom[r_1,2] = dom[r_2,1]$. Similarly for a pair r_1, r_2 of rules, then the r-sequence $r_1 \cdot r_2$ is defined if and only if $dom[r_1,2] = dom[r_2,1]$. Thus the above notions will be used to associate to sequences of rules to a set Wang tiles that can be correctly tied together.

Now Wang tiles in T_w are listed below.

(i) *Left border column rules*

For each column rule: $r = t_1 \rightarrow t_2 = \begin{smallmatrix} \# & \# \\ \# & \# \end{smallmatrix} \rightarrow \begin{smallmatrix} \# & \# \\ a & \# \end{smallmatrix}$, and each row rule $r' = comp(r) \rightarrow t'$, then we associate the Wang tile: $w_r = \begin{smallmatrix} & B & \\ B & (t',c) & B \\ & comp(r) & \end{smallmatrix}$. Then, if there exist the two row rules: $r_1 = \frac{\# \#}{\# a} \rightarrow \begin{smallmatrix} \# & a' \\ \# & \# \end{smallmatrix}$, $r_2 = \frac{\# \#}{a \ \#} \rightarrow \begin{smallmatrix} a' & \# \\ \# & \# \end{smallmatrix}$, then we associate the Wang tile $w_r = \begin{smallmatrix} & B & \\ B & (a') & B \\ & B & \end{smallmatrix}$. For each column rule: $r = t_1 \rightarrow t_2 = \begin{smallmatrix} \# & \# \\ \# & \# \end{smallmatrix} \rightarrow \begin{smallmatrix} \# & \# \\ a & b \end{smallmatrix}$, and each row rule $r' = comp(r) \rightarrow t'$, then $w_r = \begin{smallmatrix} & B & \\ B & (t',c) & t_2 \\ & comp(r) & \end{smallmatrix}$.

(ii) *Border internal column rules*

For each column rule: $r = t_1 \rightarrow t_2 = \begin{smallmatrix} \# & \# \\ a & b \end{smallmatrix} \rightarrow \begin{smallmatrix} \# & \# \\ c & d \end{smallmatrix}$, and each row rule $r' = comp(r) \rightarrow t'$, then we associate the Wang tile: $w_r =$

$$\begin{matrix} & B & \\ t_1 & (t', c) & t_2 \\ & comp(r) & \end{matrix} .$$

(iii) *Border column right rules*

For each column rule: $r = t_1 \to t_2 = \begin{matrix} \# & \# \\ a & b \end{matrix} \to \begin{matrix} \# & \# \\ c & \# \end{matrix}$, then we associate the Wang tile: $w_r = \begin{matrix} & B & \\ t_1 & (t_2, c) & B \\ & comp(r) & \end{matrix}$.

(iv) *Row left border rules*

For each row rule with $\sigma \in \Sigma \cup \{\#\}$: $r = t_1 \to t_2 = \frac{\# \ \sigma}{\# \ a} \to \begin{matrix} \# & b \\ \# & c \end{matrix}$, then we associate the Wang tile: $w_r = \begin{matrix} & t_1 & \\ B & (t_2, r) & dom[r, 2] \\ & t_2 & \end{matrix}$

(v) *Row intermediate + right border rules*

For each row rule $\sigma, \sigma' \in \Sigma \cup \{\#\}$: $r = t_1 \to t_2 = \frac{\sigma \ \sigma'}{c \ d} \to \begin{matrix} e & f \\ g & h \end{matrix}$, if we have the right border rule $r' = t'_1 \to t'_2 = \frac{\sigma \ \#}{d \ \#} \to \begin{matrix} f & \# \\ h & \# \end{matrix}$, then we associate the Wang tile: $w_r = \begin{matrix} & t_1 & \\ dom[r, 1] & (t_2, r) & B \\ & t_2 & \end{matrix}$

Otherwise we associate the Wang tile: $w_r = \begin{matrix} & t_1 & \\ dom[r, 1] & (t_2, r) & dom[r, 2] \\ & t_2 & \end{matrix}$

(vi) *Row left and below border rules*

For each row rule with $\sigma \in \Sigma \cup \{\#\}$: $r = t_1 \to t_2 = \frac{\# \ \sigma}{\# \ a} \to \begin{matrix} \# & b \\ \# & \# \end{matrix}$, then we associate the Wang tile: $w_r = \begin{matrix} & t_1 & \\ B & (t_2, r) & dom[r, 2] \\ & B & \end{matrix}$.

(vii) *Row intermediate + right and below border rules*

For each row rule $\sigma, \sigma' \in \Sigma \cup \{\#\}$: $r = t_1 \to t_2 = \frac{\sigma\ \sigma'}{c\ d} \to \begin{matrix} e & f \\ \# & \# \end{matrix}$,

if we have the right border rule $r' = t'_1 \to t'_2 = \frac{\sigma\ \#}{d\ \#} \to \begin{matrix} f & \# \\ \# & \# \end{matrix}$,

then we associate the Wang tile: $w_r = \begin{matrix} & t_1 & \\ dom[r,1] & (t_2, r) & B \\ & B & \end{matrix}$.

Otherwise we associate the Wang tile:

$$w_r = \begin{matrix} & t_1 & \\ dom[r,1] & (t_2, r) & dom[r,2] \\ & B & \end{matrix}.$$

To conclude the construction, we define the projection π as follows:

$\pi((t,c)) = b$ if and only if $t = \begin{matrix} a & b \\ c & d \end{matrix}$. Similarly, we define $\pi((t,r))$.

Now let S_i be the c-sequence consisting of column rule of the sTRuS system T. It is immediate to show that there exists a derivation sequence of c-sequences $S_1, S_2, \cdots, S_{m-1}$ producing the first two rows of a picture $\hat{p}$ of size (n, m) if and only if the Wang tiles of the system W generate a tiling of the same two rows.

It is easy to show that there exists a r-sequence $S = r_1 \cdot r_2 \cdots r_{m-1}$ of rules in R having replacement site consisting of rows $s = [\#, a_1, a_2, \cdots, a_{m-2}, \#]$ and $s' = [\#, b_1, b_2, \cdots, b_{m-2}, \#]$ if and only if there exists a tiling of the row s by Wang tiles leaving a lower colored bordered consisting of the sequence of right tiles of rules in S.

Now, S has replacement site consisting of rows $s = [\#, a_1, a_2, \cdots, a_{m-2}, \#]$ and $s' = [\#, \#, \#, \cdots, \#]$ if and only if there exists a tiling by Wang tiles of the two rows.

These observations conclude the proof of the proposition.

□

Now the following main result can be stated:

Theorem 5.1. *$\mathcal{L}$(TS) = $\mathcal{L}$(sTRuS) $\subset$ $\mathcal{L}$(TRuS) .*

Proof. The equality in the first part of the statement is proved by the above Proposition 5.3 together with the main theorem in [4] stating that $\mathcal{L}(\text{WS}) = \mathcal{L}(\text{TS})$, and Proposition 5.1 of Section 5.2, where we proved that $\mathcal{L}(\text{TS}) \subseteq \mathcal{L}(\text{sTRuS})$.

The inequality in the second part can be showed by Proposition 5.2 of Section 5.2 together with the mentioned results in [3] about palindromic languages (recalling that $\mathcal{L}(\text{sTRuS}) \subseteq \mathcal{L}(\text{TRuS})$, by definition).

□

Chapter References

[1] P. Bonizzoni, C. Ferretti, Anthonath Roslin Sagaya Mary, and G. Mauri. Picture languages generated by assembling tiles. volume Lecture Notes in Computer Science, pages 224–235, 2009.

[2] P. Bonizzoni, C. Ferretti, Anthonath Roslin Sagaya Mary, and G. Mauri. Picture languages generated by assembling tiles. *Fundam. Inform. 110*, 1-4:77 – 93, 2011.

[3] S. Crespi Reghizzi and M. Pradella. Tile rewriting grammars and picture languages. *Theoretical Computer Science*, 340:257– 272, 2005.

[4] L. De Prophetis and V. Varricchio. Recognizability of rectangular pictures by wang systems. *Journal of Automata, Language and Combinatorics*, 2:269–288, 1997.

Chapter 6

Conclusions and Future work

6.1 Conclusions

Motivated by the bio-inspired operation Splicing and the DNA computing branch of self-assembly we have introduced two formalisms for picture languages in this thesis. These are defined by applying the similar defined string case operations by rules with dominoes and tiles over set of pictures. Due to the very nature of Picture language generation and its constraints on learning its structure and combinatorics for extending the string case hierarchy to it, our study we hope will give some refinement in the process though several questions towards it with are to be studied.

The first formalism we propose is the H-array Splicing Systems extending the splicing operation on string case to picture languages in [5], cutting columns and rows by given set of domino splicing rules and pasting by the column and row concatenation operation. The application of the rules iteratively, formalizes the picture languages which enhances the picture patterns of the language obtained. In the theoretical study of the class of H Array Splicing Systems $\mathcal{L}$(HASL) in [5] also "Simple" array splicing rules is given. And in [8, 6] more examples and counter examples of the language classes $\mathcal{L}(A_{SCO})$ and $\mathcal{L}$(SASL) is given with results analogous to the main study of $\mathcal{L}$(HASL) class. Also, the power of trees for $\mathcal{L}$(SASL) is studied, which is proved to be higher than the recognizability class.

Particularly, the closure properties of column(row) concatenation, and under various rotations is studied for the language classes $\mathcal{L}$(HASL) , $\mathcal{L}(A_{SCO})$, $\mathcal{L}$(SASL) . Then the study on HAS and its restriction languages $\mathcal{L}(A_{SCO})$ and $\mathcal{L}$(SASL) is compared with other main two-

dimensional language classes like LOC, $2RLG$ wherein it is proved to be analogous with each other and are incomparable but disjoint. Also the same incomparability results with intersection comparison with the class of languages $\mathcal{L}$(HASL) and $\mathcal{L}(A_{SCO})$ is proved to be analogous to the string case Self Cross-Over Systems and H Splicing Systems.

The Splicing Array Grammar System in [9] is introduced in section 3.4 turns out to be a powerful means of generating picture arrays. It extends the image grammar system in [3] to 2D-tabled matrix grammars. It remains to compare other picture generating mechanisms with these systems.

We have also tried to give some insights along the study for extending to higher dimensional case from picture languages. By considering Pictural Network of Evolutionary Processors defined, in section 4 we have extended the study of generation of pictures of rectangular arrays to 2D pictures of arrays not necessarily rectangular by networks of Evolutionary Processors. We have introduced contextual insertion, deletion and substitution rules for picture in the network processors in [4] and studied some examples. Extension of [7] to pictures of 3D rectangular arrays are also considered giving some interesting examples and comparisons.

Second formalism being the study on tiles, Tiling rule systems TRuS provide a new formalism for defining picture languages that is based on rules to assemble tiles. Pictures of the language are generated by iteratively applying rules: they grow a picture of size (n, m) by locating a $(2, m)$ row (or $(n, 2)$ column) context site picture where the bottom row is replaced and and a new row (or column) is added. Now, tiling rule systems generate a class of languages that properly includes the class of recognizable picture languages. Actually, the proof of the inclusion shows that pictures of a recognizable language are assembled by growing them along a border, that is by adding new rows and columns. On the contrary, pictures of TRuS languages that are not recognizable languages (see Proposition 5.2), can only be assembled by adding rows or columns properly inside pictures of smaller size. The class TRuS seems to be a powerful notion generating also $(n, n!)$ picture language for unary class given in [1]. The main results being $\mathcal{L}(\text{TS}) = \mathcal{L}(\text{sTRuS}) \subset \mathcal{L}(\text{TRuS})$. Where $\mathcal{L}$(TS) is the recognizable class of picture languages defined with the LOC class. $\mathcal{L}$(sTRuS) is called the simple Tiling Rule Systems where the picture grows only in columns or rows with tiling rules.

Thus the study on splicing pictures in column(row) with domino rules and assembling tiles for enhancing the pictures and framing language

classes is done in the thesis with the above stated results. We have proposed the two interesting formalisms with the bio-inspired operations for pictures. Interesting since the very nature of the operations and picture generation seems to be linking for the underlying study on the structure of the pictures along with the combinatorics. Also, it is of main interest to make the lacking automata theory or grammar theory of formal languages to picture languages.

6.2 Future work

The bio-operations Splicing and Self-assembling in consideration or inspired by the formalisms are themselves in vast in investigation. Firstly, we have tried to construct analogous Picture language classes $\mathcal{L}$(HASL) , $\mathcal{L}(A_{SCO})$, $\mathcal{L}$(SASL) . And that of Assembling Tiles namely, TRuS . In further study to extend is the possible variants in the formalisms and the results to the approach of string case study.

The thesis though in general has been to contribute to extending the bio-operations to picture languages also main Chomsky hierarchy of the string case to picture languages is in concern. There lies most important questions on refining the already existing study on various points in correspondence to string case as well to the very own bio-computing aspects of the formalisms.

In particular, we have the questions on connecting the HAS and TRuS , with results. Then several questions concerning the notion of tiling rule system such as closure properties remain to be investigated, as well as the comparison of this new approach with other grammar language classes like Tiling Rewriting Grammars (TRG) formalism [2] is to be done.

Chapter References

[1] P. Bonizzoni, C. Ferretti, Anthonath Roslin Sagaya Mary, and G. Mauri. Picture languages generated by assembling tiles. volume Lecture Notes in Computer Science, pages 224–235, 2009.

[2] A. Cherubini, S. Crespi Reghizzi, and M. Pradella. Regional languages and tiling: A unifying approach to picture grammars. *Lecture Notes in Computer Science*, 5162:253–264, 2008.

[3] K. S. Dersanambika, K. G. Subramanian, and Anthonath Roslin Sagaya Mary. Image splicing grammar systems. *Proceedings on Grammar Systems Week*, 2004.

[4] Anthonath Roslin Sagaya Mary Dersanambika. k. S, Subramanian. K. G. 2D and 3D pictural networks of evolutionary processors. *IWINAC, Lecture Notes in Computer Science*, 2:92–101, 2005.

[5] P. Helen Chandra, K. G. Subramanian, and D. G. Thomas. Parallel splicing on images. *Int. J. of pattern recognition and artificial intelligence*, 18(6):1071–1091, 2004.

[6] Anthonath Roslin Sagaya Mary and K. G. Subramanian. Self crossover array languages. *Adv. Int. Soft Computing*, 89:291 – 298, 2011.

[7] V. Mitrana, K. G. Subramanian, and M. Tataram. Pictural networks of evolutionary processors. *RomJIST*, 6(1-2):189–199, 2003.

[8] K. G. Subramanian and Anthonath Roslin Sagaya Mary. On special forms of splicing on arrays and graphs. *Triangle*, 2011.

[9] K. G. Subramanian, Anthonath Roslin Sagaya Mary, and K. S. Dersanambika. Splicing array grammar systems. *ICTAC*, pages 125–135, 2005.

Bibliography

[1] M. Anselmo, D. Giammarresi, and M. Madonia. New operations and regular expressions for two-dimensional languages over one-letter alphabet. *Theoretical Computer Science*, 340:408 – 431, 2005.

[2] M. Anselmo and M. Madonia. Deterministic two-dimensional languages over one-letter alphabet. *Lecture Notes in Computer Science*, 4728:147–159, 2007.

[3] A. Bertoni, M. Goldwurm, and V. Lonati. On the complexity of unary tiling-recognizable picture lanaguages. *Lecture Notes in Computer Science*, 4393:381–392, 2007.

[4] P. Bonizzoni, C. Ferretti, Anthonath Roslin Sagaya Mary, and G. Mauri. Picture languages generated by assembling tiles. volume Lecture Notes in Computer Science, pages 224–235, 2009.

[5] P. Bonizzoni, C. Ferretti, Anthonath Roslin Sagaya Mary, and G. Mauri. Picture languages generated by assembling tiles. *Fundam. Inform. 110*, 1-4:77 – 93, 2011.

[6] J. Castellanos, Mart. Cín-Vide, V. Mitrana, and C. Sempere. Solving NP-complete problems with networks of evolutionary processors. *LNCS*, 2084, Springer-Verlag:621–628, 2001.

[7] J. Castellanos, Mart. Cín-Vide, V. Mitrana, and J. Sempere. Networks of evolutionary processors. *Acta Informatica*, 39:517–529, 2003.

[8] R. Ceterchi and K. G. Subramanian. Simple circular splicing systems. *Romanian Journal of Inform. Sci. and Tech.*, 6:121–134, 2003.

[9] A. Cherubini, S. Crespi Reghizzi, and M. Pradella. Regional languages and tiling: A unifying approach to picture grammars. *Lecture Notes in Computer Science*, 5162:253–264, 2008.

[10] S. Crespi Reghizzi and M. Pradella. Tile rewriting grammars and picture languages. *Theoretical Computer Science*, 340:257– 272, 2005.

[11] E. Csuhaj-Varjú. Grammar systems: 12 years, 12 problems (short version),. In Freund, R. and Kelemenov, A. (Eds.) Proceedings of the International Workshop on Grammar Systems, Silesian University, Opava.:77–92, 2000.

[12] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, and Gh. Paun. A grammatical approach to distribution and cooperation. *Grammar systems*, Gordon and Breach Science Publishers, 1994.

[13] J. Dassow and V. Mitrana. Self cross-over systems. in : Computing with bio - molecules: Theory and experiments. *(Eds.) Gh. Paun, Discrete Mathematics and Theoretical Computer Science*, Springer-Verlag:pp. 283 – 294, 1998.

[14] J. Dassow, Gh. Păun, and A. Salomaa. *Grammars with controlled derivations, In "Handbook of Formal Languages"*. Springer Verlag, 1997.

[15] L. De Prophetis and V. Varricchio. Recognizability of rectangular pictures by wang systems. *Journal of Automata, Language and Combinatorics*, 2:269–288, 1997.

[16] K. S. Dersanambika and K. Krithivasan. Recognizability of 3D rectangular pictures. *Paper presented at the 12th International Conference of Jangeon Mathematical Society, University of Mysore*, Dec. 2003.

[17] K. S. Dersanambika, K. G. Subramanian, and Anthonath Roslin Sagaya Mary. Image splicing grammar systems. *Proceedings on Grammar Systems Week*, 2004.

[18] Anthonath Roslin Sagaya Mary Dersanambika. k. S, Subramanian. K. G. 2D and 3D pictural networks of evolutionary processors. *IWINAC, Lecture Notes in Computer Science*, 2:92–101, 2005.

[19] R. Freund. Splicing systems on graphs. *Proc. Intelligence in Neural and Biological Systems*, IEEE Press:189–194, 1995.

[20] R. Freund. Array grammar systems. *Journal of Automata, Languages and Combinatorics*, 5,1:13–29, 2000.

[21] Rozenberg. G Gh. Paun and Salomaa. A. Computing by splicing. *Theoretical Computer Science*, 168:321–336, 1996.

[22] D. Giammarresi and A. Restivo. *Two-dimensional languages, In "Handbook of Formal Languages"*, volume Springer Verlag,3. Springer Verlag, Vol.3, 1997.

[23] T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. *Bull. Math. Biology*, 49:737 –759, 1987.

[24] P. Helen Chandra, K. G. Subramanian, and D. G. Thomas. Parallel splicing on images. *Int. J. of pattern recognition and artificial intelligence*, 18(6):1071–1091, 2004.

[25] P. Helen Chandra, K. G. Subramanian, D. G. Thomas, and D. L. Van. A note on parallel splicing on images. In *IWCIA-2001*, volume Proceedings of the 8th International Workshop on Combinatorial Image Analysis, 2001.

[26] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory,*. Languages and Computation, Reading, MA,, 1979.

[27] J. Kari and C. Moore. New results on alternating and nondeterministic two-dimensional finite-state automata. *Lecture Notes in Computer Science*, 2010:396–406, 2001.

[28] K. Krithivasan. Splicing systems-the graph model. pages 33–59, 1998.

[29] K. Krithivasan, V.T. Chakaravarthy, and R. Rama. Array splicing systems. *Lecture Notes in Computer Science*, Control Cooperation and Combinatorics", Eds. Gh.Paun and A. Salomaa, 1218, Springer Verlag:346–365, 1997.

[30] V. Martin-Vide, C. Mitrana, M. J. Perez-Jimenez, and F. Sancho-Caparrini. Hybrid networks of evolutionary processors, genetic and evolutionary computation conference (gecco 2003). *Lecture Notes in Computer Science*, 2723, Springer Verlag, Berlin:401–412, 2003.

[31] Anthonath Roslin Sagaya Mary and K. G. Subramanian. Self crossover array languages. *Adv. Int. Soft Computing*, 89:291 – 298, 2011.

[32] M. Mateescu, Gh. Păun, G. Rozenberg, and A. Salomaa. Simple splicing systems. *Discrete Applied Math. 84*, 84:145–163, 1998.

[33] V. Mitrana. contextual insertion and deletion. *The publishing of house of the Romanian Academic*, Mathematical Linguistre and related topic(Gh.Paun ed):271–278, 1994.

[34] V. Mitrana, K. G. Subramanian, and M. Tataram. Pictural networks of evolutionary processors. *RomJIST*, 6(1-2):189–199, 2003.

[35] R. Narasimhan. Labelling schemata and syntactic description of pictures. *Information and Control*, 7:151–179, 1964.

[36] M. Nivat, A. Saoudi, K. G. Subramanian, R. Siromoney, and V. R. Dare. Puzzle grammars and context-free array grammars. *IJPRAI*, 5:663–675, 1992.

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

[38] Y. Sakakibara and C. Ferretti. Splicing on tree-like structures. *Theoretical Computer Science*, 210(2):227–243, 1999.

[39] D. Simplot. A characterization of recognizable picture languages by tilings by finite sets. *Theoretical Computer Science*, 218:297–323, 1999.

[40] G. Siromoney, R. Siromoney, and K. Krithivasan. Abstract families of matrices and picture languages, *computer graphics and image processing* 1 (1972) 234 - 307.

[41] R. Siromoney, K. G. Subramanian, and K. Rangarajan. Parallel sequential rectangular arrays with tables. *Inter. J. Computer Math.*, 6:143–158, 1977.

[42] K. G. Subramanian. *Studies in array languages*. PhD thesis, University of Madras, 1979.

[43] K. G. Subramanian and Anthonath Roslin Sagaya Mary. On special forms of splicing on arrays and graphs. *Triangle*, 2011.

[44] K. G. Subramanian, Anthonath Roslin Sagaya Mary, and K. S. Dersanambika. Splicing array grammar systems. *ICTAC*, pages 125–135, 2005.

[45] K. G. Subramanian, R. Siromoney, V. R. Dare, and A. Saoudi. Basic puzzle languages. *Int. J. Pattern Recognition and Artificial Intelligence*, 9:763–775, 1995.

[46] K. G. Subramanian, R. Siromoney, V. R. Dare, and A. Saoudi. Basic puzzle languages. *IJPRAI*, 9:763–775, 1995.

[47] P. S. Wang. Array grammars, patterns and recognizers. *World Scientific Pub. Co.*, 1989.

[48] P. S. Wang and C. Cook. A chomsky hierarchy of isotonic array grammars and languages. *Computer Graphics and Image Processing*, 8:144 – 152, 1978.

[49] E. Winfree. Algorithmic self-assembly of DNA: theoretical motivations and 2D assembly experiments. *Journal of Biomolecular Structures and Dynamics*, 11:263–270, 2000.